Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної системи ресторану з використанням Spring Framework

Виконав: студент IV курсу, групи СП-41 спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Ратишин М.О.
(підпис) (прізвище та ініціали)

Керівник _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Стадник Н.Б.
(підпис) (прізвище та ініціали)

Тернопіль 2023

АНОТАЦІЯ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41, 2023 рік. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 82 с., 47 рис., 2 табл., 2 додатка.

Ця кваліфікаційна робота присвячена розробці інформаційної системи для ресторану з використанням Spring Framework. Основною метою роботи було створення ефективної та функціональної системи, яка допомагатиме автоматизувати процеси управління та обслуговування ресторану.

У роботі було використано Spring Framework - популярний фреймворк для розробки Java-додатків. Використання Spring дозволяє забезпечити ефективну інтеграцію компонентів системи, забезпечити керування залежностями, а також спростити розробку та тестування.

Основними функціями розробленої інформаційної системи є керування замовленнями, управління меню та інвентарем, а також забезпечення зручного інтерфейсу для клієнтів ресторану.

У процесі розробки було використано принципи ООП (об'єктно-орієнтованого програмування) та патерни проектування для створення гнучкої та розширюваної системи. Базою даних для системи використовувався PostgreSQL, який забезпечує збереження та доступ до різних типів інформації про ресторан.

Ключові слова: ресторан, інформаційна система, Spring Framework, керування замовленнями, меню, об'єктно-орієнтоване програмування, PostgreSQL.

ANNOTATION

Bachelor's Degree Qualification Work in the field of 121 - Software Engineering. Ternopil National Technical University named after Ivan Puluj, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, SP-41 group, year 2023. The explanatory note for the Bachelor's Degree Qualification Work includes: 82 pages, 47 figures, 2 tables, 2 appendices.

This qualification work is dedicated to the development of an information system for a restaurant using the Spring Framework. The main goal of the work was to create an efficient and functional system that would help automate the management and service processes of the restaurant.

The Spring Framework, a popular framework for Java application development, was utilized in this work. The use of Spring allows for effective integration of system components, dependency management, and simplification of development and testing.

The developed information system's key functionalities include order management, menu and inventory management, and providing a user-friendly interface for restaurant customers.

During the development process, object-oriented programming (OOP) principles and design patterns were employed to create a flexible and scalable system. PostgreSQL was used as the database for the system, ensuring storage and access to various types of restaurant-related information.

Keywords: restaurant, information system, Spring Framework, order management, menu, object-oriented programming, PostgreSQL.

ЗМІСТ

ВСТУП

У сучасному світі, де технології проникають у всі сфери життя, програмне забезпечення є невід'ємною частиною багатьох галузей. Ресторанний бізнес не є винятком, і його розвиток потребує ефективного використання інформаційних технологій. З метою поліпшення управління та оптимізації робочих процесів, розробка системи ресторанного управління стає надзвичайно актуальною.

Метою даної дипломної роботи є розробка інтегрованої системи ресторанного управління, яка надасть зручні та ефективні інструменти для замовлення страв, управління столами та перегляду статистики замовлень. Наша система буде спрощувати процес обслуговування клієнтів, забезпечувати точність розрахунків та забезпечувати підтримку управління рестораном.

Для досягнення поставленої мети використовується сучасний стек технологій, що включає Java та Spring Framework для розробки бекенду, PostgreSQL як базу даних, а також React.js для розробки фронтенду. Цей стек дозволить нам створити потужну та масштабовану систему, забезпечуючи гнучкість, швидкість та надійність у роботі з програмним забезпеченням.

В даний час існує попит на інтегровані системи ресторанного управління, які забезпечують комплексний підхід до автоматизації процесів. Отже, результати цієї роботи матимуть практичне значення для ресторанних закладів, допомагаючи їм покращити ефективність та задовольнити клієнтів. Розробка цієї системи є актуальним завданням, що вирішує проблему ручного управління, помилок та затримок у обслуговуванні.

У даному дипломному проекті будуть розглянуті ключові функціональні можливості системи ресторанного управління. Зокрема, розроблено модуль для перегляду столів, що дозволяє ефективно керувати їх станом та зайнятістю, а також відображати поточний чек та меню для замовлення страв. Крім того, система надає можливість здійснювати замовлення та друкувати чеки, що спрощує процес обслуговування та забезпечує точність розрахунків. Також,

реалізовано функціонал для перегляду готового меню та статистики замовлень, що дозволяє аналізувати роботу закладу та приймати відповідні управлінські рішення.

Отримані результати даного дипломного проекту є вагомим внеском у розвиток інформаційних технологій у ресторанному бізнесі. Вони можуть бути використані для подальшого вдосконалення систем ресторанного управління, забезпечуючи більш ефективну та зручну роботу для власників закладів та їх клієнтів.

У подальших розділах дипломної роботи будуть детально розглянуті процеси аналізу вимог, проєктування, конструювання та тестування системи ресторанного управління. В аналізі вимог будуть визначені основні потреби та вимоги, які мають задовольнятись, а також виконувати стандарти та норми ресторанного бізнесу. На основі цих вимог буде проведено проєктування системи, розроблено структуру бази даних, визначено логіку роботи та взаємодії різних модулів.

Конструювання системи буде включати розробку бекенду та фронтенду, використовуючи Java, Spring Framework та React.js відповідно. Буде надано детальний опис архітектури системи, розглянуті основні компоненти та їх взаємодію. Також будуть розглянуті методи тестування, які допоможуть перевірити функціональність та надійність системи.

Висновки роботи будуть містити оцінку досягнутих результатів, аналіз переваг та недоліків системи ресторанного управління, а також пропозиції щодо подальшого розвитку та вдосконалення. Також буде наведений перелік джерел посилань, використаних під час роботи, що підтверджує достовірність та наукову обґрунтованість роботи.

1 АНАЛІЗ ВИМОГ

1.1 Актуальність програмного забезпечення

Актуальність теми обумовлена швидким розвитком ресторанного бізнесу та зростанням потреб клієнтів у зручних та ефективних способах управління рестораном. Завдяки розширенню використання сучасних технологій, ресторанні заклади мають можливість оптимізувати свою роботу, покращити обслуговування клієнтів та збільшити ефективність управління.

Традиційні методи управління рестораном, такі як ручне складання меню, реєстрація замовлень на паперових бланках та ведення розрахунків вручну, є часо- та працезатратними. Крім того, такий підхід зумовлює можливість помилок та незручностей в роботі персоналу. Тому ресторанним закладам потрібна сучасна система управління, яка спростить процеси, забезпечить точність та швидкість обробки замовлень, а також надасть зручні інструменти для аналізу та планування.

Розробка системи ресторанного управління на базі Java та Spring Framework, з використанням PostgreSQL та React.js, є актуальною, оскільки ці технології широко використовуються у сфері програмування та мають великий потенціал для реалізації таких систем. Їх застосування дозволить створити надійний та функціональний продукт, який зможе задовольнити потреби ресторанних закладів та сприяти їх успішній роботі.

Крім того, зростання популярності онлайн-замовлень та доставки їжі робить необхідність ефективної системи управління рестораном ще більш актуальною. Завдяки нашій розробленій системі, ресторанні заклади зможуть простіше та швидше обробляти замовлення, відслідковувати їх стан, керувати складом і доставкою. Це дозволить забезпечити високу якість обслуговування та задоволення потреб клієнтів.

Крім функціональності для ресторанів, наша система також надає можливість аналізувати статистику замовлень, що дозволяє ресторанним закладам

збирати цінні дані щодо популярності страв, попередніх замовлень клієнтів та інших факторів. Ця інформація може бути використана для удосконалення меню, прогнозування попиту та планування стратегій розвитку.

Актуальність теми також полягає в тому, що розвиток сучасних технологій сприяє зростанню конкуренції у ресторанній галузі. Ресторанам необхідно бути в курсі останніх тенденцій та використовувати інноваційні рішення для забезпечення своєї конкурентоспроможності. Система ресторанного управління, розроблена на основі Java, Spring Framework, PostgreSQL та React.js, є одним із таких інноваційних рішень, яке допоможе ресторанам ефективно виконувати свої функції та відповідати сучасним вимогам ринку.

Таким чином, розробка системи ресторанного управління на основі зазначених технологій є актуальною та відповідає потребам ресторанної галузі [2].

1.1.1 Огляд конкурентів

Одним із лідерів у вирішенні питань розробки програмного забезпечення для ресторанного бізнесу є американська компанія ‘Toast Pos’(рис. 1)

Рисунок 1 – Логотип компанії конкурента Toast Pos

Toast POS є інтегрованою платформою для управління рестораном, яка надає широкий спектр функцій, що спрощують операції і покращують ефективність. Основні риси та можливості Toast POS включають:

1. Замовлення та операції: Toast POS дозволяє офіціантам приймати замовлення на планшетах чи касових системах, спрощуючи процес обслуговування клієнтів. Вона також надає можливість керувати модифікаторами страв, замовляти додаткові складники, налаштовувати різні параметри замовлення.

2. Управління меню: За допомогою Toast POS можна з легкістю налаштовувати та керувати меню ресторану. Вона надає інструменти для додавання нових страв, оновлення цін, налаштування акцій та знижок.

3. Оплата та розрахунок: Toast POS інтегрується з різними платіжними системами, що дозволяє приймати оплату за замовлення через різні методи, включаючи кредитні карти, мобільні платежі та подарункові картки. Також є можливість розділення рахунку між декількома клієнтами.

4. Аналітика та звіти: Toast POS забезпечує детальну аналітику продажів, замовлень та інших ключових метрик. Дані можна переглядати у реальному часі, створювати звіти та графіки для аналізу продуктивності ресторану.

5. Інтеграція з іншими системами: Toast POS пропонує інтеграцію з різними системами, включаючи системи управління складом, бронювання столиків, облік праці та інші [3].

1.2 Пошук акторів та варіантів використання

1.2.1 Виявлення акторів

На рисунку 2 представлені основні актори системи.

Рисунок 2 – Основні актори системи

Короткий опис акторів представлений в таблиці. 1.

Табл. 1. Виявлення акторів

Актор	Короткий опис
Адміністратор	Має можливість створювати, редагувати або видаляти страви з меню. Також має можливість до перегляду статистика, імпорту чи експорту меню. Може додавати або видаляти столи. Крім того, має можливість додавати або видаляти працівників.
Працівник	Має можливість додавати, видаляти страви з меню, редагувати кількість страв в чеку, обраховувати чек, змінювати статус стола.

1.2.2 Виявлення варіантів використання

Виявлення варіантів використання представлені таблиці 2.

Табл. 2. Варіанти використання

Адміністратор	Управління столами	Можливість адміністратору додавати чи видаляти столи та переглядати їх статус.
Адміністратор	Імпорт меню	Можливість адміністратору робити імпорт меню з файлу
Адміністратор	Експорт меню	Можливість адміністратору експортувати меню в файл.
Адміністратор	Отримання статистики	Можливість адміністратору отримувати статистику.
Адміністратор	Управління працівниками	Можливість адміністратору створювати чи видаляти працівників.
Адміністратор	Отримання інформації про меню	Адміністратор отримує всю інформацію про меню.
Працівник	Управління замовленнями	Можливість працівнику додавати чи видаляти страви в чеку, змінювати кількість страв в чеку, обраховувати чек, змінювати статус стола.
Працівник	Отримання інформації про замовлення	Працівник отримує всю інформацію про замовлення.

1.2.3 Розробка варіантів використання

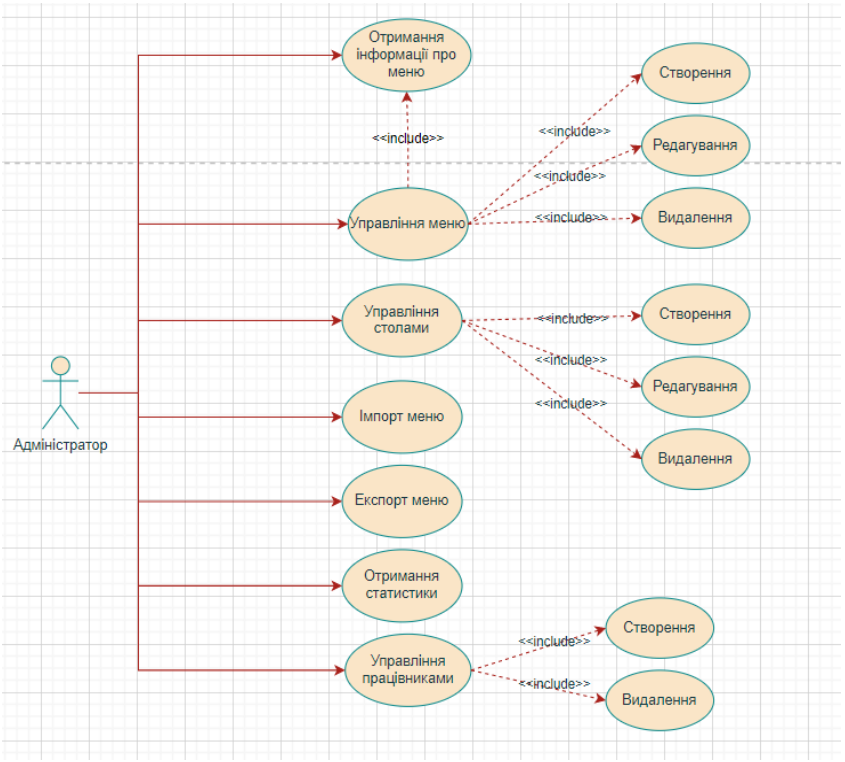


Рисунок 3 – Діаграма прецедентів системи

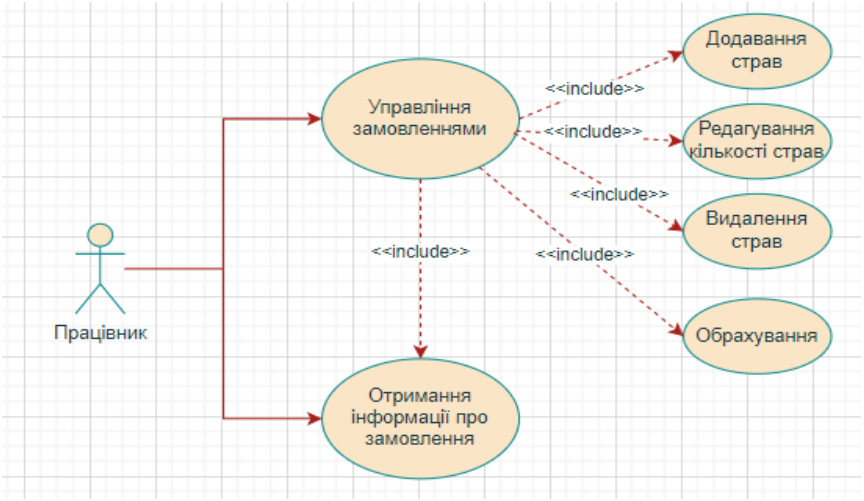


Рисунок 4 – Діаграма прецедентів системи

1.3 Діаграми послідовності

Діаграма послідовності (sequence diagram) - це тип діаграми взаємодії в мові моделювання UML (Unified Modeling Language), яка ілюструє послідовність

взаємодій між об'єктами або компонентами в системі. Вона відображає потік повідомлень або викликів методів між об'єктами протягом певного періоду часу.

У діаграмі послідовності об'єкти зображаються у вигляді вертикальних ліній життєвого циклу, а повідомлення, що обмінюються між ними, зображаються у вигляді стрілок. Послідовність повідомлень та їх порядок відображають динамічну поведінку та співпрацю між об'єктами під час певного сценарію або використання.

Діаграми послідовності часто використовуються для візуалізації та розуміння взаємодій та потоків повідомлень у складних системах або програмних додатках. Вони допомагають у проектуванні, документуванні та комунікації поведінки системи, фіксуючи динамічні аспекти співпраці об'єктів для досягнення певної функціональності.

Аналізуючи діаграму послідовності, розробники та дизайнери можуть отримати уявлення про взаємодії між об'єктами, порядок обміну повідомленнями та хронометраж подій. Це допомагає виявити можливі складнощі та зрозуміти загальну поведінку системи [4].

Процес здійснення отримання інформації меню. Спочатку користувач заходить досистеми для отримання можливості переглядати меню. Далі система обробляє запит і повертає список страв.

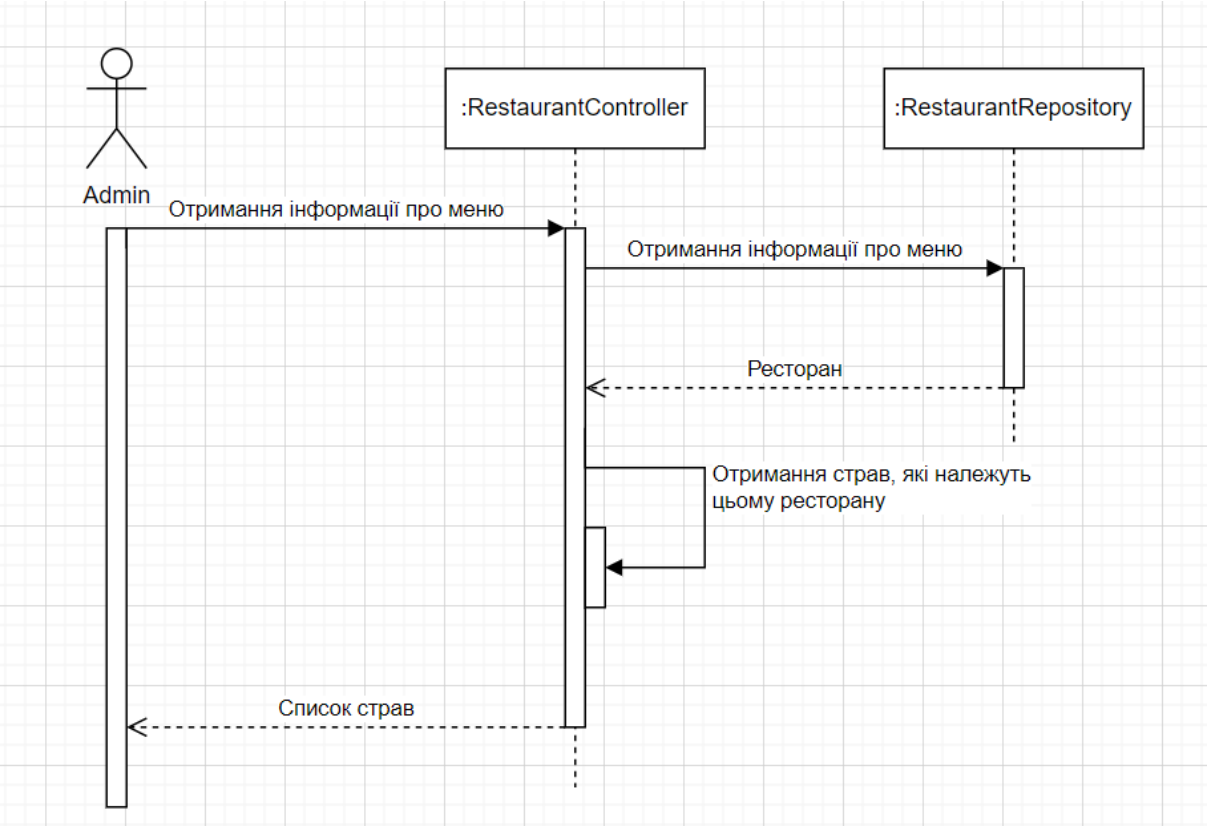


Рисунок 5 – Діаграма послідовності отримання інформації про меню

Процес здійснення імпорту. Спочатку користувач заходить до системи для отримання можливості імпортувати дані з файлу для заповнення меню. Далі користувач вибирає файл для імпорту і надсилає його на обробку. Далі система обробляє запит і повертає повідомлення про успішний імпорт.

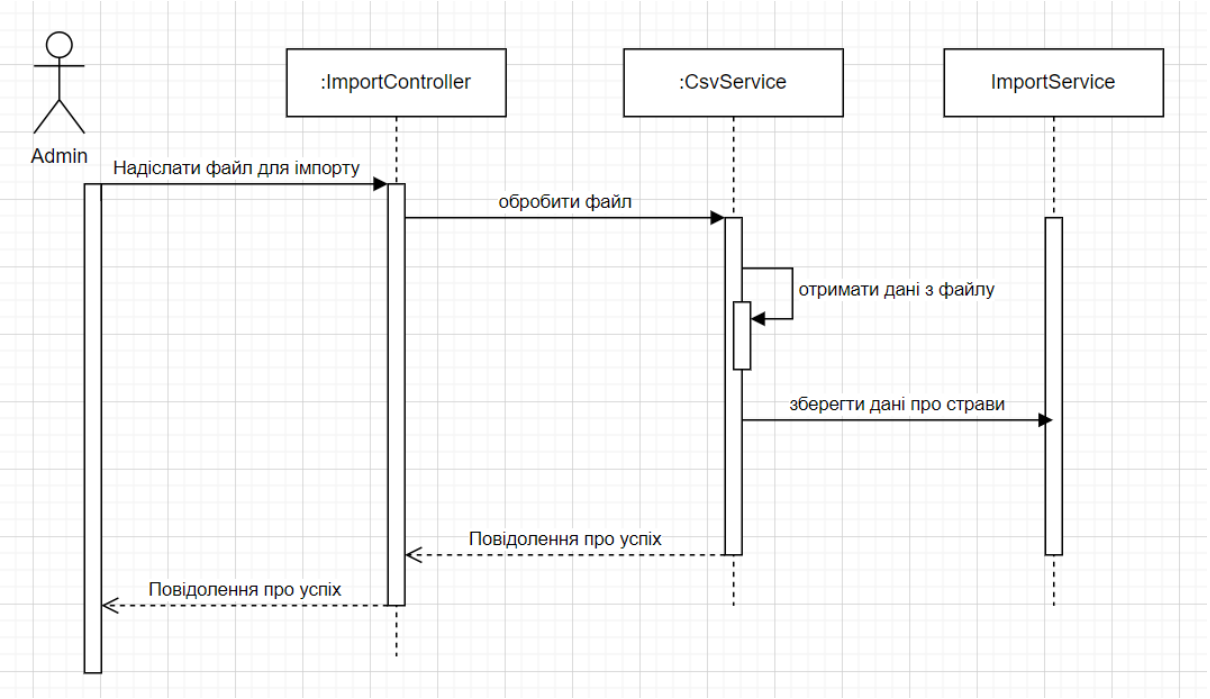


Рисунок 6 – Діаграма послідовності імпорту меню

Процес здійснення експорту. Спочатку користувач заходить до системи для отримання можливості експортувати дані про меню у файл. Далі користувач натискає на кнопку експорту. Далі система обробляє запит і повертає файл, у якому зберігаються дані про меню.

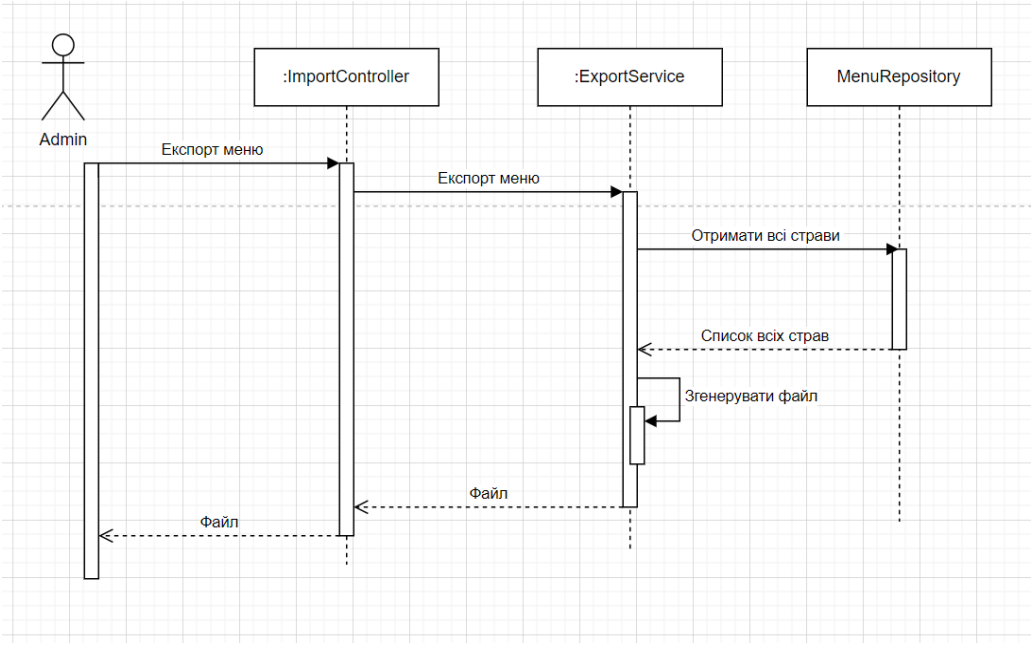


Рисунок 7 – Діаграма послідовності експорту меню

Процес здійснення обчислення чеку. Спочатку працівник заходить до системи для отримання можливості обчислювати чек. Далі працівник натискає на кнопку обрахувати. Далі система обробляє запит і повертає обрахований чек, змінює статус і очищає чек від попередніх страв та зберігає страви, які належали до чека, до статистики.

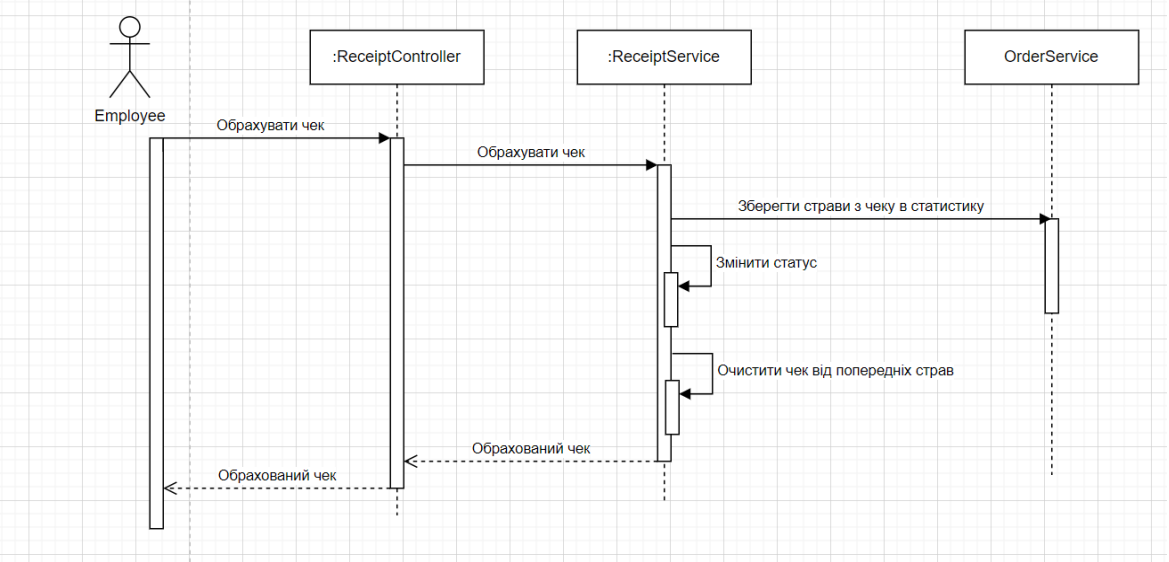


Рисунок 8 – Діаграма послідовності обчислення чеку

1.4 Вибір технологій розробки

Архітектура клієнт-сервер є моделлю обчислення, де клієнтські пристрої або додатки взаємодіють з централізованим сервером для запиту та отримання послуг, ресурсів або даних. У цій архітектурі клієнт і сервер є відокремленими сутностями, які виконують різні ролі та мають окремі обов'язки.

Клієнт, також відомий як фронтенд, зазвичай є додатком або пристроєм, який взаємодіє з користувачем і ініціює запити до сервера. Клієнт відповідає за представлення користувацького інтерфейсу, збір введених даних та відправлення запитів на обробку або отримання даних до сервера.

Сервер, також відомий як бекенд, є потужним комп'ютером або мережею комп'ютерів, які надають послуги або ресурси клієнтам. Він отримує запити від клієнтів, обробляє їх та повертає необхідну інформацію або виконує запитані дії. Сервер відповідає за управління даними, виконання бізнес-логіки та координацію загальної функціональності системи.

Комунікація між клієнтом і сервером в архітектурі клієнт-сервер зазвичай відбувається через мережу, таку як інтернет. Клієнти надсилають запити серверу, який їх обробляє та надсилає відповідні відповіді. Ця комунікація відбувається за певними протоколами та стандартами, наприклад, HTTP (протокол передачі гіпертексту) для веб-додатків.

Архітектура клієнт-сервер дозволяє масштабування, оскільки декілька клієнтів можуть підключатися до одного сервера або до групи серверів для доступу до спільних ресурсів або послуг. Вона також забезпечує централізоване управління, безпеку та стійкість даних, оскільки дані та функціональність зберігаються та контролюються сервером. Ця архітектура широко використовується в різних додатках, включаючи веб-додатки, системи баз даних, поштові сервери та багато інших мережевих систем (рис. 9) [5].

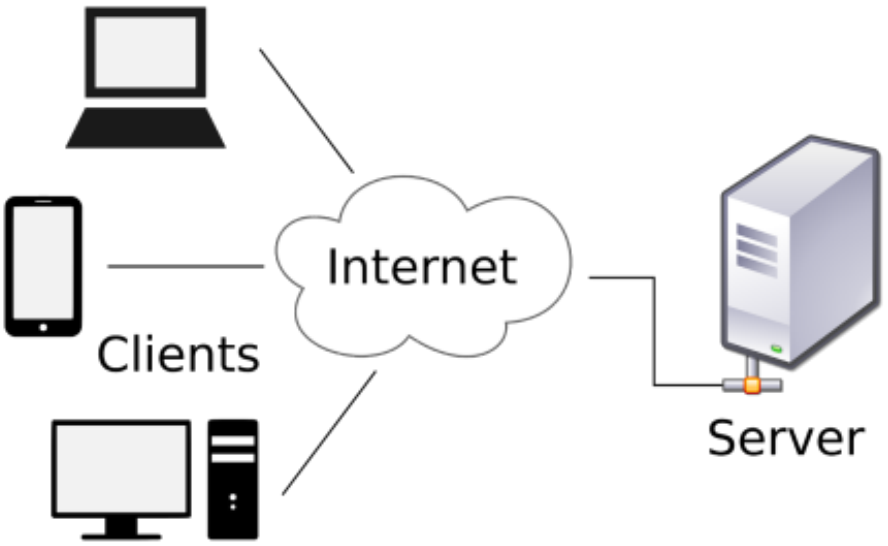


Рисунок 9 – Клієнт-серверна архітектура

У розробці серверної частини проекту буде використана мова програмування Java, яка буде відповідати за зберігання бази даних та обробку запитів, що надходять від клієнтів. Серверна частина буде забезпечувати взаємодію з базою даних та надавати необхідні відповіді клієнтам через API. Використання мови Java забезпечує надійність, ефективність та розширюваність серверної частини проекту. Мова програмування Java є загального призначення, паралельною, класовою, об'єктно-орієнтованою мовою. Мова програмування Java пов'язана з мовами C і C++, але організована трохи інакше, з виключенням деяких аспектів C і C++ та включенням деяких ідей з інших мов. Вона призначена для використання у реальних проектах, а не як дослідна мова, тому, як пропонував C. A. R. Хоар в своїй класичній статті про проектування мов, у дизайні уникнуто включення нових та неперевірених функцій [6].

Для розробки API веб-додатку було прийнято рішення використовувати Spring Framework. Spring Framework - це високопродуктивний інструментарій для розробки програмного забезпечення на платформі Java. Він надає комплексне середовище та набір готових компонентів і бібліотек, які спрощують розробку програмних додатків.

Spring Framework базується на принципах інверсії керування (IoC) та аспектно-орієнтованого програмування (AOP), що дозволяє зробити програмний код більш модульним, повторно використовуваним і легко тестируваним. Він також надає підтримку для розробки веб-додатків, роботи з базами даних, кешування, безпеки та інших аспектів розробки програмного забезпечення.

Spring Framework має велику спільноту розробників, яка активно підтримує його, надає документацію, приклади коду і відповідає на запитання. Він є одним з найпопулярніших фреймворків для розробки Java-додатків і використовується в багатьох великих проектах та підприємствах. (рис. 10) [7].

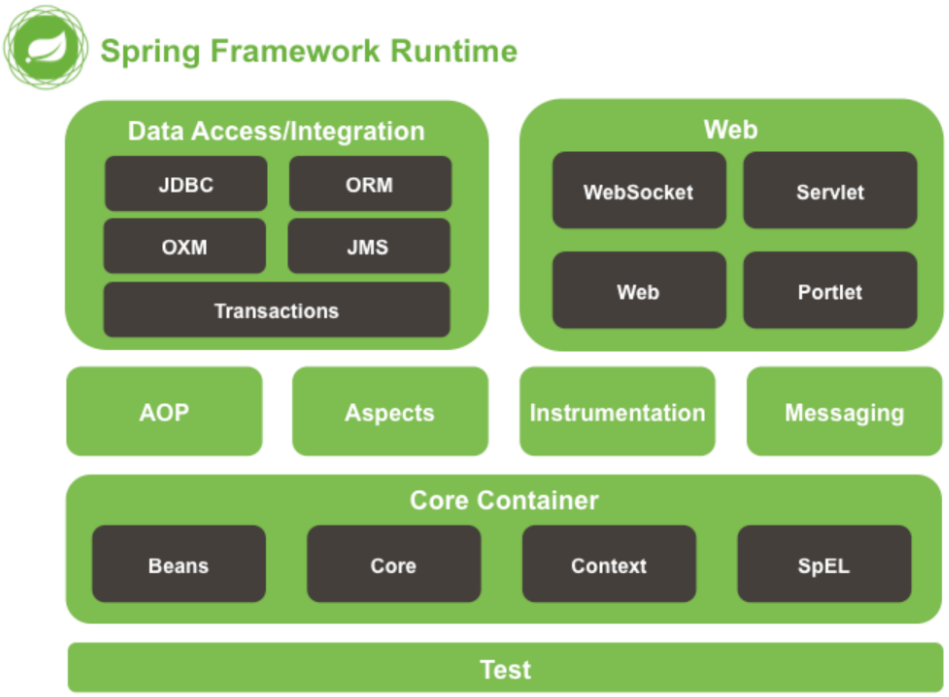


Рисунок 10 – Архітектура Spring Framework

Для серверної частини була обрана архітектура з декількома шарами, відома також як багатошарова архітектура. Цей архітектурний підхід є стандартом у багатьох серверних додатках і має широке застосування. Багатошарова архітектура відповідає комунікації та організаційній структурі в розробці програмного забезпечення, що робить його природним вибором для багатьох бізнес-орієнтованих додатків. Цей підхід також має перевагу простоти в розробці та тестуванні.

У багатошаровій архітектурі компоненти додатка організовані у вертикальні шари, де кожен шар виконує свої власні функції та відповідає за певний аспект додатку, наприклад, представлення, бізнес-логіку, збереження даних тощо. Важливо зазначити, що конкретна кількість шарів може варіюватись залежно від конкретного додатку, проте зазвичай використовуються стандартні рівні, такі як представлення, бізнес-логіка, доступ до даних та база даних. У менших додатках може бути використано менше шарів, а у складних проектах можуть бути використані додаткові шари залежно від потреб додатка. [8].

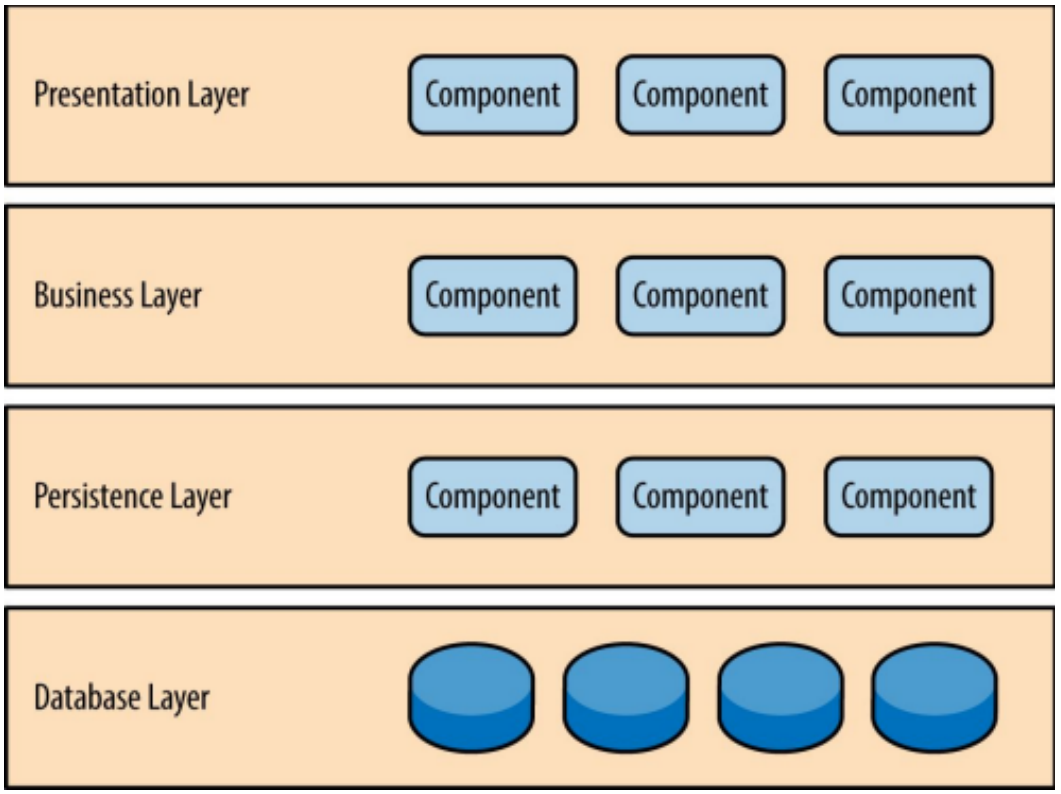


Рисунок 11 – Багатошарова архітектура

Була вибрана PostgreSQL як система управління базами даних. PostgreSQL є потужною об'єктно-реляційною системою з відкритим вихідним кодом, яка активно розвивається протягом багатьох років і відрізняється високою надійністю, розширеними функціями і високою продуктивністю. Існує велика кількість інформації, що описує процес встановлення та використання PostgreSQL у офіційній документації. Спільнота з відкритим вихідним кодом надає багато корисних ресурсів для ознайомлення з PostgreSQL, вивчення його роботи та пошуку кар'єрних можливостей [9].

Клієнтська частина буде реалізована за допомогою бібліотеки React (рис. 12). Бібліотека React.js – це відкритий фреймворк та бібліотека JavaScript, розроблена компанією Facebook. Він використовується для швидкої та ефективної розробки інтерактивних інтерфейсів користувача та веб-додатків з використанням значно меншої кількості коду, ніж при використанні чистого JavaScript [10].

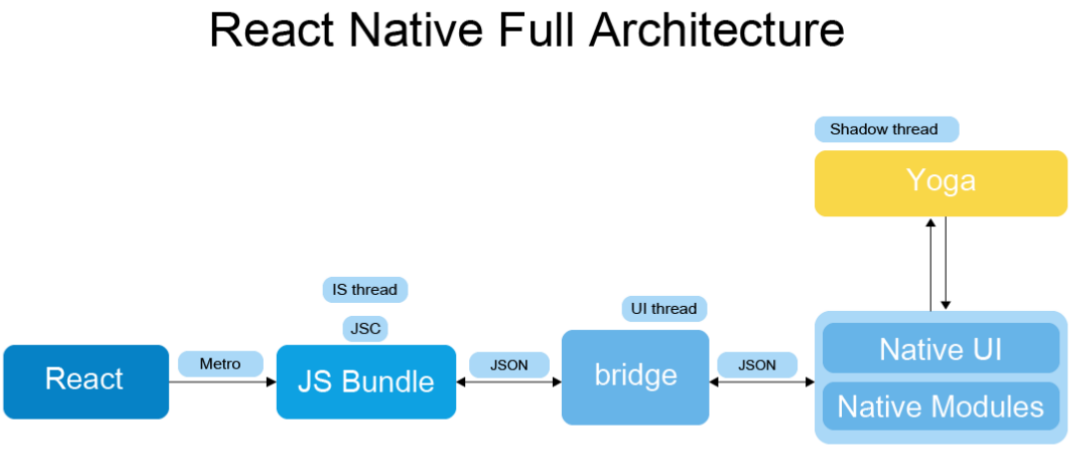


Рисунок 12 – Архітектура React

Для створення розмітки і стилів системи буде використано HTML і CSS.

HTML (HyperText Markup Language) є мовою розмітки для створення веб-сторінок. Вона використовується для структуризації та відображення вмісту на веб-сторінках, таких як текст, зображення, посилання, таблиці, форми тощо. HTML складається з різних елементів, які визначаються за допомогою тегів.

Кожен елемент в HTML має свою функцію та призначення. Наприклад, тег <p> використовується для відображення абзацу тексту, <img> - для вставки зображень, <a> - для створення посилань і так далі. За допомогою різних атрибутів, які можуть бути додані до тегів, можна налаштовувати вигляд та поведінку елементів.

HTML є базовим будівельним блоком веб-сторінок, але вона сама по собі не забезпечує динамічність та функціональність. Для цього можуть використовуватися мови програмування, такі як JavaScript, які додають інтерактивність до веб-сторінок.

HTML є стандартом, який підтримується всіма сучасними веб-браузерами, і вона є основою для розробки веб-сторінок та веб-додатків. Існують багато ресурсів та документації, де можна докладніше ознайомитись зі синтаксисом та можливостями HTML [11].

CSS (Cascading Style Sheets) - це мова стилів, яка використовується для опису зовнішнього вигляду веб-сторінок, написаних у мові розмітки HTML. CSS визначає, які кольори, шрифти, розташування елементів та інші атрибути повинні бути застосовані до елементів веб-сторінки.

Завдяки CSS веб-розробники можуть легко контролювати вигляд та макет своїх веб-сторінок. Вони можуть встановлювати стилі для конкретних елементів або груп елементів, застосовувати розмітку, задавати кольори, фони, розміри, відступи, анімації та багато іншого. CSS забезпечує розділення між вмістом (HTML) та його представленням (стилі), що робить код більш читабельним та підтримуваним [12].

Для ефективного використання React зазвичай використовують state containers, для управління станом додатку. В системі ресторан буде використано бібліотеку Redux, для управління станом (рис. 13). Redux – це передбачуваний контейнер стану, призначений для допомоги у написанні JavaScript-додатків, які поводяться послідовно на клієнтському, серверному та мобільному середовищах, а також легко піддаються тестуванню [13].

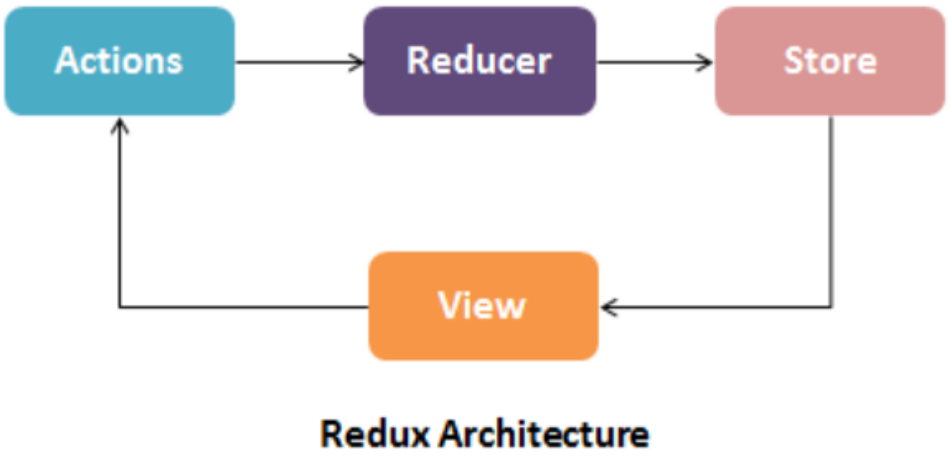


Рисунок 13 – Архітектура Redux

В якості хосту для системи використовуватиметься хмарна платформа Heroku (рис. 14). Heroku - це хмарна платформа розгортання та управління

веб-додатками, яка надає розробникам зручні інструменти для розгортання, масштабування та керування їх додатками. Основна ідея Heroku полягає в тому, щоб розробникам зосередитися на написанні коду додатку, не витрачаючи час і зусилля на налаштування серверного середовища.

Деякі ключові особливості Heroku включають:

1. Простий процес розгортання: Heroku надає простий інтерфейс для завантаження додатків та автоматичного розгортання. Ви можете розгорнути свій код лише за кілька кроків.

2. Масштабованість: Heroku дозволяє легко масштабувати ваш додаток, надаючи можливість змінювати розмір ресурсів в залежності від потреб вашого додатку.

3. Підтримка багатьох мов програмування: Heroku підтримує різні мови програмування, такі як Java, Ruby, Python, Node.js, PHP і багато інших. Ви можете використовувати мову, з якою ви найбільш знайомі і комфортні.

4. Управління базами даних: Heroku має вбудовану підтримку для різних баз даних, включаючи PostgreSQL, MySQL, MongoDB і багато інших. Ви можете легко підключити та керувати своїми базами даних без зайвого зусилля.

5. Розширені можливості: Heroku надає додаткові можливості, такі як засоби моніторингу, логування, керування налаштуваннями і доступом до сторонніх сервісів, щоб спростити розробку та керування вашим додатком [14].

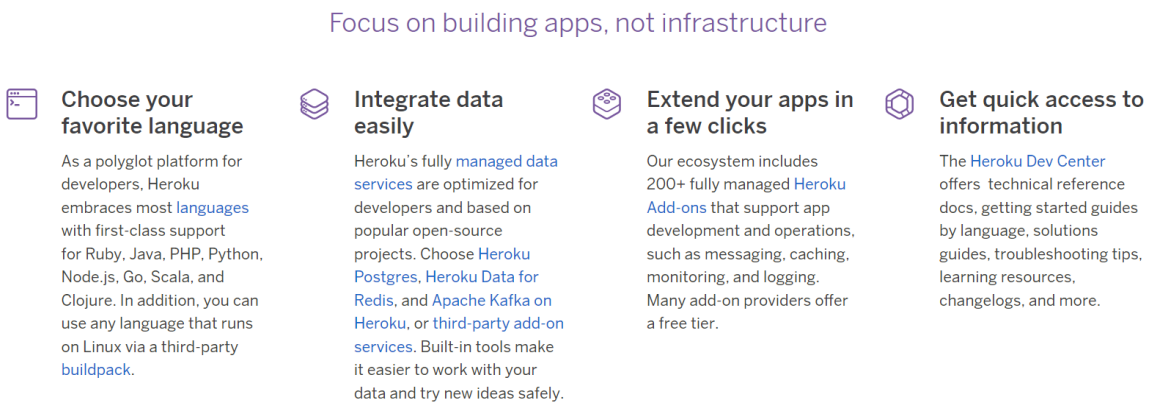


Рисунок 14 – Переваги Heroku

В якості середовища розробки було обрано Intellij Idea (рис. 15). IntelliJ IDEA - це інтегроване середовище розробки (IDE) для мов JVM, призначене для максимізації продуктивності розробника. Воно виконує рутинні та повторювані завдання, надаючи розумне автодоповнення коду, статичний аналіз коду та операції рефакторингу, і дозволяє зосередитися на позитивних аспектах розробки програмного забезпечення, забезпечуючи не лише продуктивність, але й задоволення від процесу [15].

Рисунок 15 – Особливості Intellij Idea

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Побудова схеми бази даних

У цьому додатку необхідно мати користувачів, що створює першу сутність - users. Кожен користувач має мати логін та пароль, які представлені у вигляді рядків, а також роль, також представлену у вигляді рядка. У кожного користувача є свій власний ресторан, тому ми вводимо сутність "restaurant", яка є додатковою і пов'язана з сутністю "users". Як атрибут ресторани будуть мати ім’я стрічкового типу даних із фіксованою довжиною. Кожен ресторан повинен мати меню і столи, тому вводимо сутності menu і table. Як атрибути стіл буде мати ім’я стрічкового типу даних та статус стрічкового типу даних. До кожного стола належить замовлення, тому вводимо сутність receipt. Меню повинно складатися зі страв, тому вводимо ще одну сутність meal. Страви, як атрибути мають, назву стрічкового типу даних, ціну грошового типу даних, розмір порції стрічкового типу даних. Кожна страва може належати до певної групи, тому вводимо ще одну сутність mealGroup. Група як атрибут має ім’я стрічкового типу даних. Для зберігання статистики замовлень, вводимо сутність order. Як атрибут order має час створення типу даних дати.

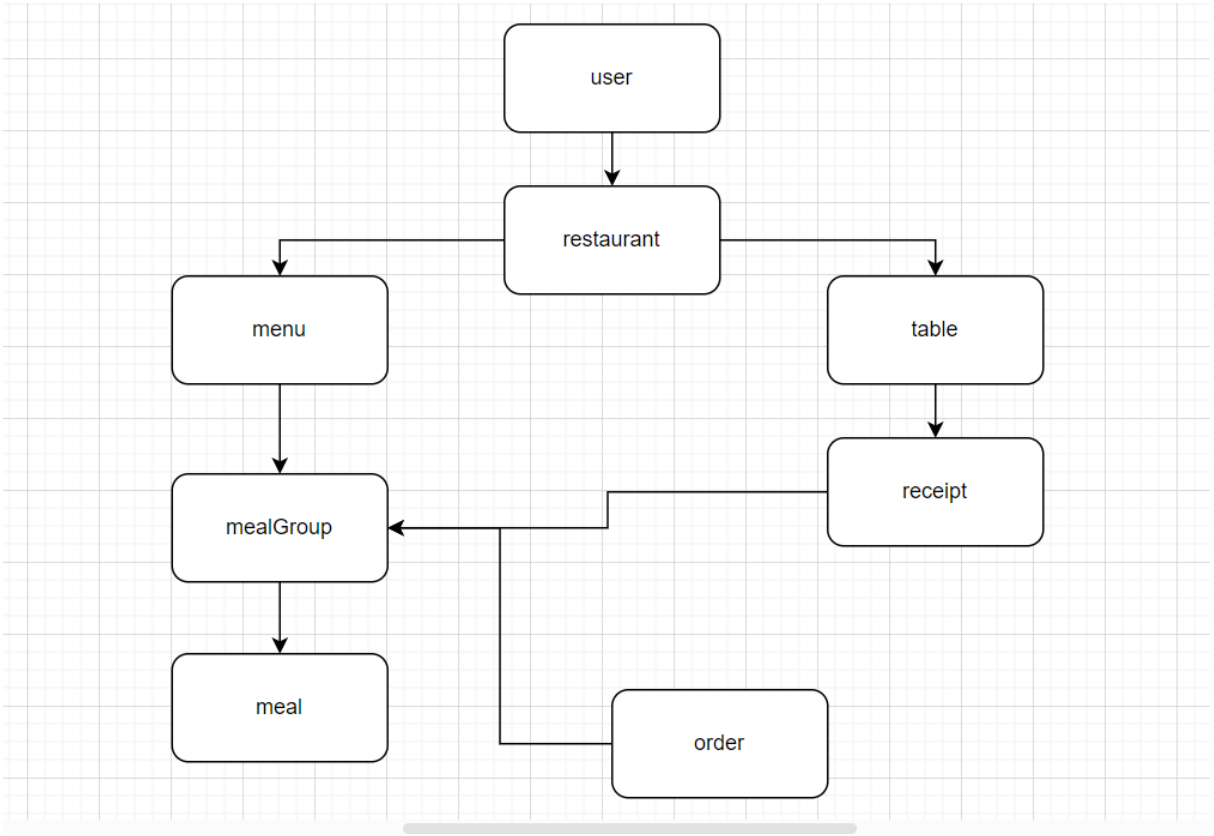


Рисунок 16 – Схема бази даних на рівні концепцій

Користувач може мати тільки один ресторан, але один ресторан може належати кільком користувачам. Тому між ними зв’язок один до багатьох (рис. 17).

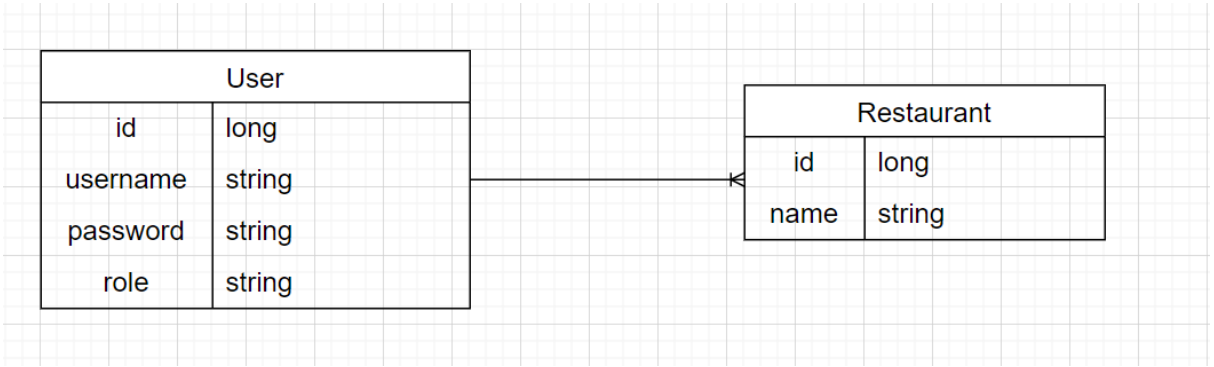


Рисунок 17 – Зв’язок users-restaurant

Ресторан повинен мати меню, для подальших сторень замовлень. Оскільки меню складається з груп страв, а страви належать до груп. Було вирішено, що один ресторан може мати кілька груп тому між ними зв’язок один до багатьох. А

також до однієї групи може належати багато страв, тому тут також один до багатьох (рис. 18).

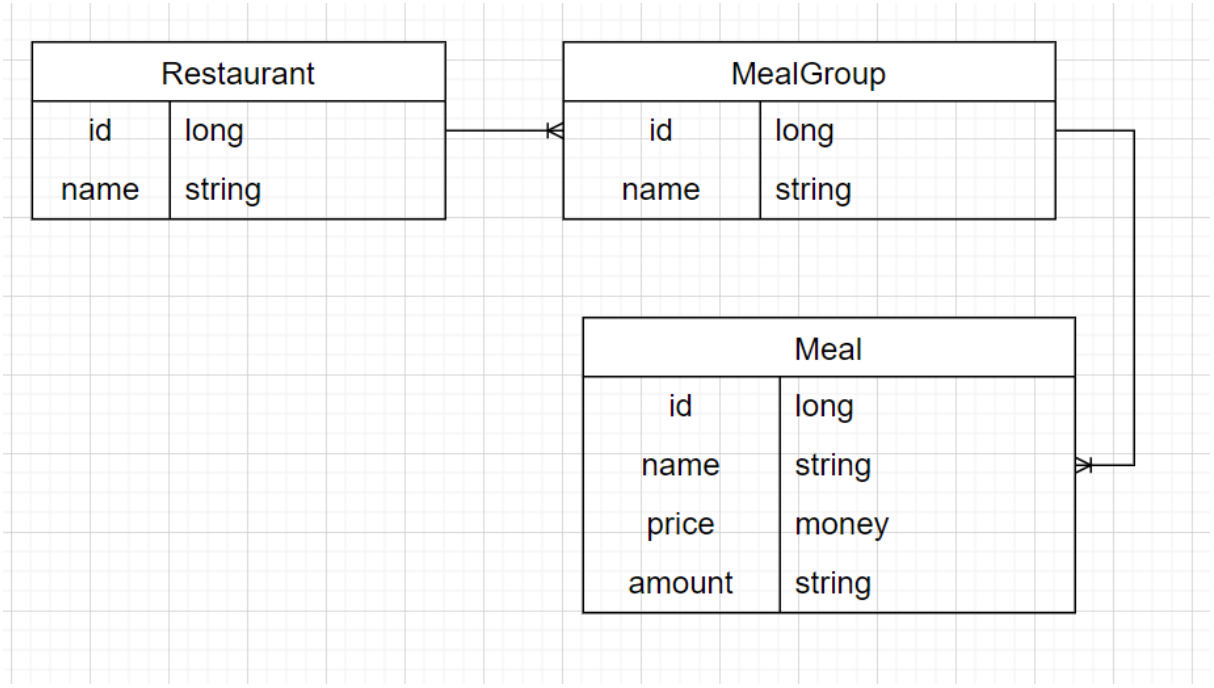


Рисунок 18 – Зв’язок restaurant-mealGroup-meal

Ресторан повинен мати стіл, для подальших сторень замовлень. Оскільки в стола є замовлення, а в замовлені, замовлені страви, було вирішено, що один ресторан може мати кілька столів тому між ними зв’язок один до багатьох. А стіл може мати тільки одне замовлення, тому зв’язок один до одного. Також замовлення може мати багато страв, але так як, страви належать до груп, то зв’язок між замовленням і групами один до багатьох (рис. 19).

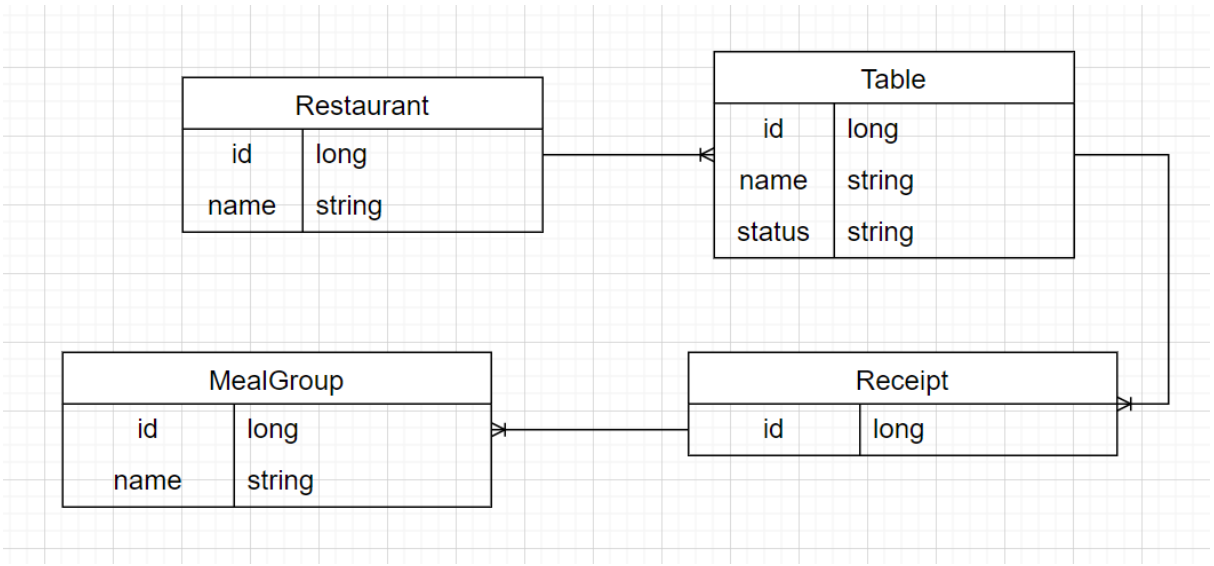


Рисунок 19 – Зв’язок restaurant-table-receipt-mealGroup

Статистика повинна мати список страв, які були замовленні, а отже зв’язок між статистикою і групами страв один до багатьох (рис. 20).

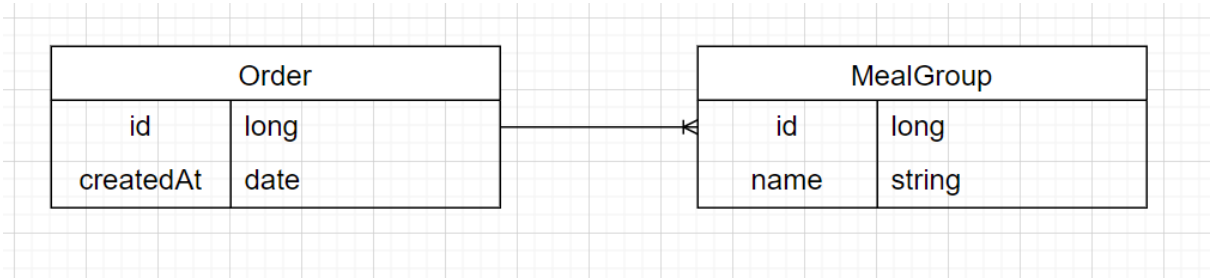


Рисунок 20 – Зв’язок order-mealGroup

Логічна модель бази даних в повному обсязі представлена на рисунку 21.

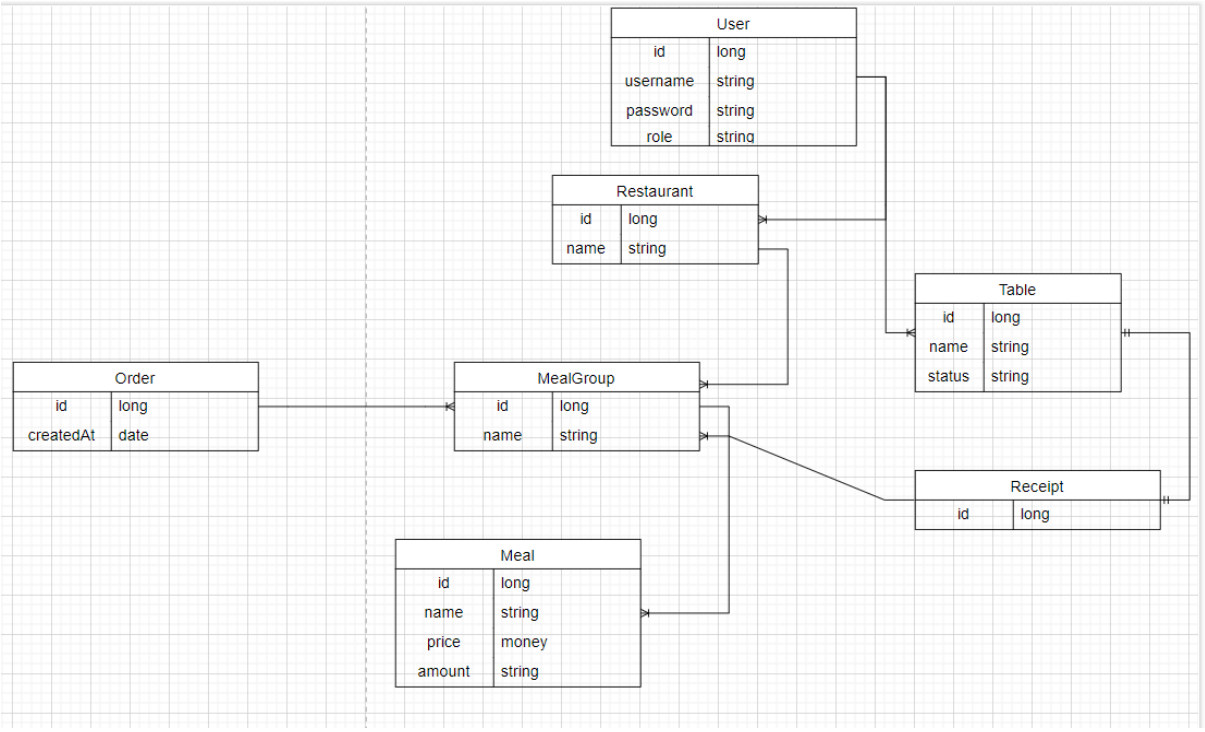


Рисунок 21 – Модель бази даних на рівні логіки

Враховуючи той факт, що в системі користувачі можуть бути або користувачами або адмінами, а по правилам системи користувачів може створювати тільки адмін, звідси випливає що між користувачами з правами адміна і звичайними користувачами існує зв’язок, звичайний користувач має свого адміна, адмін може мати кілька користувачів, отже зв’язок один до багатьох. Тому для збереження такої інформації нам потрібно окрема таблиця.

Оскільки нам не може бути дозволено зберігати кілька записів для одного користувача через обмеження первинного ключа, ми повинні створити таблицю з назвою "user_employees". Ця таблиця буде мати зв'язок один до багатьох з таблицею "users". Для цього вона буде містити поля "id користувача" та "id адміна". Ці атрибути будуть мати обмеження зовнішнього ключа, а для унікальності та уникнення повторення входження одного користувача до однієї групи, вони разом створюватимуть складений ключ (рис. 22).

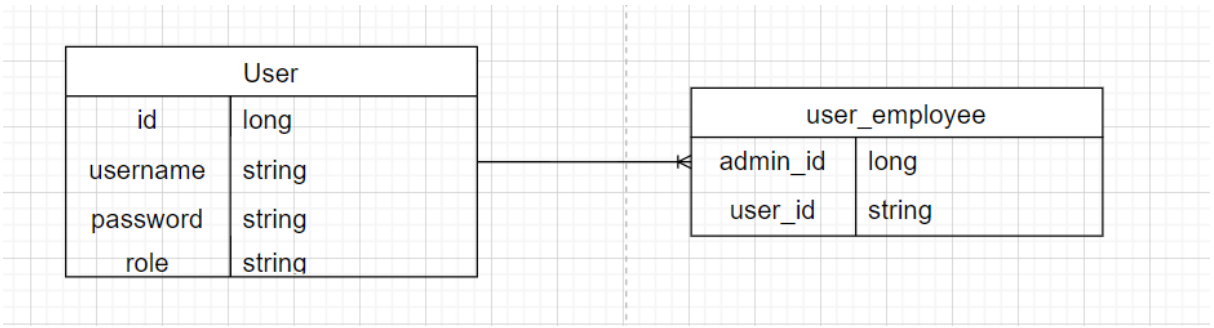


Рисунок 22 – Зв’язок user до user_employees

Оскільки потрібно добавити кількість страв в одному замовленні, звідси випливає, що між замовленнями і стравами є зв’язок багато до багатьох. Щоб реалізувати це в базі даних, потрібно створити таблицю receipt_meal. Таблиця, про яку йдеться, встановить зв'язок один до багатьох з іншою таблицею receipt і один до багатьох з таблицей meal. Для цього вона повинна зберігати значення id замовлення та id страви, а також значення кількості страв в замовленні. Атрибути даних будуть мати обмеження зовнішнього ключа, а для запобігання повторного входження одного й того ж користувача до однієї групи вони будуть утворювати композитний ключ (рис. 23).

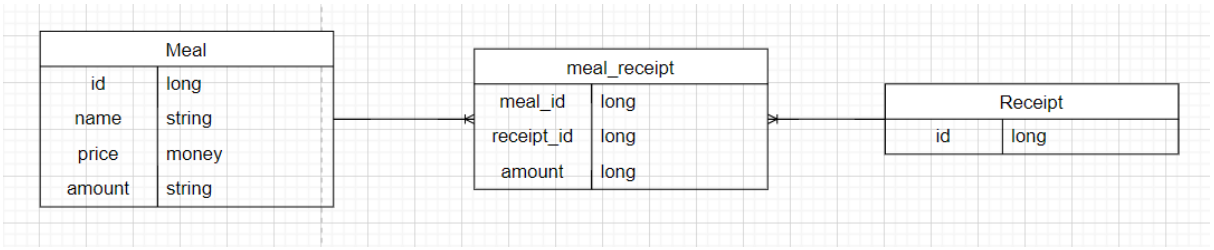


Рисунок 23 – Зв’язок meal-meal_receipt-receipt

2.2. Архітектура системи

2.2.1 Визначення класів системи

Маючи всі відомості про систему з першого розділу, можна приступити до

проектування архітектури системи, а сама визначення, якими класами буде володіти система. Визначивши всі класи системи побудуємо діаграму класів за допомогою мови UML.

Діаграма класів (Class Diagram) є графічним інструментом для статичного відображення структури моделі системи, яка базується на об'єктно-орієнтованому програмуванні. Вона надає візуальне представлення класів, типів даних, взаємозв'язків та взаємовідносин між ними. Діаграма класів дозволяє відображати не лише самі класи і їх атрибути, але й інтерфейси, об'єкти та їх поведінку [16].

У програмуванні, клас є основним будівельним блоком, який визначає шаблон для створення об'єктів. Він служить як шаблон або основа для створення екземплярів об'єктів, які є конкретними представниками на основі визначення класу.

Клас інкапсулює дані (атрибути або властивості) та поведінку (методи або функції), які пов'язані з певним поняттям або сутністю. Атрибути представляють стан або характеристики об'єкта, а методи визначають операції або дії, які об'єкт може виконувати.

Класи є основним концептом об'єктно-орієнтованого програмування (ООП), парадигми програмування, яка ставить акцент на організацію коду в повторно використовувані об'єкти. ООП дозволяє модульне, структуроване та ефективне розроблення програмного забезпечення, пропагуючи повторне використання коду, інкапсуляцію та абстракцію.

Створюючи кілька екземплярів класу, ви можете створювати об'єкти, які мають спільні характеристики та поведінку, визначені класом. Кожен об'єкт може мати власний стан (значення атрибутів) і може незалежно виконувати методи, визначені класом.

Класи надають можливість моделювання сутностей реального світу, систем або абстрактних концепцій в програмному застосуванні. Вони допомагають організувати код, сприяють повторному використанню коду та забезпечують чітку структуру для проектування та реалізації складних систем.

Загалом, клас діє як шаблон або основа, яка визначає структуру, атрибути та поведінку об'єктів в об'єктно-орієнтованому програмуванні [17].

Для серверної частини була обрана архітектура з декількома шарами, відома також як багатошарова архітектура. Цей архітектурний підхід є стандартом у багатьох серверних додатках і має широке застосування. Багатошарова архітектура відповідає комунікації та організаційній структурі в розробці програмного забезпечення, що робить його природним вибором для багатьох бізнес-орієнтованих додатків. Цей підхід також має перевагу простоти в розробці та тестуванні.

У багатошаровій архітектурі компоненти додатка організовані у вертикальні шари, де кожен шар виконує свої власні функції та відповідає за певний аспект додатку, наприклад, представлення, бізнес-логіку, збереження даних тощо. Важливо зазначити, що конкретна кількість шарів може варіюватись залежно від конкретного додатку, проте зазвичай використовуються стандартні рівні, такі як представлення, бізнес-логіка, доступ до даних та база даних. У менших додатках може бути використано менше шарів, а у складних проектах можуть бути використані додаткові шари залежно від потреб додатка [18].

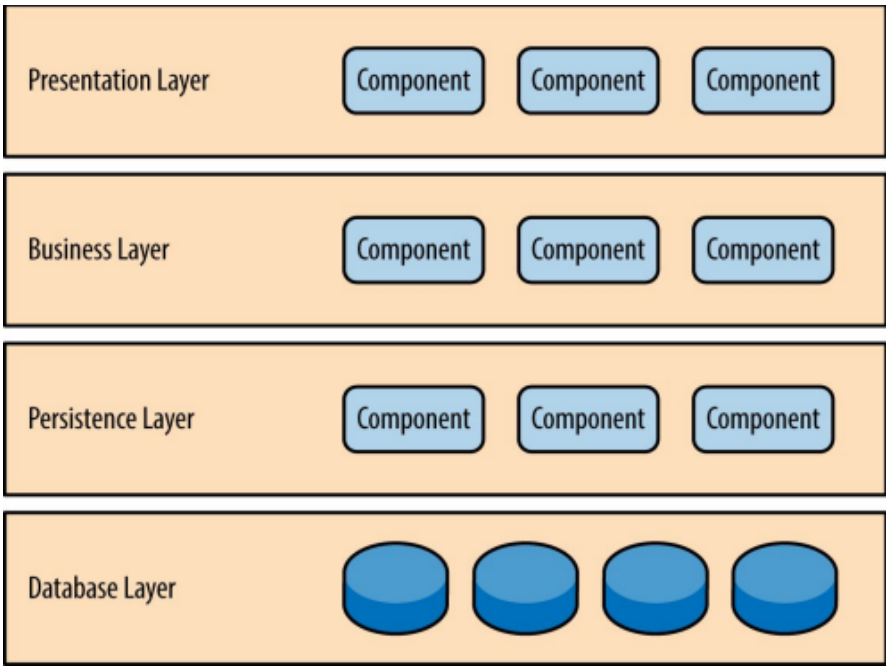


Рисунок 24 – Багатошарової архітектура

На рівні бази даних відбувається збереження і обробка даних, а також розташовується значна частина логіки, пов'язана з бізнес-процесами. Це дозволяє відділити логічну структуру даних від фізичної, що сприяє більшій гнучкості та підтримці масштабованості системи.

Структура класів представлена на рисунку 25.

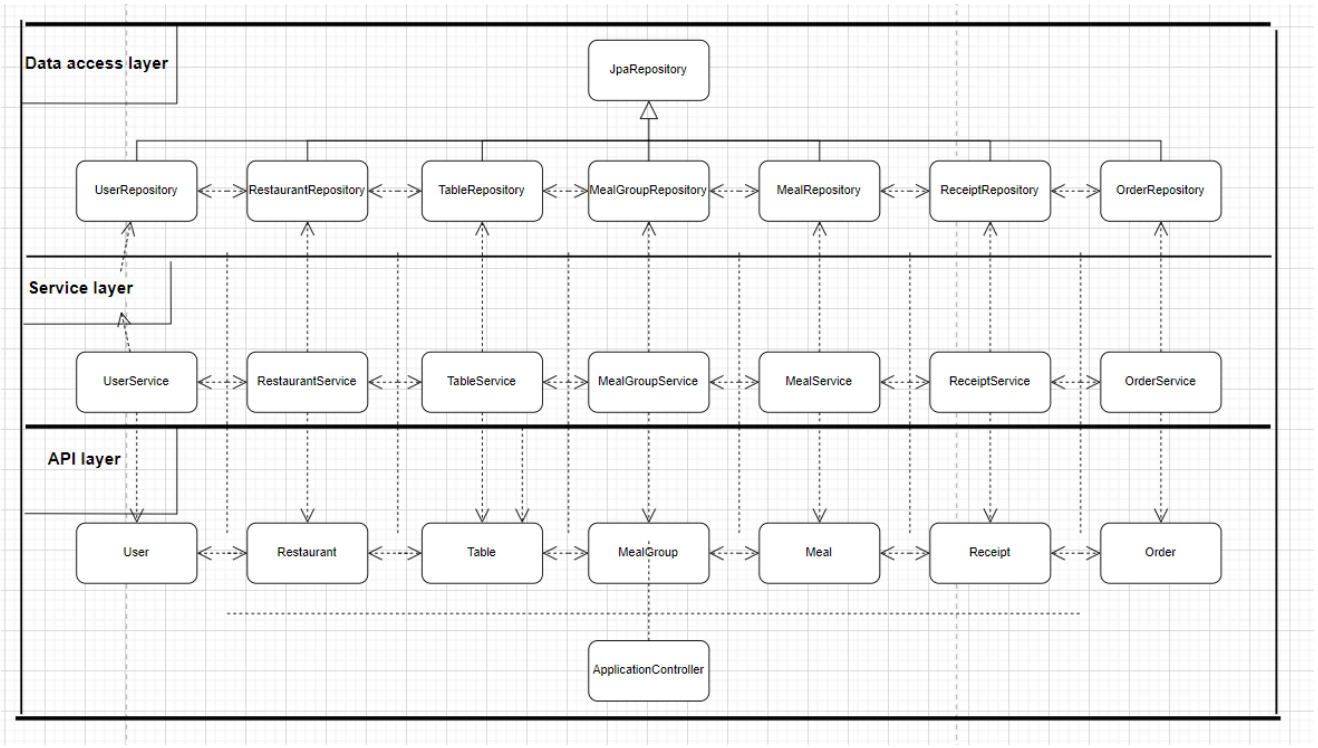


Рисунок 25 – Діаграма класів серверного додатку

Архітектура клієнтської частини представлена на рисунку 26.

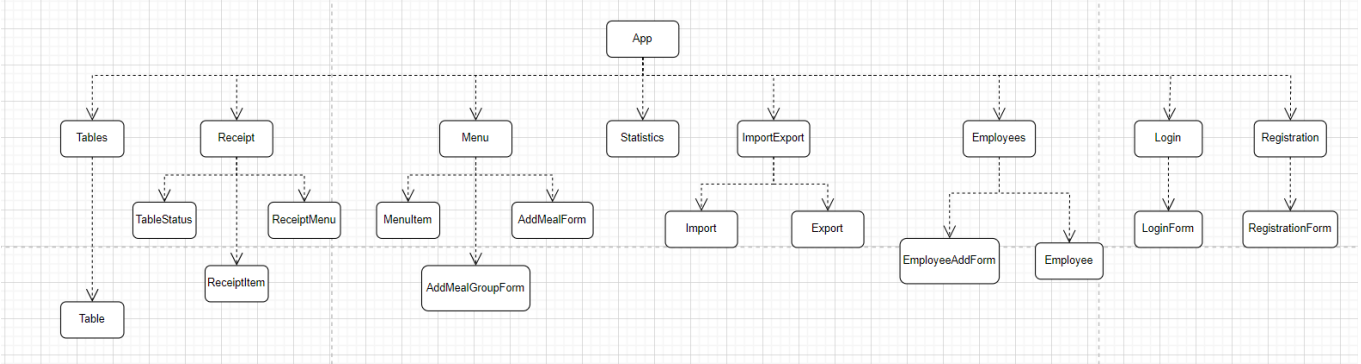


Рисунок 26 – Діаграма компонентів клієнтської частини

3 КОНСТРУЮВАННЯ

3.1 Реалізація класів

Лістинг коду для отримання інформації про меню представлений на рисунку 27.

```
public List<MealGroupResponse> getMenu(@PathVariable(value = "restaurantId") Long id) {
    Restaurant restaurantById = restaurantService.getRestaurantById(id);
    return restaurantById.getMealGroups() List<MealGroup>
            .stream() Stream<MealGroup>
            .map(mealGroup -> {
                MealGroupResponse response = modelMapper.map(mealGroup, MealGroupResponse.class);
                response.setMenu(mealGroup.getMenu().stream() Stream<Meal>
                        .map(meal -> modelMapper.map(meal, MealResponse.class)) Stream<MealResponse>
                        .toList());
                response.setRestaurantId(mealGroup.getRestaurant().getId());
                return response;
            }) Stream<MealGroupResponse>
            .toList();
}
```

Рисунок 27 – Метод для отримання інформації про меню

Лістинг коду для імпорту меню представлений на рисунку 28.

```
public String parseAndSave(MultipartFile file, Long restaurantId) {
    if (EXCEL_CSV_TYPE.equals(file.getContentType()) || CSV_TYPE.equals(file.getContentType())) {
        try {
            InputStream inputStream = file.getInputStream();
            parser.parse(inputStream);
            clearDatabaseFromPreviousMeals(restaurantId);
            rowProcessor.getBeans().forEach(dto -> importService.storeToDatabase(dto, restaurantId));
            return "Save of file " + file.getOriginalFilename() + " was successful";
        } catch (IOException e) {
            throw new UploadExceptions(UploadExceptions.Error.SAVE_WAS_NOT_SUCCESSFUL);
        }
    }
    throw new UploadExceptions(UploadExceptions.Error.INVALID_FILE_FORMAT);
}
```

Рисунок 28 – Метод для імпорту меню

Лістинг коду для експорту меню представлений на рисунку 29.

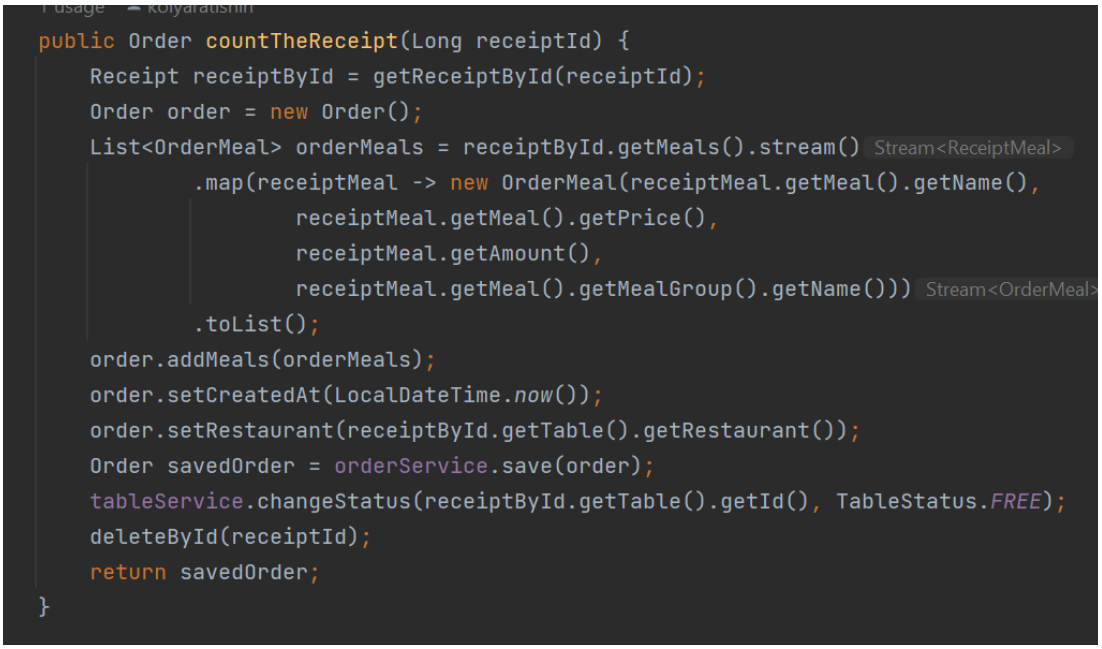
```
public ResponseEntity<ByteArrayResource> exportToCsv(@PathVariable(value = "restaurantId") Long restaurantId) {
    byte[] bytes = csvService.export(restaurantId);
    ByteArrayResource resource = new ByteArrayResource(bytes);

    HttpHeaders headers = new HttpHeaders();
    headers.add( headerName: "Content-Disposition", headerValue: "attachment; filename=export.csv");
    headers.add( headerName: "Cache-Control", headerValue: "no-cache, no-store, must-revalidate");
    headers.add( headerName: "Pragma", headerValue: "no-cache");

    return ResponseEntity.ok()
            .headers(headers)
            .contentType(MediaType.APPLICATION_OCTET_STREAM)
            .body(resource);
}
}
```

Рисунок 29 – Метод для експорту меню

Лістинг коду для обрахування чеку представлений на рисунку 30.



```
public Order countTheReceipt(Long receiptId) {
    Receipt receiptById = getReceiptById(receiptId);
    Order order = new Order();
    List<OrderMeal> orderMeals = receiptById.getMeals().stream() Stream<ReceiptMeal>
            .map(receiptMeal -> new OrderMeal(receiptMeal.getMeal().getName(),
                    receiptMeal.getMeal().getPrice(),
                    receiptMeal.getAmount(),
                    receiptMeal.getMeal().getMealGroup().getName())) Stream<OrderMeal>
            .toList();
    order.addMeals(orderMeals);
    order.setCreatedAt(LocalDateTime.now());
    order.setRestaurant(receiptById.getTable().getRestaurant());
    Order savedOrder = orderService.save(order);
    tableService.changeStatus(receiptById.getTable().getId(), TableStatus.FREE);
    deleteById(receiptId);
    return savedOrder;
}
```

Рисунок 30 – Метод для обрахування чеку

3.2 Забезпечення якості

Тестування – це процес, який дозволяє оцінити якість продукту, що

розроблюється. Якісний продукт повинен відповідати поставленим до нього вимогам. Програмна система повинна реалізовувати всі необхідні варіанти використання і не мати дефектів. Крім того, програмна система повинна мати властивості надійності (не має бути зависань, аварійних відмов і т. п.), безпеки, забезпечувати потрібну продуктивність, бути зручною в експлуатації, розширюваною. Таким чином, тестування представляє собою процес аналізу програмної системи, мета якого це виявлення дефектів і оцінка його характеристик [19].

Завдяки прогресу технологій, інтерфейс користувача стає все більш пріоритетним елементом в проектах. Це пояснюється тим, що успіх сервісу або веб-сайту значно залежить від задоволення користувачів та їх взаємодії з інтерфейсом. Наявність зручного та привабливого інтерфейсу може сприяти популярності та успіху проекту, оскільки користувачі швидше залишатимуться на сайті та будуть активно взаємодіяти з сервісом. Відгуки та реакції користувачів на інтерфейс допомагають вдосконалювати його і пристосовувати до потреб та очікувань аудиторії. Таким чином, якісний інтерфейс користувача є важливим фактором у популярності та успіху проекту.

Користувацький API, також відомий як API користувацького інтерфейсу або UI API, відноситься до набору програмних інтерфейсів або методів, які дозволяють розробникам взаємодіяти з та змінювати користувацький інтерфейс програми або системи. Він надає спосіб створення, модифікації та керування візуальними елементами та поведінкою користувацького інтерфейсу.

За допомогою користувацького API розробники можуть отримувати доступ до різних компонентів користувацького інтерфейсу, таких як кнопки, форми, меню, діалогові вікна та інші інтерактивні елементи. Вони можуть програмно визначати вигляд, розташування та поведінку цих компонентів, а також обробляти введення користувача та реагувати на події, спричинені діями користувача.

Користувацький API абстрагує складності взаємодії з користувацьким інтерфейсом, надаючи спрощений і стандартизований спосіб роботи з елементами інтерфейсу. Він зазвичай включає методи для створення та управління

елементами інтерфейсу, встановлення властивостей і стилів, обробки взаємодій користувача та оновлення інтерфейсу залежно від логіки програми або змін даних.

За допомогою користувацького API розробники можуть створювати багатофункціональні та інтерактивні користувацькі інтерфейси, налаштовувати вигляд і поведінку своїх програм, а також забезпечувати безперешкодну взаємодію між користувачами та базовою системою або програмою [20].

При розробці інтерфейсу для даного проекту було надано увагу зручності і простоті для безпроблемного користування сервісом користувачем. Основні елементи інтерфейсу програмної системи наступні:

При заході на веб сайт, користувачу доступні дві можливості: авторизація в системі та реєстрація в системі. Автоматично при заході на веб сайт, користувачеві відображається сторінка авторизації.

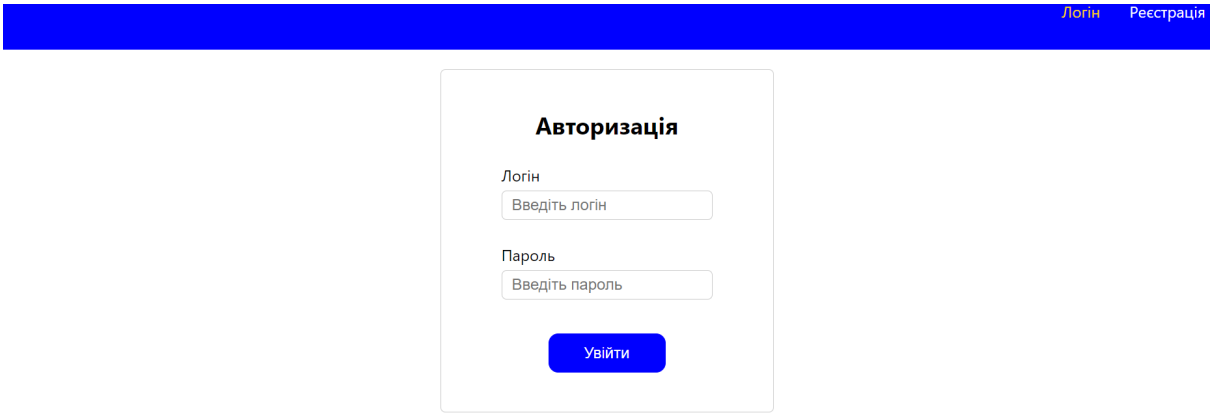


Рисунок 31 – Сторінка авторизації

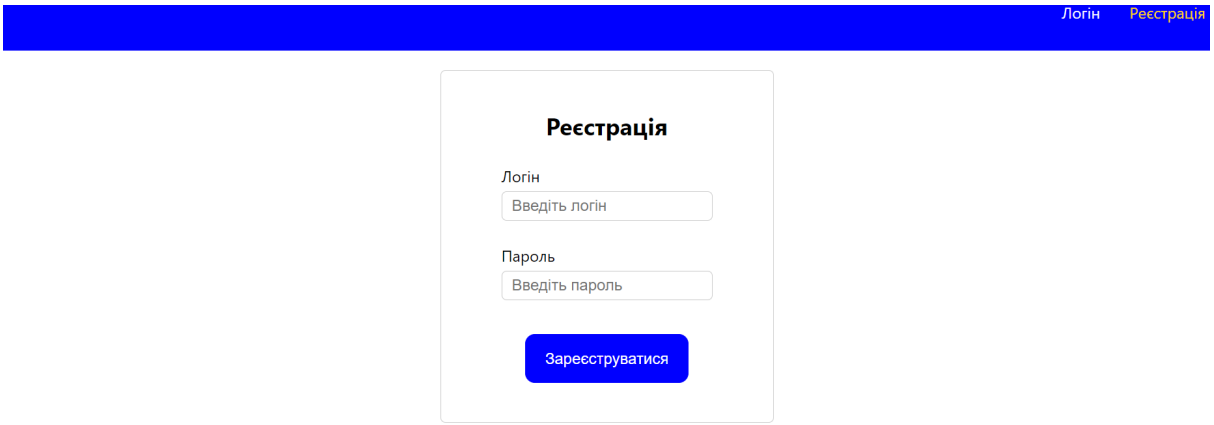


Рисунок 32 – Сторінка реєстрації

Для того щоб користувач отримав можливості використовувати функціонал додатку, йому потрібно авторизуватися. Якщо користувач не має облікового запису, то він попередньо повинен зареєструватися. Кожен користувач, який реєструється, автоматично стає адміном в своїй системі. Адмін в подальшому може створювати працівників, в яких є обмежений функціонал.

При успішній авторизації, адміна перенаправляє на сторінку зі списком столів. Також зверху сайту можна побачити шапку сайту з різними посиланнями, такі як: «Столи», «Меню», «Статистика», «Імпорт/Експорт», «Працівники». На сторінці столів відображається список столів, а також форма для додавання столів. Крім того, кожен стіл має ім’я і статус, а також біля кожного стола знаходиться кнопка для зручного і швидкого видалення стола.

Рисунок 33 – Порожня сторінка столів

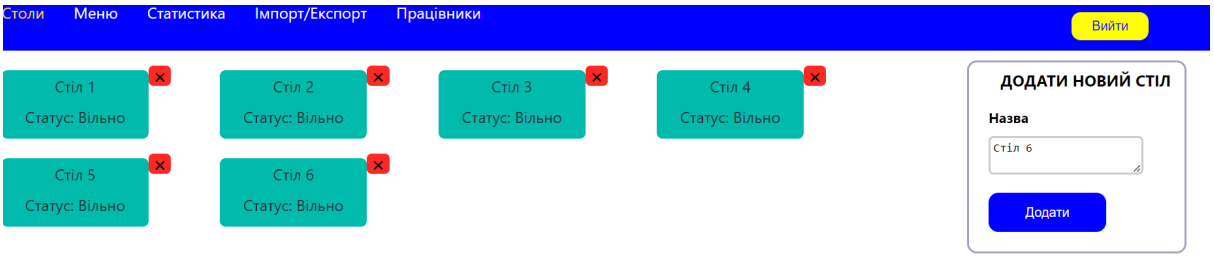


Рисунок 34 – Сторінка зі столами

На сторінці меню знаходиться таблиця зі стравами, форма для додавання страв, а також форма для додавання груп страв. Крім того над таблицею страв, знаходиться поле для пошуку страв, а також навпроти кожної страви і групи знаходиться кнопка для відповідного видалення.

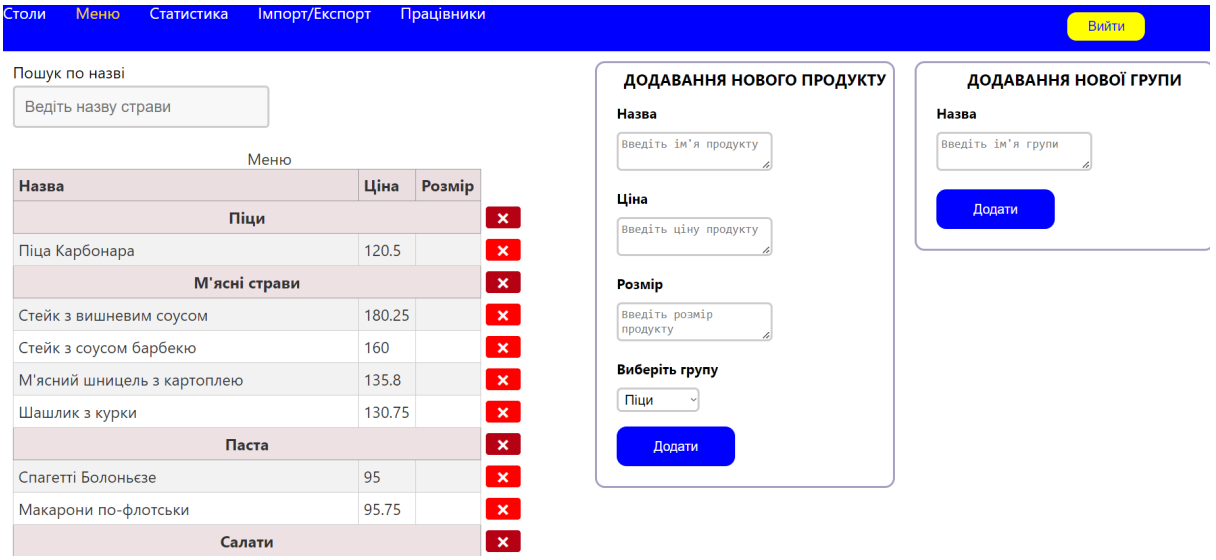


Рисунок 35 – Сторінка меню

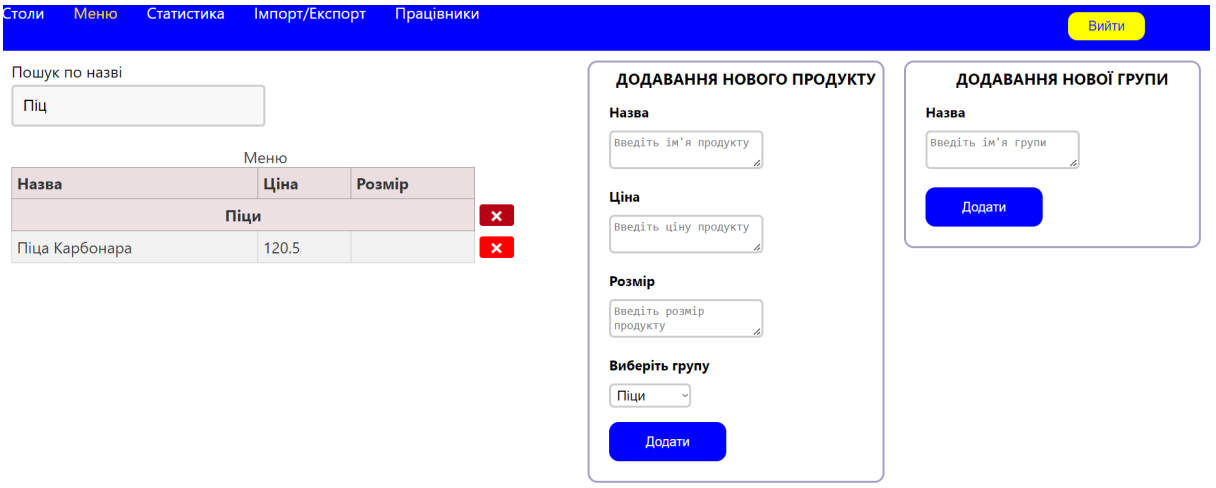


Рисунок 36 – Результат пошуку страви

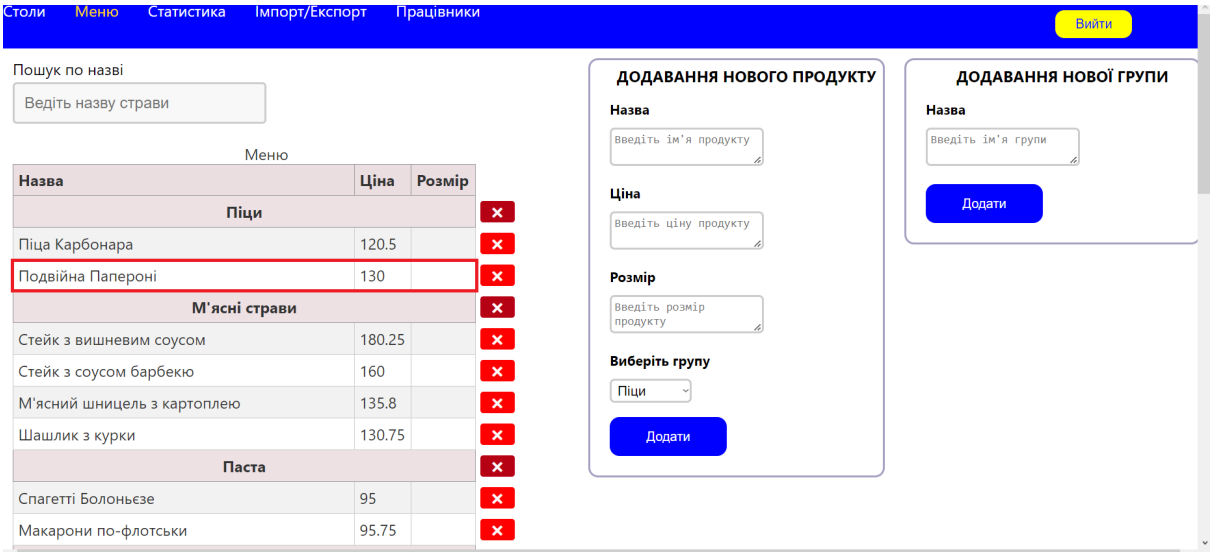


Рисунок 37 – Результат додавання страви

На сторінці статистики відображається статистика замовлень. На цій сторінці доступно дві статистики: загальна кількість замовлень страв та загальна кількість замовлень товарів згрупованих по групах. Для зручного переглядання адмін може наводити на конкретний графік і зрозуміти кількість страв.

Рисунок 38 – Загальна кількість замовлень страв

Рисунок 39 – Загальна кількість замовлень товарів згрупованих по групах

На сторінці Імпорт/Експорт адмін має можливість імпортувати або експортувати меню з або у файл формату .csv. При виникненні проблеми в процесі імпорту, з’явиться помилка, яка допоможе адміну зрозуміти,що не пішло не так.

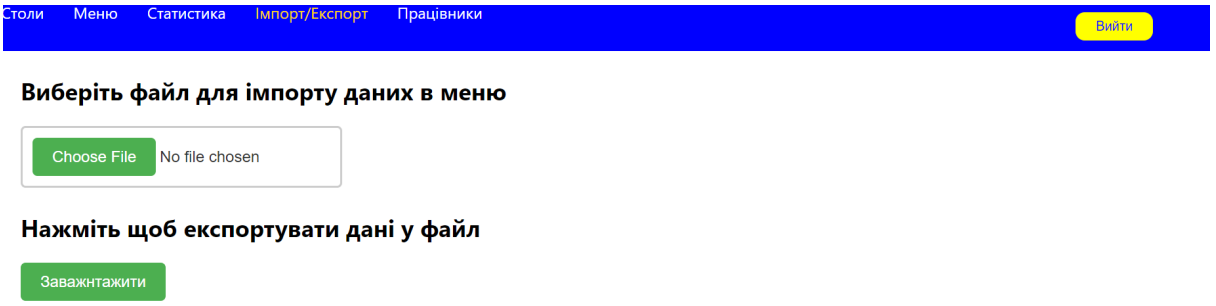


Рисунок 40 – Сторінка імпорту/експорту

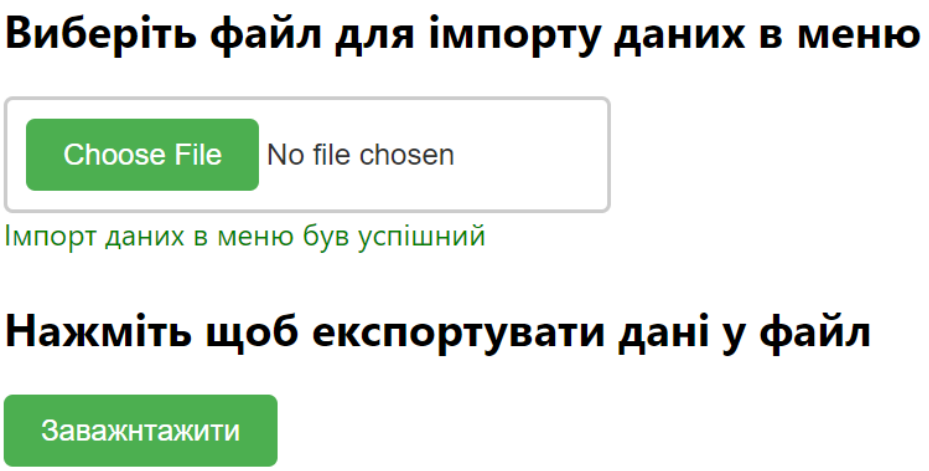


Рисунок 41 – Результат успішного імпорту

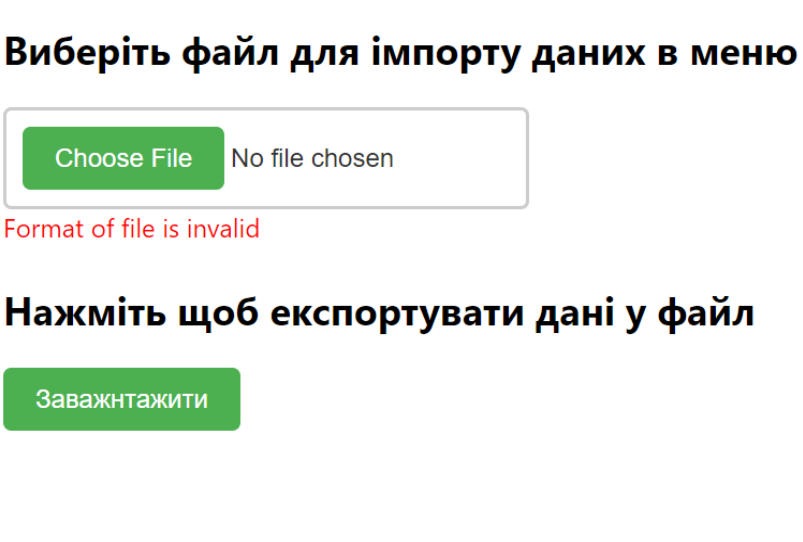


Рисунок 42 – Результат не успішного імпорту у зв’язку невірного формату файлу

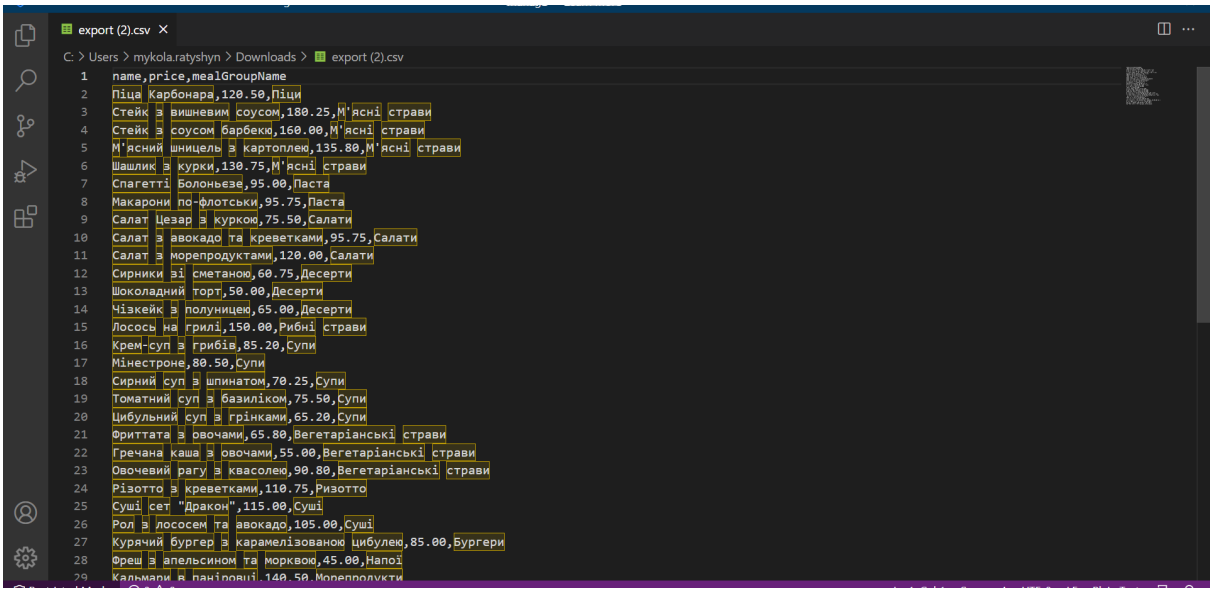


Рисунок 43 – Вміст експортованого файлу

На сторінці Працівники, адмін може створювати або видаляти працівників, а також переглядати їх у таблиці. Крім того на сторінці є форма для додавання працівників. Працівники мають обмежений функціонал, а саме вони можуть тільки переглядати список столів та керувати замовленнями.

Рисунок 44 – Сторінка працівників

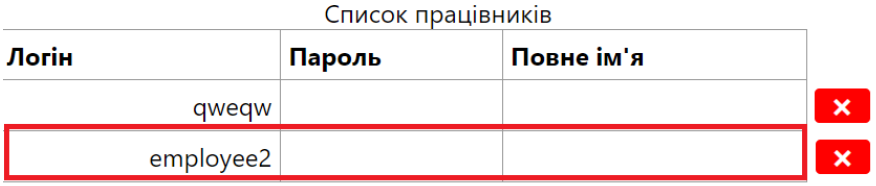


Рисунок 45 – Результат додавання працівника

Як було згадано вище у працівника обмежений функціонал. Працівник може тільки переглядати столи та керувати замовленням.

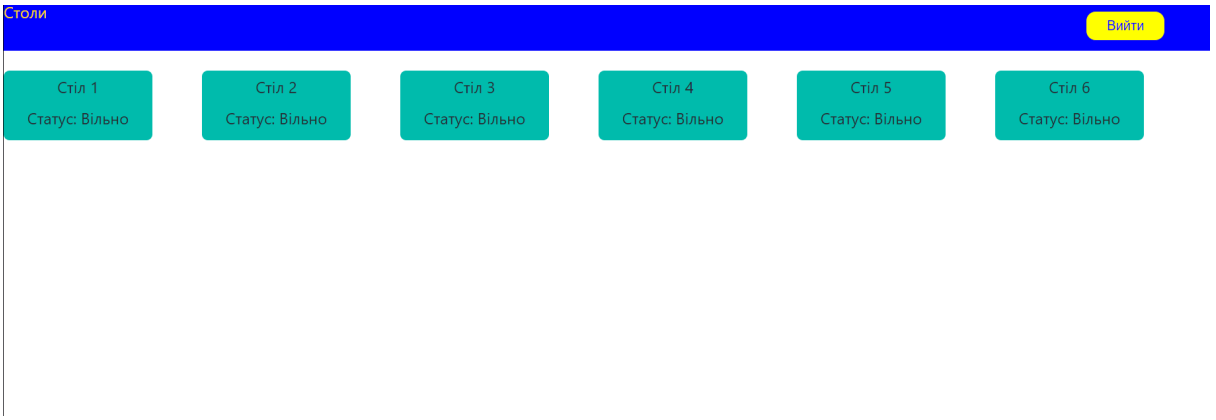


Рисунок 46 – Вигляд додатку для працівника

На сторінці замовлення знаходиться випадаюче меню для зміни статусу та дві таблички: одна представляє поточне замовлення, а друга меню для вибору страв. А також на сторінці знаходиться кнопка для обрахування замовлення.

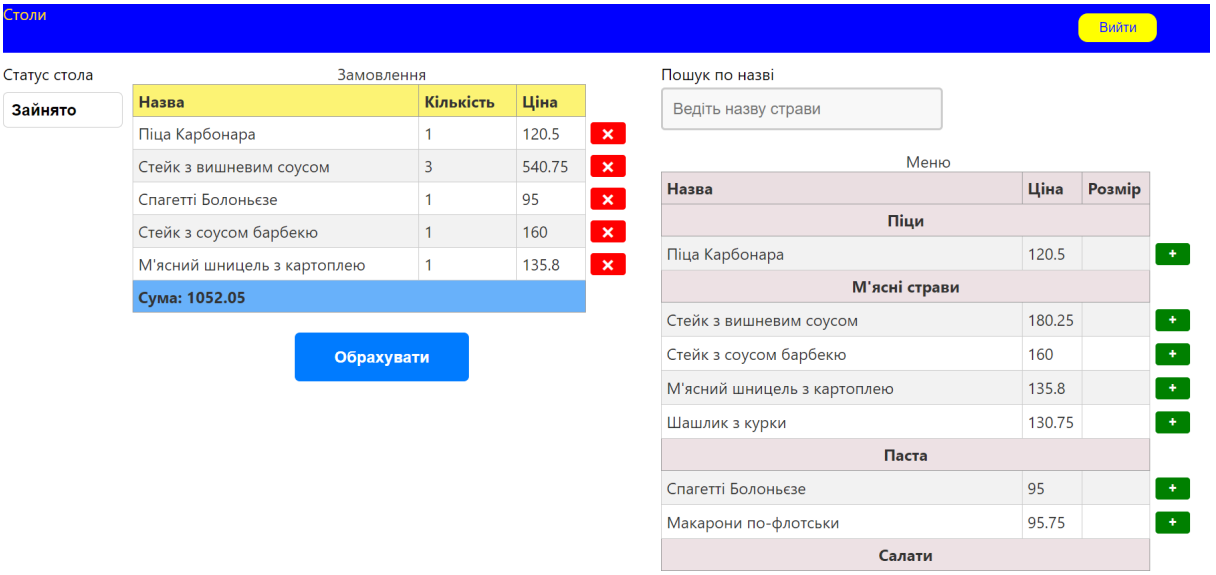


Рисунок 47 – Сторінка замовлення

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Естетичне оформлення та ергономічне дослідження робочого місця оператора

Проєктування та розробка веб-додатку передбачає безперервну роботу з персональним комп’ютером, тому вкрай важливо правильно організувати робоче місце оператора ПК. Ергономічно оптимізована робоча зона відіграє важливу роль у сприянні здоров’ю, комфорту та продуктивності людей, які проводять тривалий час за комп’ютером.

Ергономіка досліджує, розробляє та дає рекомендації щодо конструювання, виготовлення та експлуатації технічних засобів, які забезпечують людині в процесі праці необхідні зручності, зберігають її сили та працездатність [21].

Робочі місця працівників, які обладнані комп’ютерами пристроями, повинні відповідати вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце під час виконання робіт стоячи» [22] та ДСанПІН 3.3.2.007-98 «Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [23].

Основні положення ДСТУ 8604:2015:

Конструкція робочого місця та взаємне розташування всіх його елементів (сидіння, органів керування, засобів відображення інформації тощо) повинні відповідати антропометричним, фізіологічним і психологічним вимогам, а також характеру виконуваної роботи.

Конструкцією робочого місця повинно бути забезпечено виконання трудових операцій у межах зони досяжності моторного поля.

Висоти сидіння та підставки для ніг (у разі нерегульованої висоти робочої поверхні). У цьому разі висоту робочої поверхні встановлюють за номограмою для працюючих зростом 1800 мм. Оптимальна робоча поза для менших за зростом працюючих досягається збільшенням висоти робочого сидіння й підставки для ніг на величину, що дорівнює різниці між висотою робочої поверхні для працюючого

зростом 1800 мм і висотою робочої поверхні, оптимальної для зросту даного працюючого.

Підставка для ніг повинна бути регульованою по висоті. Її ширина повинна бути не менше ніж 300 мм, довжина - не менше ніж 400 мм. Поверхня підставки повинна бути рифленою. По передньому краю доцільно передбачати бортик висотою 10 мм.

Основні положення ДСанПІН 3.3.2.007-98:

Площа на одне робоче місце має становити не менше ніж 6,0 м2, а об'єм не менше ніж 20,0 м3.

Приміщення для роботи повинні мати природне та штучне освітлення.

Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і зебезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%.

Для внутрішнього оздоблення приміщень слід використовувати дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі 0,7-0,8, для стін 0,5-0,6.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50 до 90 градусів з вертикаллю в повздовжній та поперечній площинах має становити не більше ніж 200 кд/м2, захисний кут світильників - не менше ніж 40 градусів

При розміщенні робочих столів, відстані між бічними поверхнями: 1,2 м, відстань від тильної поверхні одного до екрана іншого - 2,5 м.

Показник засліплення у разі використання джерел загального штучного освітлення у виробничих приміщеннях має не перевищувати 20.

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору працюючих з екранами. При цьому співвідношення яскравості робочих поверхонь має бути не більшим ніж 3:1, а співвідношення яскравості робочих поверхонь та поверхонь стін, обладнання тощо - 5:1.

4.2 Долікарська допомога при переломах

Долікарська допомога – це комплекс медичних заходів, спрямованих на надання медичної допомоги при невідкладних станах, які стаються на виробництві, у побуті, під час дорожньо-транспортних пригод, катастроф, техногенних аварій та при гострих неврологічних, терапевтичних, хірургічних та термінальних станах. Не надання долікарської допомоги при нещасних випадках, раптових гострих захворюваннях людини призводить до тяжких наслідків, аж до летальних. Своєчасна допомога відіграє важливу роль у подальшому лікуванні потерпілих і хворих, сприяє скороченню термінів їх медичної та трудової реабілітації [24].

Перелом (crisis) порушення цілісності кісток. Переломи бувають травматичні і патологічні, закриті (без пошкоджень шкіри) і відкриті (шкіра пошкоджена в зоні перелому). Відкриті переломи небезпечні тим, що вони можуть інфікувати уламки і розвинути остеомієліт. Переломи бувають повні і неповні. При неповному переломі порушується якась частина поперечних кісток, з являються тріщини. Переломи за формою поділяються на поперечні, косі, спіральні, осколочні, від стиснення, компресійні тощо. Буває зміщення кісткових уламків під кутом, зміщення по довжині, бокові зміщення.

Переломам притаманні різкий біль, порушення функції ураженої ділянки, набряк і крововилив в зоні перелому, вкорочення кінцівки, ненормальна патологічна рухомість кістки. При переломах спостерігається нерівність кісток, хрумтіння при натисканні, у випадку відкритого перелому виступає уламок кістки.

Заходи долікарської допомоги при переломах: фіксація кісток в області перелому; протишокові заходи; транспортування в медпункт.

Основне завдання закріпити пошкоджені кістки, суглоби, зв язані з ними кінцівки в нерухомому і найзручнішому для потерпілого стані.

Іммобілізація зменшує біль. Це основний засіб попередження шоку. Найчастіше зустрічаються переломи кінцівок. Правильна фіксація пошкоджених

кінцівок попереджає зміщення уламків, зменшує пошкодження судин, нервів, м'язів і шкіри гострими краями уражених кісток. Накладають транспортні шини з підручного твердого матеріалу. Кінцівки біля рани, перелому обробляють йодом, антисептиком і накладають асептичну пов'язку при відкритому переломі.

При наданні допомоги не треба намагатись встановити, є, чи немає перелому: мацати місце ушкодження, примушувати потерпілого рухати, піднімати або згинати кінцівку. Такі дії можуть різко підсилити біль, спричинити зміщення і ушкодження м'яких тканин. Для забезпечення нерухомості зламаної кінцівки застосовують спеціальні дротяні або фанерні (дерев’яні) шини. Шина повинна бути накладена так, щоб були надійно іммобілізовані два сусідні з місцем ушкодження суглоби (вище і нижче), а якщо перелом плеча або стегна, то три суглоби. Накладають шину поверх одягу або кладуть під неї що-небудь м яке вату, шарф, рушник. Накладену шину необхідно прикріпити до кінцівки бинтом, рушником, ременем. Як шину можна використати дошку, палицю, лижу тощо. Таку імпровізовану шину необхідно прикласти з двох протилежних сторін уздовж ушкодженої кінцівки і обгорнути бинтом. Шина повинна бути накладена так, щоб центр її знаходився на рівні перелому, а кінці накладалися на сусідні суглоби по обидві сторони перелому. Фіксація відкритого перелому потребує дотримання додаткових умов: не можна накладати шину на місце відкритого перелому, а слід перебинтовувати її поверх одягу (взуття) і, крім того, підкласти під неї що небудь м’яке, попередньо зупинивши кровотечу.

При транспортуванні шину надійно закріплюють, щоб зафіксувати область перелому; під шину підкладають вату, тканину; фіксують 2 суглоби вище і нижче перелому. Правильна фіксація запобігає шоку [25].

ВИСНОВКИ

Під час виконання дипломної роботи було створено систему для ресторану з використанням сучасних технологій та найкращих практик розробки програмного забезпечення. Розроблена система успішно відповідає поставленим вимогам та забезпечує ефективне управління рестораном.

Система ресторану дозволяє автоматизувати процеси замовлення страв, обробки платежів, управління складом продуктів, розкладом персоналу та забезпеченням високоякісного обслуговування клієнтів. Вона включає функціонал для зберігання та управління меню, резервування столиків, обліку складу продуктів, генерації звітів та аналітики, а також інтеграцію з платіжними системами.

Проектування та розробка системи включали в себе використання клієнт-серверної архітектури, розробку веб-інтерфейсу для замовлення страв та адміністрування системи, інтеграцію з базою даних для зберігання даних ресторану, а також застосування безпекових механізмів для захисту конфіденційної інформації.

Під час розробки системи було використано широкий набір технологій, таких як HTML, CSS, JavaScript для розробки фронтенду, та Java для розробки бекенду. Також були використані фреймворки та бібліотеки, які сприяють прискоренню розробки та покращенню якості програмного продукту.

Результати роботи над дипломною роботою демонструють успішне створення функціональної та надійної системи для ресторану, яка забезпечує ефективне управління ресторанним бізнесом та задоволення потреб клієнтів. Розроблена система може бути використана як основа для подальшого розширення та вдосконалення функціоналу ресторану.

СПИСОК ПОСИЛАНЬ

1. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с.

2. Pressman R. S., Maxim B. Software Engineering: A Practitioner's Approach. McGraw-Hill Education, 2019. 704 с.

3. Офіційний сайт Toast Pos. URL: https://pos.toasttab.com

4. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Pearson Education, Limited, 2018.

5. Клієнт-серверна архітектура. URL: https://www.wikiwand.com/uk/Клієнт-серверна_архітектура

6. Офіційна документація Java. URL: https://docs.oracle.com/javase/specs/jls/se11/html/jls-1.html

7. Офіційна документація Spring. URL: https://spring.io

8. Software architecture patterns / Mark Richards. URL: www.oreilly.com/ideas/software-architecture-patterns/page/2/layered-architecture

9. Офіційна документація PostgreSQL. URL:https://www.postgresql.org

10. Офіційна документація React. URL:https://uk.legacy.reactjs.org

11. Офіційна документація HTML. URL: https://www.w3.org/TR/html53/

12. Офіційна документація CSS. URL: https://www.w3.org/Style/CSS/current-work

13. Офіційна документація Redux. URL: https://redux.js.org

14. Офіційна документація Heroku. URL: https://www.heroku.com/what

15. Офіційна документація Intellij Idea. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

16. Official UML Specification. URL: http://www.omg.org/spec/UML/

17. Visual Paradigm: Class DiagramGuide. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

18. Microsoft Docs: Multilayered architecture. URL: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/multilayered

19. Software Testing. URL:https://www.ibm.com/topics/software-testing

20. User API. URL: https://help.userguiding.com/en/articles/4493538-userguiding-user-api

21. В. Ц. Жидецький, В. С. Джигирей, О. В. Мельников. Основи охорони праці. — Вид. 2-е, стереотипне. — Львів: Афіша, 2004., 16 с.

22. ДСТУ 8604:2015. URL: https://zakon.rada.gov.ua/rada/show/v0204774-15#Text

23. ДСанПІН 3.3.2.007-98. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text

24. Перша долікарська допомога. URL: https://www.pharmencyclopedia.com.ua/article/790/persha-dolikarska-dopomoga

25. «Безпека життєдіяльності – Березюк О. В., Лемешев М. С, Вінниця 2011, 98 с.

ДОДАТКИ

ДОДАТОК А

Лістинг коду:

Файл Meal.java

```
package com.restaurant.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import java.math.BigDecimal;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
public class Meal {
    @Id
    @GeneratedValue
    private Long id;

    private String name;

    private BigDecimal price;

    @ManyToOne
    @JoinColumn(name = "group_id")
    private MealGroup mealGroup;
}
```

Файл MealGroup.java

```
package com.restaurant.model;
```

```
import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@RequiredArgsConstructor
@Entity
public class MealGroup {
    @Id
    @GeneratedValue
    private Long id;

    private String name;

        @OneToMany(mappedBy = "mealGroup", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<Meal> menu = new ArrayList<>();

    @ManyToOne
    @JoinColumn(name = "restaurant_id")
    private Restaurant restaurant;

    public void removeAllMeals(){
        menu.clear();
    }

    public void removeMeal(Meal meal){
        menu.remove(meal);
        meal.setMealGroup(null);
    }

    public void addMeal(Meal meal){
        menu.add(meal);
```

```
        meal.setMealGroup(this);
    }
}
```

Файл Order.java

```
package com.restaurant.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@RequiredArgsConstructor
@Entity
@Table(name = "orders")
public class Order {
    @Id
    @GeneratedValue
    private Long id;
    @ElementCollection
     @CollectionTable(name = "order_meals", joinColumns = @JoinColumn(name
= "order_id"), foreignKey = @ForeignKey(name = "order_meal_orders_fk"))
    private List<OrderMeal> meals;

    private LocalDateTime createdAt;

    @OneToOne
    private Restaurant restaurant;

    public void addMeals(List<OrderMeal> meals){
        this.meals = new ArrayList<>();
```

```
        this.meals.addAll(meals);
    }

    public BigDecimal getTotalPrice(){
        if(meals != null) {
            return meals.stream()
                    .map(OrderMeal::getPrice)
                    .reduce(BigDecimal::add)
                    .orElse(BigDecimal.ZERO);
        }
        return BigDecimal.ZERO;
    }
}
```

Файл Receipt.java

```
package com.restaurant.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;

import java.math.BigDecimal;
import java.util.List;

@Getter
@Setter
@RequiredArgsConstructor
@Entity
public class Receipt {

    @Id
    @Column(name = "table_id")
    private Long id;

        @OneToMany(mappedBy = "receipt", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<ReceiptMeal> meals;
```

```
@OneToOne
@MapsId
@JoinColumn(name = "table_id")
private RestaurantTable table;

public BigDecimal getTotalPrice(){
    if(meals != null) {
        return meals.stream()
                .map(ReceiptMeal::getTotalPrice)
                .reduce(BigDecimal::add)
                .orElse(BigDecimal.ZERO);
    }
    return BigDecimal.ZERO;
}

public void removeAllMeals(){
    this.meals.clear();
}

public void removeTable(){
    this.table.setReceipt(null);
    this.table = null;
}

public void addAllMeals(List<ReceiptMeal> meals){
    this.meals.addAll(meals);
}

public void addMeal(ReceiptMeal meal){
    this.meals.add(meal);
    meal.setReceipt(this);
}

public void removeMeal(ReceiptMeal receiptMeal){
    this.meals.remove(receiptMeal);
    receiptMeal.setReceipt(null);
    receiptMeal.setMeal(null);
}
```

```
    public void addTable(RestaurantTable table){
        this.table = table;
        table.setReceipt(this);
    }
}
```

Файл Restaurant.java

```
package com.restaurant.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

@Getter
@Setter
@RequiredArgsConstructor
@Entity
public class Restaurant {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

        @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<MealGroup> mealGroups;

        @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<RestaurantTable> tables;
```

```
        @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)
    private Set<User> owners = new HashSet<>();

    public void removeAllMealGroups(){
        mealGroups.clear();
    }

    public void addOwner(User user){
        owners.add(user);
        user.setRestaurant(this);
    }
}
```

Файл RestaurantTable.java

```
package com.restaurant.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

@Getter
@Setter
@RequiredArgsConstructor
@Entity
public class Restaurant {

    @Id
    @GeneratedValue
    private Long id;

    private String name;
```

```
        @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<MealGroup> mealGroups;

        @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<RestaurantTable> tables;

        @OneToMany(mappedBy = "restaurant", cascade = CascadeType.ALL,
orphanRemoval = true)
    private Set<User> owners = new HashSet<>();

    public void removeAllMealGroups(){
        mealGroups.clear();
    }

    public void addOwner(User user){
        owners.add(user);
        user.setRestaurant(this);
    }
}
```

Файл User.java

```
package com.restaurant.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import java.util.List;

@Entity
@Getter
@Setter
@RequiredArgsConstructor
```

```
@Table(name = "restaurant_user")
public class User {
    @Id
    @GeneratedValue
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    private String password;

    private Role role;

    @OneToOne(cascade = CascadeType.ALL)
    private Restaurant restaurant;

    @OneToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<User> employees;

    @OneToOne(cascade = CascadeType.ALL)
    private User admin;

    public void addEmployee(User user){
        employees.add(user);
        user.setAdmin(this);
    }

    public void removeEmployee(User user){
        employees.remove(user);
        user.setAdmin(null);
    }

    public void clearEmployees(){
        employees.clear();
    }

    public void addRestaurant(Restaurant restaurant){
        this.restaurant = restaurant;
        restaurant.addOwner(this);
```

```
    }
}
```

Файл CsvService.java

```
package com.restaurant.service;

import com.restaurant.exception.ExportExceptions;
import com.restaurant.exception.UploadExceptions;
import com.restaurant.model.MealGroup;
import com.restaurant.model.Restaurant;
import com.restaurant.service.dto.ImportMealDto;
import com.univocity.parsers.common.processor.BeanListProcessor;
import com.univocity.parsers.csv.CsvParser;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

@Service
@RequiredArgsConstructor
public class CsvService {
    private final CsvParser parser;
    private final BeanListProcessor<ImportMealDto> rowProcessor;
    private static final String CSV_TYPE = "text/csv";
                private    static    final    String    EXCEL_CSV_TYPE    =
"application/vnd.ms-excel";

    private ByteArrayOutputStream byteArrayOutputStream;
    private final RestaurantService restaurantService;
    private final ImportService importService;
    private static final String DELIMITER = ",";
```

```
    public String parseAndSave(MultipartFile file, Long restaurantId) {
                    if   (EXCEL_CSV_TYPE.equals(file.getContentType())   ||
CSV_TYPE.equals(file.getContentType())) {
            try {
                InputStream inputStream = file.getInputStream();
                parser.parse(inputStream);
                clearDatabaseFromPreviousMeals(restaurantId);
                                rowProcessor.getBeans().forEach(dto   ->
importService.storeToDatabase(dto, restaurantId));
                  return "Save of file " + file.getOriginalFilename() + "
was successful";
            } catch (IOException e) {
                                                                throw    new
UploadExceptions(UploadExceptions.Error.SAVE_WAS_NOT_SUCCESSFUL);
            }
        }
                                                            throw       new
UploadExceptions(UploadExceptions.Error.INVALID_FILE_FORMAT);
    }

    private void clearDatabaseFromPreviousMeals(Long restaurantId) {
                                          Restaurant     restaurant     =
restaurantService.getRestaurantById(restaurantId);
        restaurant.getTables()
                .forEach(table -> table.getReceipt().removeAllMeals());
        Restaurant savedRestaurant = restaurantService.save(restaurant);
        savedRestaurant.removeAllMealGroups();
        restaurantService.save(savedRestaurant);
    }

    public byte[] export(Long restaurantId) {
        byteArrayOutputStream = new ByteArrayOutputStream();
        addHeader();
        addRows(restaurantId);
        return byteArrayOutputStream.toByteArray();
    }
```

```
    private void addHeader() {
        Arrays.stream(createHeader())
                .map(String::getBytes)
                .forEach(this::write);
    }

    private String[] createHeader() {
           return new String[]{"name" + DELIMITER, "price" + DELIMITER,
"mealGroupName"};
    }

    private void addRows(Long restaurantId) {
                                          Restaurant     restaurant     =
restaurantService.getRestaurantById(restaurantId);
        restaurant.getMealGroups().stream()
                .map(this::createRow)
                .forEach(row -> row.forEach(this::write));
    }

    private List<byte[]> createRow(MealGroup mealGroup) {
        List<byte[]> bytesList = new ArrayList<>();

        mealGroup.getMenu()
                .forEach(meal -> {
                                                        byte[]   price   =
meal.getPrice().toString().getBytes(StandardCharsets.UTF_8);

                    bytesList.add("\n".getBytes());

bytesList.add(meal.getName().getBytes(StandardCharsets.UTF_8));
                    bytesList.add(DELIMITER.getBytes());
                    bytesList.add(price);
                    bytesList.add(DELIMITER.getBytes());

bytesList.add(mealGroup.getName().getBytes(StandardCharsets.UTF_8));
                });

        return bytesList;
    }
```

```
    private void write(byte[] bytes) {
        try {
            byteArrayOutputStream.write(bytes);
        } catch (IOException e) {
                                                                    throw     new
ExportExceptions(ExportExceptions.Error.WRITE_TO_CSV_IS_BAD);
        }
    }

}
```

Файл MealGroupService.java

```
package com.restaurant.service;

import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.MealGroup;
import com.restaurant.repository.MealGroupRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class MealGroupService {
    private final MealGroupRepository mealGroupRepository;

    public MealGroup getMealGroupById(Long id){
            return mealGroupRepository.findById(id).orElseThrow(() -> new
EntityByIdNotFoundException(id));
    }

    public MealGroup save(MealGroup mealGroup) {
        return mealGroupRepository.save(mealGroup);
    }
```

```
    public void deleteById(Long id) {
        mealGroupRepository.deleteById(id);
    }

    public List<MealGroup> getAll(){
        return mealGroupRepository.findAll();
    }

    public Optional<MealGroup> getMealGroupByName(String name){
        return mealGroupRepository.findByName(name);
    }
}
```

Файл MealService.java

```
package com.restaurant.service;

import com.restaurant.controller.request.UpdateMealRequest;
import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.Meal;
import com.restaurant.repository.MealRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class MealService {

    private final MealRepository mealRepository;

    public Meal save(Meal meal){
        return mealRepository.save(meal);
    }

    public void deleteById(Long id){
        mealRepository.deleteById(id);
    }
```

```
    public Meal getMealById(Long id){
                return  mealRepository.findById(id).orElseThrow(()  ->  new
EntityByIdNotFoundException(id));
    }

    public Meal updateMeal(Long id, UpdateMealRequest request){
        Meal mealById = getMealById(id);
        mealById.setName(request.getName());
        mealById.setPrice(request.getPrice());
        return save(mealById);
    }
}
```

Файл OrderService.java

```
package com.restaurant.service;


import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.Order;
import com.restaurant.repository.OrderRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class OrderService {
    private final OrderRepository orderRepository;

    public Order save(Order order) {
        return orderRepository.save(order);
    }

    public Order getOrderById(Long id) {
              return  orderRepository.findById(id).orElseThrow(()  ->  new
EntityByIdNotFoundException(id));
```

```
    }

    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }
}

package com.restaurant.service;

import com.restaurant.controller.request.ReceiptAddMealRequest;
import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.*;
import com.restaurant.repository.ReceiptMealRepository;
import com.restaurant.repository.ReceiptRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ReceiptService {
    private final RestaurantTableService tableService;
    private final ReceiptRepository receiptRepository;
    private final OrderService orderService;
    private final MealService mealService;
    private final ReceiptMealRepository receiptMealRepository;

    public Receipt save(Receipt receipt) {
                                        RestaurantTable     tableById     =
tableService.getTableById(receipt.getId());
        receipt.addTable(tableById);
        return receiptRepository.save(receipt);
    }

    public void deleteById(Long id) {
        Receipt receiptById = getReceiptById(id);
        receiptById.removeAllMeals();
```

```
        receiptRepository.delete(receiptById);
    }

    public Receipt getReceiptById(Long id) {
            return receiptRepository.findById(id).orElseThrow(() -> new
EntityByIdNotFoundException(id));
    }

    public Receipt update(Receipt receipt, Long receiptId) {
        Receipt receiptById = getReceiptById(receiptId);
        receiptById.removeAllMeals();
        receiptById.addAllMeals(receipt.getMeals());
        return receiptRepository.save(receiptById);
    }

    public Order countTheReceipt(Long receiptId) {
        Receipt receiptById = getReceiptById(receiptId);
        Order order = new Order();
        List<OrderMeal> orderMeals = receiptById.getMeals().stream()
                                                .map(receiptMeal -> new
OrderMeal(receiptMeal.getMeal().getName(),
                        receiptMeal.getMeal().getPrice(),
                        receiptMeal.getAmount(),
                        receiptMeal.getMeal().getMealGroup().getName()))
                .toList();
        order.addMeals(orderMeals);
        order.setCreatedAt(LocalDateTime.now());
        order.setRestaurant(receiptById.getTable().getRestaurant());
        Order savedOrder = orderService.save(order);
                tableService.changeStatus(receiptById.getTable().getId(),
TableStatus.FREE);
        deleteById(receiptId);
        return savedOrder;
    }

    public Receipt addMealToReceipt(ReceiptAddMealRequest request) {
        Receipt receipt = getReceiptById(request.getReceiptId());
        Meal meal = mealService.getMealById(request.getMealId());
        receipt.getMeals().stream()
```

```
                                                .filter(receiptMeal   ->
receiptMeal.getMeal().getId().equals(meal.getId()))
                .findFirst()
                .ifPresentOrElse((receiptMeal) -> {
                    receiptMeal.setAmount(receiptMeal.getAmount() + 1);
                }, () -> {
                    ReceiptMeal receiptMeal = new ReceiptMeal();
                    receiptMeal.setReceipt(receipt);
                    receiptMeal.setMeal(meal);
                    receiptMeal.setAmount(1L);
                                        ReceiptMeal  savedReceiptMeal  =
receiptMealRepository.save(receiptMeal);
                    receipt.addMeal(savedReceiptMeal);
                });
        return save(receipt);
    }

    public Receipt removeMealFromReceipt(Long receiptId, Long mealId) {
        Receipt receipt = getReceiptById(receiptId);
        Meal meal = mealService.getMealById(mealId);
        receipt.getMeals().stream()
                                                .filter(receiptMeal   ->
receiptMeal.getMeal().getId().equals(meal.getId()))
                .findFirst()
                .ifPresent((receiptMeal) -> {
                    receiptMeal.setAmount(receiptMeal.getAmount() - 1);
                    if(receiptMeal.getAmount() <= 0){
                        receipt.removeMeal(receiptMeal);
                    }
                });
        return save(receipt);
    }

      public Receipt updateMealAmount(Long receiptId, Long mealId, Long
amount) {
        Receipt receipt = getReceiptById(receiptId);
        receipt.getMeals().stream()
                                                .filter(receiptMeal   ->
receiptMeal.getMeal().getId().equals(mealId))
```

```
                .findFirst()
                .ifPresent(receiptMeal -> {
                    receiptMeal.setAmount(amount);
                });
        return save(receipt);
    }
}
```

Файл ReceiptService.java

```
package com.restaurant.service;

import com.restaurant.controller.request.ReceiptAddMealRequest;
import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.*;
import com.restaurant.repository.ReceiptMealRepository;
import com.restaurant.repository.ReceiptRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ReceiptService {
    private final RestaurantTableService tableService;
    private final ReceiptRepository receiptRepository;
    private final OrderService orderService;
    private final MealService mealService;
    private final ReceiptMealRepository receiptMealRepository;

    public Receipt save(Receipt receipt) {
                                        RestaurantTable     tableById     =
tableService.getTableById(receipt.getId());
        receipt.addTable(tableById);
        return receiptRepository.save(receipt);
    }
```

```
    public void deleteById(Long id) {
        Receipt receiptById = getReceiptById(id);
        receiptById.removeAllMeals();
        receiptRepository.delete(receiptById);
    }

    public Receipt getReceiptById(Long id) {
            return receiptRepository.findById(id).orElseThrow(() -> new
EntityByIdNotFoundException(id));
    }

    public Receipt update(Receipt receipt, Long receiptId) {
        Receipt receiptById = getReceiptById(receiptId);
        receiptById.removeAllMeals();
        receiptById.addAllMeals(receipt.getMeals());
        return receiptRepository.save(receiptById);
    }

    public Order countTheReceipt(Long receiptId) {
        Receipt receiptById = getReceiptById(receiptId);
        Order order = new Order();
        List<OrderMeal> orderMeals = receiptById.getMeals().stream()
                                                .map(receiptMeal -> new
OrderMeal(receiptMeal.getMeal().getName(),
                        receiptMeal.getMeal().getPrice(),
                        receiptMeal.getAmount(),
                        receiptMeal.getMeal().getMealGroup().getName()))
                .toList();
        order.addMeals(orderMeals);
        order.setCreatedAt(LocalDateTime.now());
        order.setRestaurant(receiptById.getTable().getRestaurant());
        Order savedOrder = orderService.save(order);
                tableService.changeStatus(receiptById.getTable().getId(),
TableStatus.FREE);
        deleteById(receiptId);
        return savedOrder;
    }
```

```
    public Receipt addMealToReceipt(ReceiptAddMealRequest request) {
        Receipt receipt = getReceiptById(request.getReceiptId());
        Meal meal = mealService.getMealById(request.getMealId());
        receipt.getMeals().stream()
                                                .filter(receiptMeal   ->
receiptMeal.getMeal().getId().equals(meal.getId()))
                .findFirst()
                .ifPresentOrElse((receiptMeal) -> {
                    receiptMeal.setAmount(receiptMeal.getAmount() + 1);
                }, () -> {
                    ReceiptMeal receiptMeal = new ReceiptMeal();
                    receiptMeal.setReceipt(receipt);
                    receiptMeal.setMeal(meal);
                    receiptMeal.setAmount(1L);
                                        ReceiptMeal  savedReceiptMeal  =
receiptMealRepository.save(receiptMeal);
                    receipt.addMeal(savedReceiptMeal);
                });
        return save(receipt);
    }

    public Receipt removeMealFromReceipt(Long receiptId, Long mealId) {
        Receipt receipt = getReceiptById(receiptId);
        Meal meal = mealService.getMealById(mealId);
        receipt.getMeals().stream()
                                                .filter(receiptMeal   ->
receiptMeal.getMeal().getId().equals(meal.getId()))
                .findFirst()
                .ifPresent((receiptMeal) -> {
                    receiptMeal.setAmount(receiptMeal.getAmount() - 1);
                    if(receiptMeal.getAmount() <= 0){
                        receipt.removeMeal(receiptMeal);
                    }
                });
        return save(receipt);
    }

      public Receipt updateMealAmount(Long receiptId, Long mealId, Long
amount) {
```

```
        Receipt receipt = getReceiptById(receiptId);
        receipt.getMeals().stream()
                                                .filter(receiptMeal   ->
receiptMeal.getMeal().getId().equals(mealId))
                .findFirst()
                .ifPresent(receiptMeal -> {
                    receiptMeal.setAmount(amount);
                });
        return save(receipt);
    }
}
```

Файл RestaurantService.java

```
package com.restaurant.service;

import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.Restaurant;
import com.restaurant.repository.RestaurantRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class RestaurantService {

    private final RestaurantRepository restaurantRepository;

    public Restaurant getRestaurantById(Long id){
           return  restaurantRepository.findById(id).orElseThrow(() -> new
EntityByIdNotFoundException(id));
    }

    public Restaurant save(Restaurant restaurant){
        return restaurantRepository.save(restaurant);
    }

    public void deleteById(Long id){
```

```
        restaurantRepository.deleteById(id);
    }
}
```

Файл RestaurantTableService.java

```
package com.restaurant.service;

import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.*;
import com.restaurant.repository.RestaurantTableRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class RestaurantTableService {
    private final RestaurantTableRepository tableRepository;

    public RestaurantTable save(RestaurantTable table){
        table.createReceipt();
        table.setStatus(TableStatus.FREE);
        return tableRepository.save(table);
    }

    public void deleteById(Long id){
        tableRepository.deleteById(id);
    }

    public RestaurantTable getTableById(Long id){
              return  tableRepository.findById(id).orElseThrow(()  ->  new
EntityByIdNotFoundException(id));
    }

    public RestaurantTable changeStatus(Long id, TableStatus status){
        RestaurantTable tableById = getTableById(id);
        tableById.setStatus(status);
        return tableRepository.save(tableById);
```

```
    }
}
```

Файл UserService.java

```
package com.restaurant.service;

import com.restaurant.controller.dto.UserDto;
import com.restaurant.controller.request.EmployeeRequest;
import com.restaurant.exception.EntityByIdNotFoundException;
import com.restaurant.model.Restaurant;
import com.restaurant.model.Role;
import com.restaurant.model.User;
import com.restaurant.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public User register(UserDto userDto) {
        User user = new User();
        user.setUsername(userDto.getUsername());
        user.setPassword(passwordEncoder.encode(userDto.getPassword()));
        user.setRole(Role.ADMIN);
        Restaurant restaurant = new Restaurant();
        user.setRestaurant(restaurant);
        restaurant.addOwner(user);
        userRepository.save(user);
        return user;
```

```
    }

    public Optional<User> getUserByUsername(String username) {
        return userRepository.findByUsername(username);
    }

           public   User   addEmployee(EmployeeRequest   request,   String
adminUsername) {
        User employee = new User();
        employee.setUsername(request.getUsername());

employee.setPassword(passwordEncoder.encode(request.getPassword()));
        employee.setRole(Role.USER);
        User adminUser = getUserByUsername(adminUsername).orElseThrow();
        employee.addRestaurant(adminUser.getRestaurant());
        userRepository.save(employee);

        adminUser.addEmployee(employee);
        userRepository.save(adminUser);
        return employee;
    }

    public List<User> getAllEmployees(String adminUsername) {
                                                                    return
getUserByUsername(adminUsername).orElseThrow().getEmployees();
    }

    public User getUserById(Long id) {
               return  userRepository.findById(id).orElseThrow(()  ->  new
EntityByIdNotFoundException(id));
    }

    public void deleteById(Long id) {
        User user = getUserById(id);
        if (user.getRole().equals(Role.ADMIN)) {
            user.clearEmployees();
            userRepository.deleteById(id);
        } else {
            User admin = user.getAdmin();
```

```
            admin.removeEmployee(user);
            userRepository.deleteById(id);
            userRepository.save(admin);
        }
    }

    public boolean isUserExist(String username, String password) {
        User user = getUserByUsername(username).orElseThrow();
        return passwordEncoder.matches(password, user.getPassword());
    }
}
```

ДОДАТОК Б