Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____*Комп'ютерних наук*_____
(повна назва)

Кафедра_____*Системотехніки*_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти_____другий (магістерський)_____

Системне проектування інформаційного забезпечення обслуговування відвідувачів мережі підприємств харчування
(тема)

Виконав:
студент_2__курсу, групи ___СПРм-19-2____
____Пироженко М.Ю._____
(прізвище, ініціали)

Спеціальність_122 Комп'ютерні науки _____
_____
(код і повна назва спеціальності)
Тип програми ___*освітньо-наукова*_____
(освітньо-професійна або освітньо-наукова)
Освітня програма __*Системне проектування*__
_____
(повна назва освітньої програми)

Керівник ___Вишняк М.Ю._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____ __Гребеннік І.В.___
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет_____*Комп'ютерних наук*_____

Кафедра_____*Системотехніки*_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність_____122 Комп'ютерні науки _____
(код і повна назва)

Тип програми _____освітньо наукова_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____Системне проектування_____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____*Пироженко Михайло Юрійовичу*_____
(прізвище, ім'я, по батькові)

1. Тема роботи __*Системне проектування інформаційного забезпечення обслуговування__ __відвідувачів мережі підприємств харчування*_____
затверджена наказом університету від _23_ _березень__2021 р. № 389Ст_
2. Термін подання студентом роботи до екзаменаційної комісії _24_ __*травня*__ 2021 р.
3. Вихідні дані до роботи _____*Функції системи: авторизація, перегляд меню, оформлення замовлення, зміна статусу оборонного замовлення, додавання страв меню, перегляд оформлених замовлень, формування звіту, додавання користувачів. Перелік використовуваних програмних засобів: ОС Microsoft Windows XP та вище. Технічне забезпечення: IBM-сумісний ПК з МП Pentium II та вище. Інтегроване середовище розробки Eclipse.*_____
4. Перелік питань, що потрібно опрацювати в роботі _*Вступ. Аналіз та постановка задачі.*_ __*Аналіз нормативно правових документів. Аналіз діяльності з обслуговування клієнтів.*__ __*Визначення сфери застосування системи. Постановка задачі. Проектування системного*_ __*забезпечення. Специфікація вимог до системи. Визначення функціональних вимог.*_____ __*Вимоги до інтерфейсу користувача. Розробка моделі бази даних. Діаграми варіантів*_____ *використання. Діаграма класів. Діаграма послідовності дій. Розробка алгоритму системи.* _ __*Розробка та тестування системи. Розробка серверної частини. Розробка функції обробки*_ __*замовлень. Реалізація функції оформлення замовлення. Реалізація функції захисту*_____ __*Реалізація функції захисту інформації. Розробка системи зберігання даних. Розробка*_____ __*інтерфейсу клієнтської частини. Тестування розробленого програмного забезпечення.*___
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____

5.1 Організаційна структура (1 аркуш формату А4). 5.2 Концептуальна діаграмаї (1 аркуш формату А4). 5.3 Діаграма декомпозиції (1 аркуш формату А4). 5.4 Декомпозиція функції «облік замовлення» (1 аркуш формату А4). 5.5 Декомпозиція функції «облік кухні» (1 аркуш формату А4). 5.6 Декомпозиція функції «облік меню» (1 аркуш формату А4). 5.7 Головна сторінка (1 аркуш формату А4). 5.8 Сторінка адміністрації (1 аркуш формату А4). 5.9 Сторінка офіціанта (1 аркуш формату А4). 5.10 Сторінка кухаря (1 аркуш формату А4). 5.11 Логічна модель бази даних (1 аркуш формату А4). 5.12 Діаграма варіантів використання системи адміністратором (1 аркуш формату А4). 5.13 Діаграма варіантів використання системи офіціантом (1 аркуш формату А4). 5.14 Діаграма варіантів використання системи кухарем (1 аркуш формату А4). 5.15 Діаграма класів системи (1 аркуш формату А4). 5.16 Алгоритм роботи інформаційної системи (1 аркуш формату А4)

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання, аналіз завдання, уточнення*	*23.03.2021*	
	плану роботи		
2	*Аналіз діяльності закладів харчування*	*26.03.2021*	
3	*Постановка задачі та вибір методу її*	*03.04.2021*	
	вирішення		
4	*Проведення експериментальних досліджень*	*18.04.2021*	
5	*Оформлення пояснювальної записки*	*27.04.2021*	
6	*Підготовка презентації*	*11.05.2021*	
7	*Подання закінченої роботи науковому*	*12.05.2021*	
	керівникові		
8	*Подання роботи на рецензування*	*20.05.2021*	
9	*Попередній захист*	*23.05.2021*	
10	*Подання роботи до комісії*	*24.05.2021*	

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Звіт: _64_ с., _20_ рис., _17_ джерел.

Об’єктом дослідження виступають методи та інструменти для створення системи інформаційного забезпечення обслуговування відвідувачів ресторану.

Метою роботи є системне проектування інформаційного забезпечення обслуговування відвідувачів мережі підприємств харчування. Розроблюваний продукт являє собою веб сервер і призначений для внутрішнього користування, а саме координації роботи співробітників ресторану.

Відповідно до мети проекту були поставлені наступні завдання:

- Перегляд меню, списку страв;
- Пошук страви за назвою, опису;
- Перегляд інформації по обраної страви;
- Оформлення замовлення;
- Перегляд оформлених замовлень і їх станів;
- Зміна статусу обраного замовлення;
- Авторизація (для співробітників ресторану).

Методи розробки включають аналізу літератури, аналізу нормативно-правової документації, аналізу системи, моделювання, абстрагування, конкретизація і ідеалізація, класифікація, узагальнення.

КЛЮЧОВІ СЛОВА: ПРЕДМЕТНА ОБЛАСТЬ, СПЕЦИФІКАЦІЯ, ФУНКЦІОНАЛЬНІ ВИМОГИ, СИСТЕМНІ ВИМОГИ, РОЗРОБКА, БАЗА ДАНИХ, ЛОГІЧНА МОДЕЛЬ, UML, IDEF0, АВТОМАТИЗАЦІЯ, РЕСТОРАН.

ABSTRACT

Report: _64_ pp., _20_ fig., _17_ sources.

The object of research are methods and tools for creating a system of information support for restaurant visitors.

The purpose of the work is the systematic design of information support services for visitors to the network of food companies. The developed product is a web server and is intended for internal use, namely the coordination of the restaurant staff.

In accordance with the purpose of the project, the following tasks were set:

— View the menu, list of dishes;
— Search for a dish by name, description;
— View information on the selected dish;
— Ordering;
— View orders and their status;
— Change the status of the selected order;
— Authorization (for restaurant employees).

Development methods include literature analysis, analysis of regulatory documentation, system analysis, modeling, abstraction, concretization and idealization, classification, generalizatio

KEYWORDS: SUBJECT AREA, SPECIFICATION, FUNCTIONAL REQUIREMENTS, SYSTEM REQUIREMENTS DEVELOPMENT, DATABASE, LOGICAL MODEL, UML, IDEF0, AUTOMATION, RESTAURANT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АІС - автоматизована інформаційна система;

БД - база даних;

СУБД - система управління базами даних;

CASE (Computer Aided Software Engineering) - сукупність методів і засобів автоматизованого проектування інформаційних систем;

COM (Component Object Model) - компонент, заснований на об'єктної моделі;

ERD (Entity Relationship Diagram) - модель даних, що дозволяє описувати концептуальні схеми предметної області;

HTML (HyperText Markup Language) - мова гіпертекстової розмітки;

HTTP (HyperText Transfer Prorocol) - протокол передачі гіпертекстових файлів;

JSP (Java Server Pages) - розширення технології сервлетів для спрощення роботи з web-вмістом;

JSTL (Java Standard Tag Library) - бібліотеки стандартних тегів на мові Java;

MVC (Model-View-Controller) - архітектурний шаблон підходу до створення об'єктно-орієнтованих web-додатків;

SQL (Structured Query Language) - мова структурованих запитів;

URL (Uniform Resource Locator) - уніфікований покажчик інформаційного ресурсу;

XML (eXtensible Markup Language) - розширена мова розмітки документів.

ВСТУП

Робота присвячена дослідженню проблем та перспективам застосування інформаційного забезпечення обслуговування відвідувачів мережі підприємств харчування. Актуальність роботи обумовлена реаліями сьогодення, які вимагають використання інновацій у виробничому процесі ресторанів для підвищення конкурентоспроможності бізнесу. Метою роботи є аналіз теоретичних та практичних аспектів автоматизації підприємств громадського харчування та, в першу чергу, автоматизації обслуговування клієнтів за рахунок впровадження систем інформаційного забезпечення на зазначених підприємствах.

Перші системи з автоматизації діяльності закладів харчування беруть свій початок з 1974 року, коли Вільям Бробек та його партнери побудували керовані мікропроцесором касові системи для ресторанів McDonald's. У цій системі натискання відповідних клавіш товару та цифрових клавіш робить замовлення для клієнта, а потім продовжували обчислювати рахунок, коли оператор натискає кнопку загальної суми та друкували чек. Після цього у 1978 році Джин Мошер винайшов першу графічну систему торгових точок (POS) із підтримкою сенсорного екрану.

Протягом десятиліть програмне забезпечення обслуговування відвідувачів мережі підприємств харчування розвивалася, щоб охопити більше функціональних аспектів. Деякі системи забезпечують повне охоплення допоміжних операцій, таких як: контроль запасів, управління відносинами з клієнтами, бронювання столиків та планування зміни персоналу. Однією з рушійних сил інновації такої системи є спроба замінити схильну до помилок монотонну паперову систему.

Зазвичай робочий процес закладу починався з того, що офіціанти збирали замовлення від замовника на аркуші замовлення, потім передавали це кухарям для приготування їжі і, нарешті, збирали плату від замовника. Цей процес може

спричинити певні ризики, однак, особливо в піковий період, вони не обмежуються втратою аркуша замовлення, але також неправильною послідовністю приготування їжі та додатковими витратами через помилкові замовлення. Згодом вони можуть призвести до низької продуктивності та незадоволення споживачів. Усвідомлення цих проблем може вплинути на ефективність бізнесу, тому власник ресторану швидко шукає засіб, застосовуючи інформаційні системи у своїй бізнес-моделі.

Зважаючи на складність системи, системне проектування інформаційного забезпечення обслуговування відвідувачів мережі підприємств харчування є одним із важливих завдань з автоматизації підприємства. При проектуванні моделі системи необхідно враховувати певні аспекти, кожен з яких надає певні обмеження до системи. Це можуть бути не тільки технічні показники, а й облік ринку та економічні міркування. Важливим критерієм для вибору є наявність надійних та ефективних засобів розробки. Інформаційне забезпечення дозволяють фахівцям на підприємстві самостійно та швидко налаштовувати інформаційну систему відповідно до вимог.

Цей проект передбачає розробку, побудову та тестування системи ресторанів, що спрямована на вирішення проблем ресторану за допомогою інформаційних технологій. З точки зору експлуатаційних переваг, це може підвищити ефективність процесу, зменшити можливі людські помилки та максимізувати використання ресурсів. Крім того, він також підтримує довгострокові бізнес-цілі, включаючи досягнення економічної ефективності, максимізацію прибутку та потенціал для виходу на більш широкі ринки.

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Заклади харчування є підприємствами де клієнтам подаються страви та напої. Заклади громадського харчування, залежно від типу, можуть відрізняється якістю послуг та наявним меню. Окрім харчування, підприємства також можуть надавати клієнтам послуги з розваги та відпочинку.

Заклади громадського харчування мають характерну підрозділену організаційну структуру. Концептуально в цьому типі структури організація бізнесу поділяється на компоненти, на які покладаються обов'язки на основі оперативних вимог. Кожен підрозділ опрацьовує конкретну операційну область або набір стратегічних цілей. Однією з цілей цієї корпоративної структури є підтримка автономії та організаційної гнучкості у задоволенні потреб бізнесу в різних організаційних аспектах та на ринках.

Організаційна структура закладу наведена на рис. 1.1.

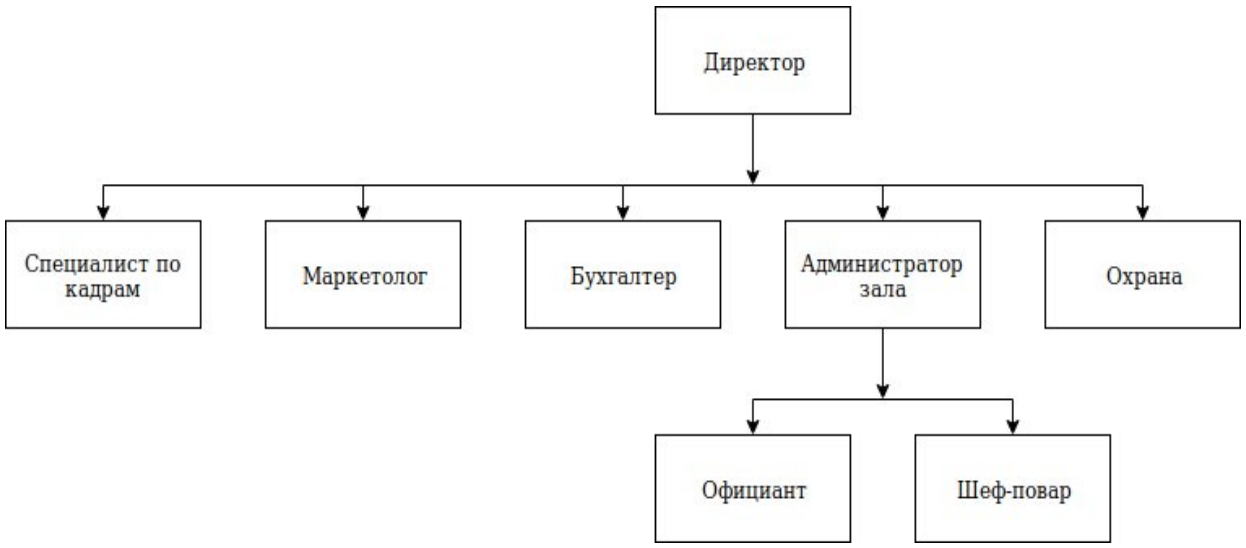

Рисунок 1.1 – Організаційна структура

Власник або основний акціонер контролює цю структуру. Цей чоловік має найвищий контроль над рестораном, як він працює, але часто не делегує завдання нікому напряму, робить це через бухгалтера та генерального менеджеру. Іноді і в залежності від закладу, власник делегує завдання помічникам управління та помічникам на кухні.

Бухгалтер має справу з великою кількістю різних сфер бізнесу, таких як бухгалтерські системи, що зберігають дані, складання фінансової звітності та юридичні міркування. Бухгалтер виконує одну з найважливіших ролей у будь-якому бізнесі, незалежно від того, працює він у великих корпораціях чи малому бізнесі. Як фінансовий кістяк бізнесу, вони готують та вивчають фінансові записи, забезпечують точність усіх грошових операцій та своєчасну сплату податків.

Офіціанти несуть відповідальність за прийняття замовлень та подачу їжі та напоїв гостям. Вони відіграють важливу роль у задоволенні гостей, оскільки вони також відповідають за перевірку споживачів, щоб вони насолоджувались їжею та вживали заходів для усунення будь-яких проблем. Посада містить такі обов'язки та відповідальність: надати чудове обслуговування клієнтів, завжди прагне до найкращого задоволення споживачів, привітає клієнтів та подаруйте меню, вносити пропозиції, виходячи зі своїх уподобань, приймає та подає замовлення на їжу та напої, підвищує продаж, коли це доречно, завжди тримайє столи в чистоті та порядку, перевіряє продукти на якість, доставляє чеки та збирає платежі, співпрацює та спілкується з кухонним персоналом, дотримується усіх відповідних правил / положень департаменту охорони здоров’я та всіх рекомендацій щодо обслуговування споживачів

Обов'язки шеф-кухаря включають вивчення рецептів, складання меню та приготування високоякісних страв. Шеф-кухар повинен мати можливість делегувати завдання працівникам кухні, щоб забезпечити своєчасне приготування їжі. Крім того, він повинні бути ознайомлені з санітарними правилами. Зрештою, вони підготують та доставлять повне меню, яке порадує гостей.

Менеджери ресторанів відповідають за нагляд, координацію, організацію та оцінку всіх операцій, пов'язаних з ефективним та безперебійним функціонуванням закладів харчування. Ці спеціалісти планують і керують виконанням найрізноманітніших завдань, включаючи нагляд за фінансовими аспектами ресторану, такими як доходи та витрати, управління доходами, а також відповідальність за графіки роботи працівників та оплату праці. Головною метою менеджера ресторану є забезпечення високої якості послуг та продуктів, що підвищує визнання закладу, достаток споживачів і, врешті-решт, прибуток.

Менеджери ресторанів мають право брати участь у реальному процесі приготування їжі; це завдання, як правило, покладається на оперативного персоналу. Однак у менших ресторанах лінія, яка розділяє функції менеджера ресторану та виконавчих шеф-кухарів, часто розмита, і обидві посади може займати одна людина. У цих випадках менеджер ресторану, як правило, має досвід підготовки їжі та управління харчовими послугами.

Професіонал з кадрів веде кадровий документообіг. Це підрозділ компанії, яке відповідає за пошук, перевірку, набір та навчання кандидатів на роботу, а також для адміністрування програм нагородження співробітників. Додаткові зобов'язання щодо людських ресурсів включають компенсацію та льготи, наїм, звільнення та сполучення будь-яких законопроектів, які можуть пов'язувати компанію та її співробітників.

Рекламодавець є суб’єктом господарювання з боку покупців екосистеми маркетингу. У контексті реклами, рекламодавець часто є торговою маркою, яка прагне поширити конкретне повідомлення про свій товар (наприклад, кампанії із залучення нових користувачів та перенацілювання). Рекламодавці купують рекламний простір у видавців та рекламних мереж, які допомагають їм передавати своє повідомлення користувачам, яким буде цікаво почути це повідомлення.

При встановленні політики та процедур керівники відділів безпеки повинні звертатися до доступних ресурсів, що надаються професійними групами. Відділ безпеки забезпечує безпеку приміщенню, персоналу, обладнанню для

моніторингу спостереження; робить огляд будівель, обладнання та точок доступу; дозволяють в'їзд; отримують допомогу, подаючи сигнал тривоги. Запобігають втратам та збиткам, повідомляючи про порушення.

1.1 Аналіз нормативно правових документів

Провідними правовими документами у відносинах меж підприємствами вважаються договори, що представляють собою домовленості сторін про встановлення відносин та врегулювання конфліктів. Заклади харчування здійснюють свою діяльність на основі законів держави знаходження. Нормативні документи, на підставі яких працюють заклади харчування в Україні:

– Закон України «Про забезпечення санітарного та епідемічного благополуччя населення» від 24.02.94 р. № 4004-XII.

– Закон України «Про дозвільну систему у сфері господарської діяльності» від 06.09.2005 р. № 2806-IV.

– Закон України «Про пожежну безпеку» від 17.12.93 р. № 3745-XII.

– Закон України «Про планування і забудову територій» від 20.04.2000 р. № 1699-III.

– Закон України «Про підтвердження відповідності» від 17.05.2001 р. № 2406-III.

– Закон України «Про податок на додану вартість» від 03.04.97 р. № 168/97-ВР.

– Закон України «Про ліцензування певних видів господарської діяльності» від 01.06.2000 р. № 1775-III.

– Закон України «Про державне регулювання виробництва і обороту спирту етилового, коньячного та плодового, алкогольних напоїв та тютюнових виробів» від 19.12.95 р. № 481/95-ВР.

– Закон України «Про патентування деяких видів підприємницької діяльності» від 23.03.96 р. № 98/96-ВР.

– Закон України «Про застосування реєстраторів розрахункових операцій у сфері торгівлі, громадського харчування та послуг» від 01.06.2000 р. № 1776-III.

– Декрет КМУ «Про місцеві податки та збори» від 20.05.93 р. № 56-93.

– Порядок видачі органами державного пожежного нагляду дозволу на початок роботи підприємств та оренду приміщень затверджений постановою КМУ від 14.02.2001 р. № 150.

– «Правила роботи закладів (підприємств) ресторанного господарства, затверджені наказом Мінекономіки та з питань європейської інтеграції України від 24.07.2002 р. № 219».

– Порядок видачі Свідоцтва про сплату єдиного податку, затверджений наказом ДПАУ від 29.10.99 р. № 599.

– Порядок проведення обов'язкових профілактичних медичних оглядів та видачі особистих медичних книжок, затверджений постановою КМУ від 23.05.2001 р. № 559.

– Порядок проведення торговельної діяльності та правила торговельного обслуговування населення, затверджений постановою КМУ від 15.06.2006 р. № 833.

– Правила роздрібної торгівлі продовольчими товарами, затверджені наказом Міністерства економіки від 11.07.2003 р. № 185.

– Правила роздрібної торгівлі алкогольними напоями, затверджені постановою КМУ від 30.07.96 р. № 854.

– «Тимчасовий порядок видачі ліцензій на право імпорту, експорту спирту етилового, коньячного та плодового, алкогольних напоїв та тютюнових виробів і роздрібної торгівлі алкогольними напоями та тютюновими виробами, затверджений постановою КМУ від 13.05.96 р. № 493».

– Перелік окремих форм та умов проведення діяльності у сфері торгівлі, громадського харчування і послуг, яким дозволено проводити розрахункові операції без застосування реєстраторів розрахункових операцій з використанням

розрахункових книжок та книг обліку розрахункових операцій, затверджений постановою КМУ від 23.08.2000 р. № 1336.

1.2 Аналіз діяльності з обслуговування клієнтів

Невдоволені клієнти ресторану скаржаться на погане обслуговування. Незалежно від концепції чи бізнес-моделі, заклад отримує належну частку скарг клієнтів ресторану. Гості можуть попросити побачити менеджера, кухаря, зателефонувати для скарги або просто залишити негативний відгук в інтернеті. Однак велика кількість людей взагалі нічого не говорить. Незадоволені гості не даватимуть усних рекомендацій або рекомендацій у соціальних мережах. Вони перестануть заходити, і власник ніколи не дізнаєтесь причин. Можна вжити заходів для запобігання та вирішення поширених скарг клієнтів ресторану. Дослідіть найпопулярніші скарги гостей і дізнайтеся, як орієнтуватися в хитрому світі обслуговування клієнтів.

Закладу потрібно заповнити свої місця, збільшити продажі та тримати свій бренд на висоті. Більшість людей навіть не скаржаться. Вони не позначатимуть ваш бренд та не дадуть можливості відповісти.

Брудний посуд, холодна їжа та неправильні замовлення створюють незадоволених гостей. Проте власники ресторанів стикаються з проблемами цифрового досвіду, службами доставки сторонніх виробників та проблемами пандемії. Деякі приклади скарг клієнтів у ресторанах включають: відсутність чистоти у ванних кімнатах, посуді, їдальні або службовців; неправильна температура їжі або їжа, яка не виглядає або не смакує, як описано у вашому меню, на веб-сайті чи в соціальних мережах; почуття поспіху закінчити їжу, часто посилюється сервером, який забирає тарілку гостя до закінчення; неввічливий персонал ресторану вдома або по телефону; неточні внутрішні замовлення, замовлення на виконання чи доставка; повільне обслуговування або довгий час очікування доставки; політика, пов’язана з пандемією, наприклад вимоги до маски.

Стандарти співробітників, побічна робота та нагляд керівника є частинним вирішенням цієї проблеми. Наприклад, у вашому довіднику слід обговорювати гігієну працівників, правила волосся та політику безпеки щодо миття рук та носіння масок. Доручіть керівництву контролювати та брати участь у регулярних допоміжних роботах, таких як чистка плінтусів, вікон та дверей. Гості беруть до уваги, коли члени екіпажу роблять невелике додаткове прибирання під час повільних періодів, а не грають на своїх телефонах.

Зараз холодна їжа ніколи не є прийнятною, але це трапляється. Інші проблеми виникають внаслідок неналежних зусиль для покриття та презентації. Рішення для холодної їжі порівняно просто. Однак менеджер повинен переглянути, чому в першу чергу подавали холодну їжу. Зазвичай це може бути неправильно працюючи мікрохвильові печі, термометри та нагрівальні лампи. Інколи не кожен співробітник виконував свою роботу точно, починаючи від замовлення.

Хоча надзвичайно груба взаємодія може спричинити скаргу в інтернеті, люди висловлюють своє незадоволення іншими способами. Власникові потрібно уникати поганого сервісу, наймаючи потрібних людей, навчаючи нових членів команди та обирати наставників для кращих навичок. Управлінська команда повинна вловити та вирішити проблему поганої поведінки або етикету, перш ніж це переросте в повномасштабну скаргу.

Звичайно, якщо клієнт чекає їжу незвично довго, це призводить до поганого досвіду. Однак у вас, мабуть, були гості, які скаржились, лише щоб поспішити і побачити, що минуло лише вісім хвилин. Річ у тім, що час рухається по-різному залежно від гостя, його настрою чи обставин, пов’язаних із трапезою.

Частково зазначені проблеми можна усунути. Забезпечити відмінне обслуговування клієнтів ресторану за допомогою впровадження технологій які будуть залежати від типу ресторану, але деякі форми технологій можуть бути впроваджені в багато бізнес-моделі ресторанного бізнесу. Можливість миттєвого замовлення з меню забезпечує легкий доступ для офіціантів. Це дозволяє їм зручно переглядати, а потім робити замовлення з меню. Ваші звичайні клієнти

пообідають швидко і легко знайдуть цю систему замовлення. Вони також оцінять зручність і швидкість.

1.4 Визначення сфери застосування системи

Зацікавлені сторони мають першочергове значення для будь-якого проекту завдяки величезному проектному ресурсу, який було вкладено, щоб точно знати, чого хоче користувач. Якщо до зацікавлених сторін звертаються раніше в рамках проекту, простіше повідомити їх вимоги та вирішити їхні пріоритетні проблеми. Початковим кроком для виявлення вимог зацікавлених сторін буде аналіз зацікавлених сторін. Аналіз зацікавлених сторін розглядає систему як «складний набір взаємодіючих елементів, які працюють разом для задоволення потреб або цілей». Ідея полягає в тому, щоб з’ясувати, як, коли і де зацікавлені сторони беруть участь у процесі.

Процес аналізу починається з виявлення першої групи людей, які складають частину системи, а саме офіціантів, касирів, господарів та шеф-кухарів, оскільки “підвищення ефективності управління знаннями на підприємствах можливо, якщо враховувати особливості людини по роботі з інформацією і знаннями, прибрати плутанину з основними поняттями, визначити дійсний об'єкт управління і побачити коріння всього того, що впливає на результат діяльності”. Ця група людей бере на себе конкретні завдання для досягнення своїх оперативних цілей, і система не може працювати без них. Після цього аналіз зацікавлених сторін продовжує виявляти наступну групу зацікавлених сторін, як правило, бенефіціарів першої групи. Керівник може розглядатися як безпосередній функціональний бенефіціар, який покладається на персонал для управління рестораном. Крім того, менеджеру потрібно придбати інгредієнти для ресторану у відповідний час, суми та ціну; отже, існує необхідність встановити довгострокові відносини з постачальником для стабільного постачання матеріалів. Нарешті, клієнт, який оплачує послуги, що надаються рестораном, є ще одним функціональним бенефіціаром системи.

Насправді єдиний зацікавлений учасник може виконувати комбіновані ролі інших зацікавлених сторін. Наприклад, менеджер іноді бере на себе роль касира та господаря у певному ресторані.

По суті, автоматизація ресторанів робить речі більш ефективними - якщо все зробити правильно, це може одночасно зменшити витрати ресторану, одночасно підвищуючи ефективність. І хоча деякі з цих технологій мають велику ціну, нові розробки та вдосконалення старих роблять ці системи більш доступними та доступними.

Ось чому ми можемо розраховувати, що все більше ресторанів продовжуватимуть рухатись на шляху автоматизації, оскільки галузь продовжує дозрівати та спеціалізуватися. Існує безліч привілеїв для ресторанів, які можуть знайти правильний баланс людської праці та автоматизації.

Автоматизація може пришвидшити процес обіду приблизно на шість-дев'ять хвилин, оскільки кухарям не потрібно чекати, щоб прийняти замовлення або виконати інші запити. Швидший досвід, природно, призводить до збільшення кількості клієнтів на день, що означає більший дохід для ресторанів. Коли програмне забезпечення поєднує підвищену ефективність та доходи з меншою заробітною платою персоналу та меншими помилками, спричиненими цими працівниками, у керівника є чіткий рецепт стійкої, прибуткової бізнес-моделі. Однак ця ключова перевага є однією з найбільших суперечок. У всьому світі працівники стурбовані тим, що їх замінять роботи, а їхні навички застаріють. Далі цей вплив на робочу силу створює очікування щодо її потенційного впливу на місцеву та глобальну економіки.

Хоча неможливо передбачити результат, певно, що робоча сила буде розвиватися у відповідь на цю нову технологію.

1.5 Постановка задачі

Загальною метою цього проекту є системне проектування, розробка та експериментальна перевірка інформаційної системи забезпечення закладів

харчування. Проект також досліджує практичні методи проектування програмного забезпечення з метою побудови високоякісної системи. Крім того, він також досліджує можливість побудови інформаційної системи в режимі реального часу для забезпечення ефективного спілкування. Нарешті, проект також досліджує та аналізує проблеми побудови ієрархії та адаптивного веб-дизайну, щоб визначити, змоделювати та розробити ергономічну поведінку графічного інтерфейсу.

Система вимагає архітектурного дизайну, що підтримує відокремлення інтерфейсів та реалізацію, щоб система була багаторазовою. Це повинно дозволяти всім функціям бути доступними для цілого ряду пристроїв. Отже, він повинен мати рівень презентації, щоб полегшити відповідний інтерфейс користувача, рівень бізнес-логіки, що містить загальні бізнес-функції, і рівень даних для управління транзакціями баз даних. Для реалізації цього впровадження будуть потрібні поглиблений аналіз та хороша практика проектування.

Обмін інформацією має відбуватися миттєво, оскільки затримка з виконанням замовлення, безсумнівно, зменшує задоволеність клієнтів. Проект потребуватиме суттєвого розслідування того, як спілкування в реальному часі має відбуватися, щоб забезпечити ефективну співпрацю між персоналом. Крім того, передача даних для міжшарового зв'язку може також вплинути на продуктивність системи і повинна вирішуватися оптимізованим рішенням.

У будь-якій системі користувачам потрібно буде виконати кілька завдань для досягнення цілі високого рівня. Інтерфейс користувача повинен вести користувача через завдання та допомагати йому досягти кінцевої мети. Оскільки операції в ресторані передбачають численні завдання, слід провести детальний аналіз проектування простого та інтуїтивно зрозумілого інтерфейсу, що ефективно відповідає цілям користувачів (функціональним вимогам).

До системи повинен мати легкий доступ різного типу пристроїв; щоб він був портативним та багаторазовим. Враховуючи, що кожен мобільний пристрій може мати різну роздільну здатність та розмір екрана, інтерфейс користувача

системи повинен забезпечувати реагуючи механізм, щоб компенсувати ці обмеження за допомогою динамічного розміщення інтерфейсу та зміни розміру вмісту.

Необхідно провести значну кількість тестувань, щоб забезпечити відсутність помилок та помилок у системі-прототипі. Крім того, слід оцінити ефективність прототипу, щоб проаналізувати ефективність запропонованої методології.

2 ПРОЕКТУВАННЯ СИСТЕМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Системне проектування

Проектування систем — це процес визначення елементів системи, таких як модулі, архітектура, компоненти та їх інтерфейси, та дані для системи на основі визначених вимог.

Для узгодженої та добре функціонуючої системи необхідний системний підхід, “Ми розглядаємо проблему складності з двох взаємопов'язаних точок зору: наскільки точно ми можемо описати інформаційний об'єкт за допомогою доступних і зрозумілих об'єктів; як спрощені отримані описи шляхом декомпозиції наявних предикатів.”. Для врахування всіх пов'язаних змінних системи необхідний підхід знизу вгору або зверху вниз. Розробник використовує мови моделювання для вираження інформації та знань у структурі системи, яка визначається послідовним набором правил та визначень. Дизайн може бути визначений графічними або текстовими моделями моделювання.

Системне проектування комплексно вирішує поставлені завдання, бере до уваги взаємодію і взаємозв'язок окремих об'єктів-систем і їх частин, як між собою, так і з зовнішнім середовищем, враховує соціальні та економічні наслідки їх функціонування. Системне проектування ґрунтується на ретельному спільному розгляді об'єкта проектування і процесу проектування, який в свою чергу включають ще ряд важливих частин.

Для проектування систем використовують уніфіковану мова моделювання (UML). UML використовується для опису програмного забезпечення як структурно, так і за поведінкою з графічними позначеннями. Блок-схеми схематично та поетапно представляють алгоритм.

Архітектурне проектування описує погляди, моделі, поведінку та структуру системи. Логічне проектування використовуються для подання потоку даних, входів та виходів системи. Приклад: Діаграми ER (Діаграми

взаємозв’язків).Фізичний дизайн визначає як користувачі додають інформацію до системи та як система представляє інформацію назад до користувача, як дані моделюються та зберігаються в системі, як дані рухаються по системі, перевіряються, захищаються та трансформуються, коли вони протікають через систему та з неї.

2.2 Специфікація вимог до системи

Вимоги системи можуть бути визначені як опис того, які послуги вона може надати, та обмеження її функціонування. Ці вимоги безпосередньо відповідають потребам користувачів у термінах використання системи для досягнення їх ділової діяльності або процесу.

Процес розуміння вимог включає широкий спектр процесів, завдань і методів збору, аналізу, документування та перевірки вимог. Результат таких процесів може служити вхідним матеріалом для моделювання проектування систем. Він починається зі збору вимог користувача, потім проводить аналіз зібраних вимог і, нарешті, документує їх у відповідному форматі.

Виявлення проблем що підлягають вирішенню, що користувач (або зацікавлені сторони) намагаються досягти за допомогою системи та як система задовольняє бізнес-потреби є важливим етапом в проектуванні системи. Процес починається з розуміння та аналізу проблем ресторанного бізнесу. Бізнес-аналіз часто показує, що люди в організації мають різні потреби (або думки) та погляди, пов'язані із загальними вимогами системи. Вони є зацікавленими сторонами, які або безпосередньо взаємодіють із системними вимогами, або на них побічно впливають. Отже, фокус пошуку вимог у полягає в аналізі ролі зацікавлених сторін і як операції, які вони виконують впливає на систему; і, отже, має бути зазначена у вимогах.

Специфікація вимог має на меті визначити вимоги чіткою та однозначною мовою на основі вимоги, визначеної під час формування та аналізу списку проблем. Вимоги змінюються з часом і стають точніше відображати потреби

зацікавлених сторін. Наступними кроками є задокументування цих вимог та розгляд їх як остаточного посилання на знання домену та бізнес-правил. Вимоги щодо документування можуть бути складними, і це узгоджується з тим, що індустрія програмного забезпечення, як видається, використовує термін "вимога" непослідовно. Вимогою може бути абстрактний опис послуг, що надаються системою, або вичерпне формальне визначення функціональних одиниць системи.

З огляду на величезну різницю поглядів та інтересів зацікавлених сторін щодо найкращого способу визначення вимог, необхідне подальше дослідження підходів до документування вимог.

Відповідно до цілей створення програмного забезпечення, були встановленні наступні системні вимоги:

а) реалізувати серверну частину у якості сервлетів;

б) реалізувати клієнтську частину у якості web сайту;

в) реалізувати інтерфейс користувача у вигляді вебсторінок для глобальної мережі інтернету;

г) використовувати реляційні бази даних для збереження інформації;

г) стабільна та надійна робота програмного забезпечення;

д) збереження можливості для доопрацювання.

2.3 Визначення функціональних вимог

Функціональні вимоги описують функції, які повинна забезпечувати система. У них часто описуються очікувані функції, які користувач може використовувати для виконання свого завдання. Він може охоплювати певні бізнес-процеси та процедури, яких програмне забезпечення повинно виконувати для досягнення цілей користувача. Таким чином, функціональні вимоги є очікуваними характеристиками поведінки програмного забезпечення для отримання очікуваного результату. Основними завданнями є підтримка або

допомога ресторанним працівникам у процесі або управлінні робочим процесом замовлення та приготування їжі.

Моделюючи діяльність закладу харчування, ми можемо знайти як вхідні інформацію так і вихідну, так само нам варто ще враховувати й фактори, що теж впливають на діяльність закладу. Такими факторами є законодавство, правила та порядок приготування страв, технічні можливості.

Для визначення функціональних вимог до програмного забеспечення побудована її функціональна модель з використанням стандарту IDEF0.

На рис. 2.1 представлена концептуальна діаграма, яка надає погляд на функції інформаційного забезпечення з боку керівника закладу.

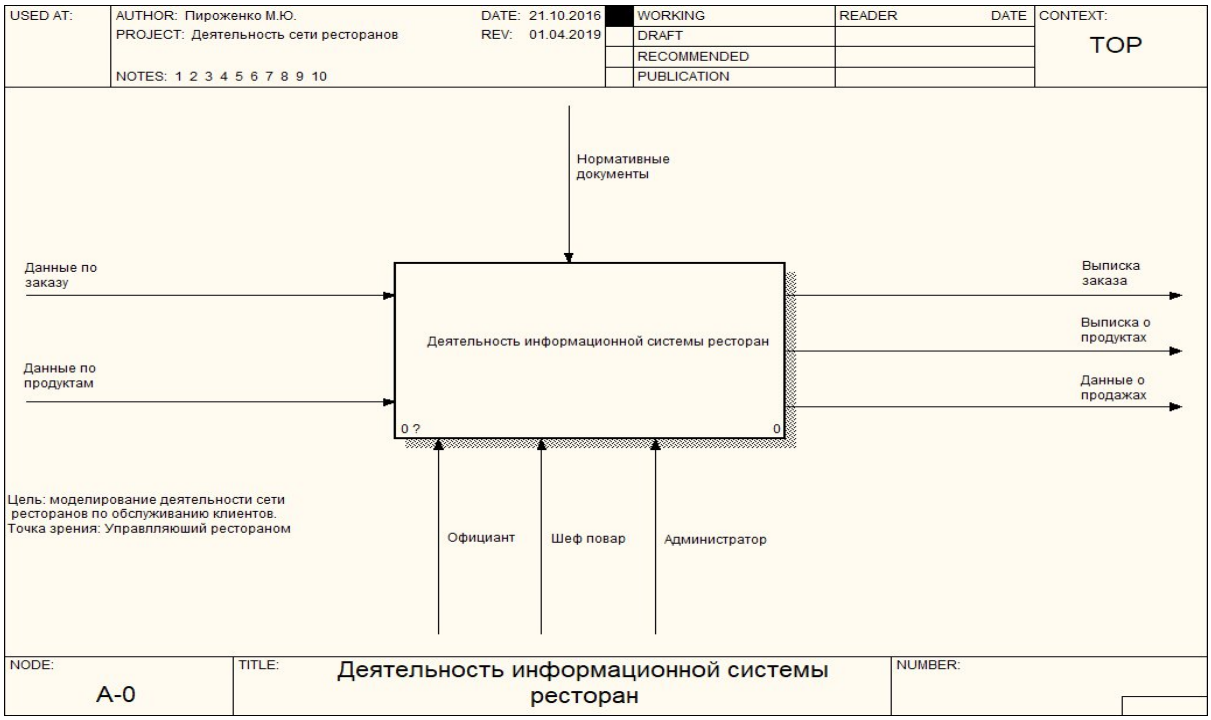

Рисунок 2.1 – Концептуальна діаграма

Ця діаграма може бути деталізована для уточнення вимог. На рис. 2.2 представлена декомпозиція концептуальної діаграми.

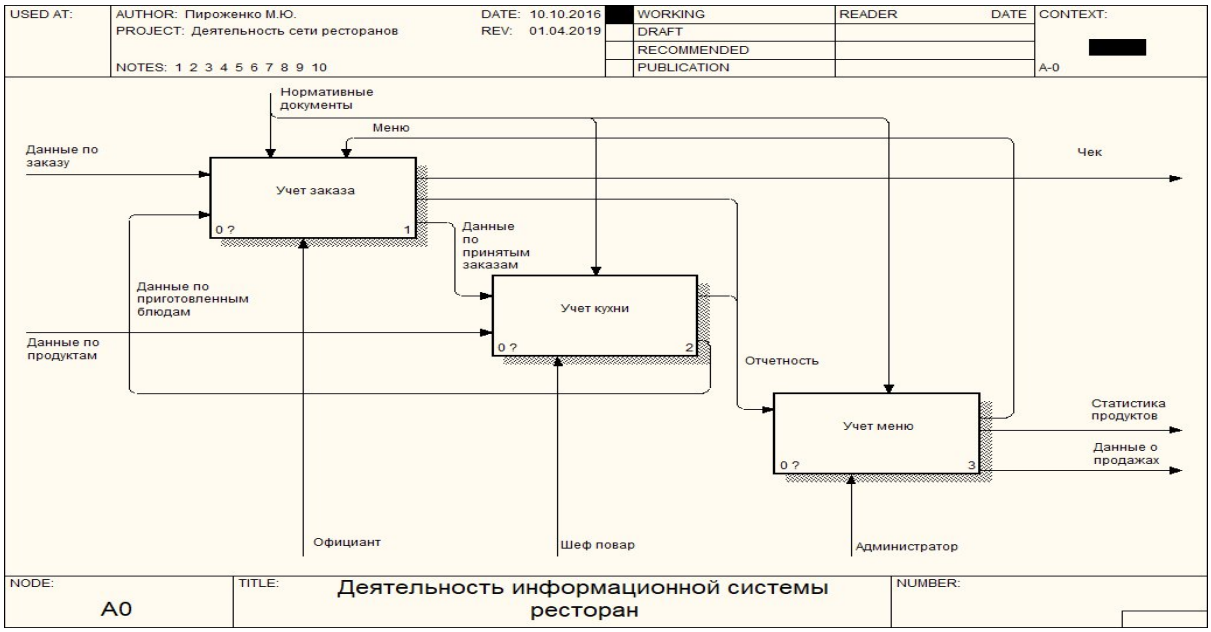

Рисунок 2.2 – Діаграма декомпозиції

На рис. 2.3 представлена декомпозиція функції обліку замовлення.

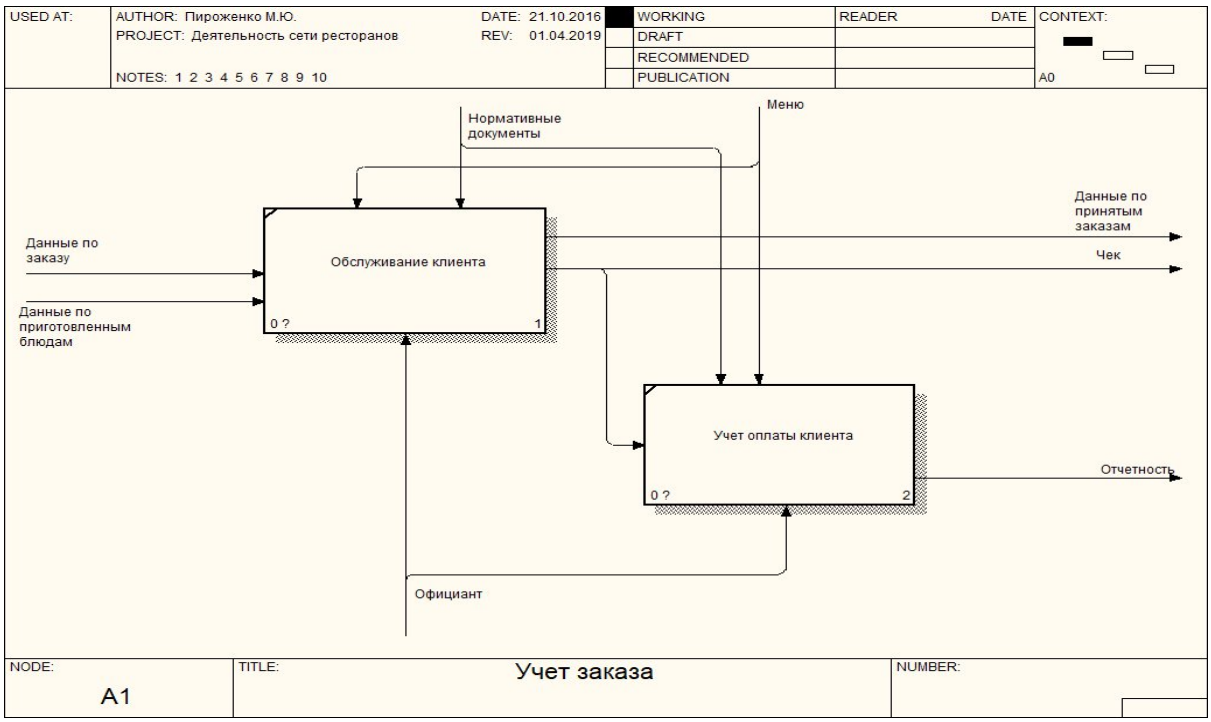

Рисунок 2.3 – Декомпозиція функції «облік замовлення»

На рис. 2.4 представлена декомпозиція функції обліку кухні.

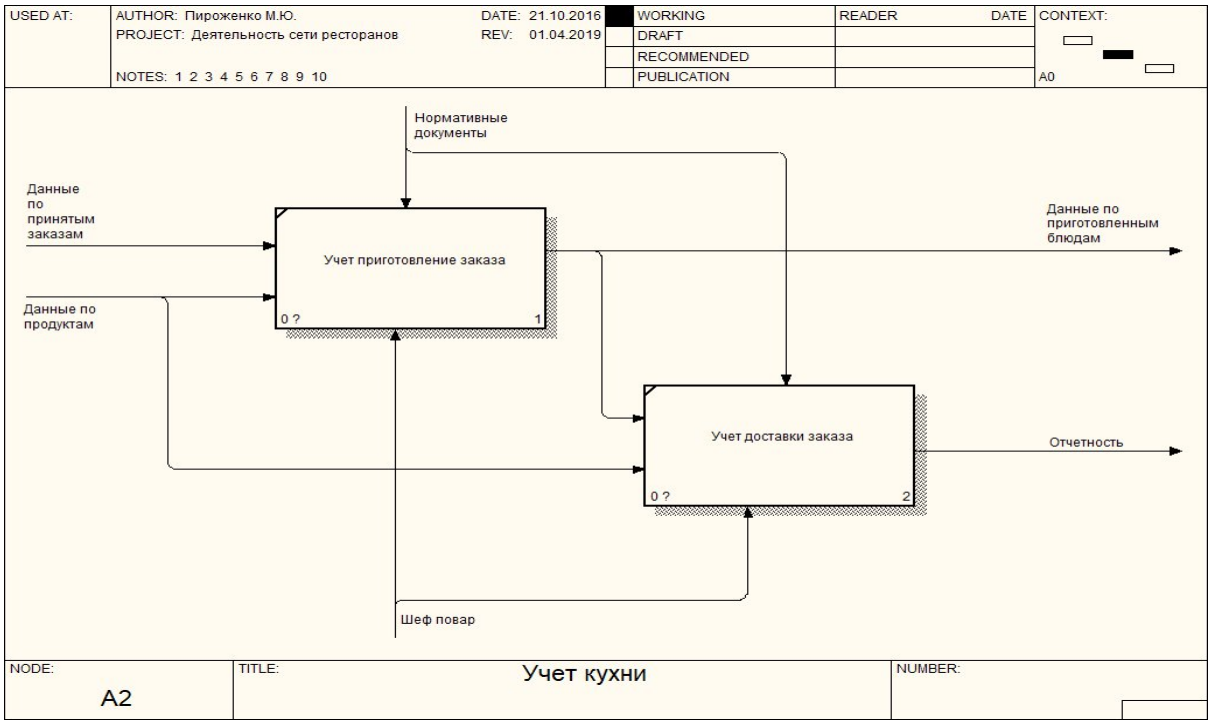

Рисунок 2.4 – Декомпозиція функції «облік кухні»

На рис. 2.5 представлена декомпозиція функції обліку меню.

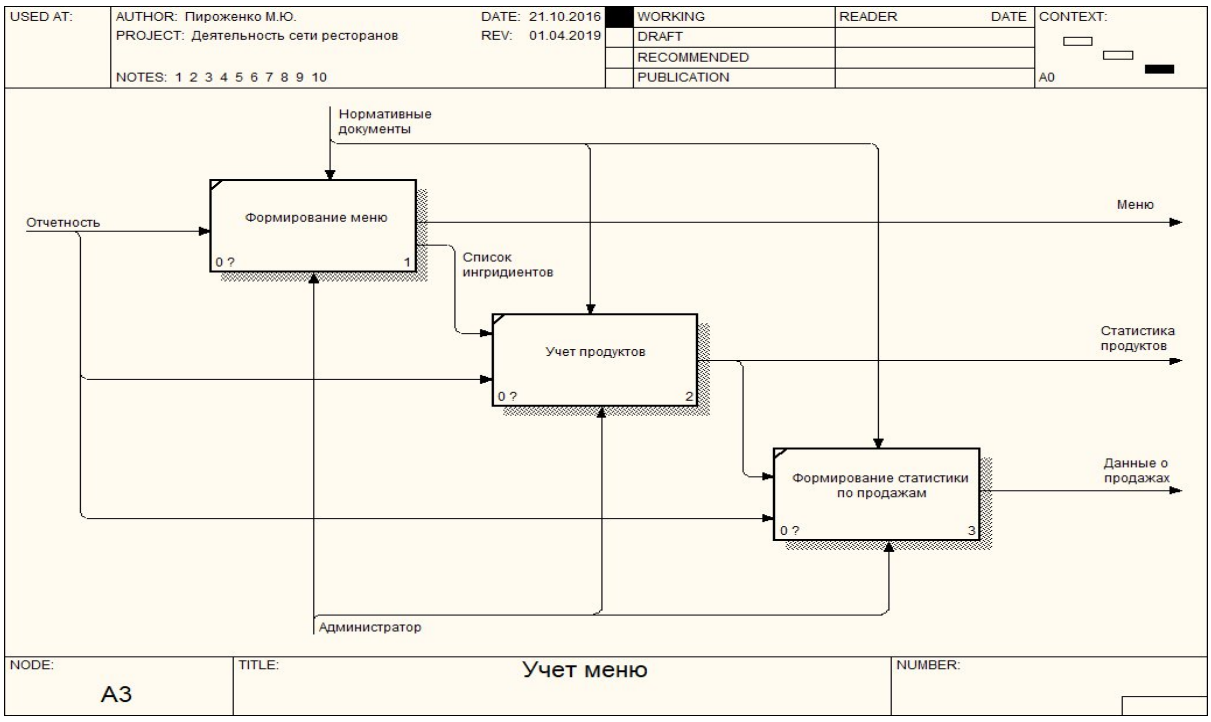

Рисунок 2.5 – Декомпозиція функції «облік меню»

2.4 Вимоги до інтерфейсу користувача

Коли користувачі взаємодіють із системою, вони намагаються виконати дії, які можуть відповідати їхнім цілям або потребам на високому рівні. Ці дії часто можна розбити на низку невеликих кроків. Інтерфейс користувача будь-якої системи є засобом, що допомагає користувачам пройти ці кроки та забезпечити комфортний досвід під час цього процесу. Це розуміння призведе до створення дизайну з відповідними елементами інтерфейсу та поведінкою, яка відповідає завданням користувача.

Як веб-програма, система в основному взаємодіє зі своїм користувачем через веб-браузер. Веб-браузери існують майже в усіх обчислювальних пристроях, включаючи мобільні та смарт-телевізори. Отже, це означає, що система має потенційний доступ через будь-який тип пристрою. Можна стверджувати, що ми можемо розробити інтерфейси, націлені на кожен тип пристрою. Однак ролі користувача можуть збігатися, і явно змусити їх перейти на інші пристрої для виконання ролей може бути проблематично. Це також накладає обмеження на носій, який буде використовуватися для доступу до системи, що призводить до додаткових витрат на придбання потрібних пристроїв для інтерфейсу користувача. Тому користувальницький інтерфейс повинен адаптувати роздільну здатність екрана різних пристроїв.

Інформаційна система має простий та зрозумілий користувацький інтерфейс. Це досягається завдяки використанню стандартного оформлення веб-сторінок, що істотно полегшує використання системи. Якщо співробітники закладу харчування вже працювали з подібними системами, їм буде інтуїтивно зрозуміло навігація, розташування кнопок та пунктів меню, вони будуть знати, як все це працює в системі.

На рисунках 2.6-2.9 представлено інтерфейс користувача.

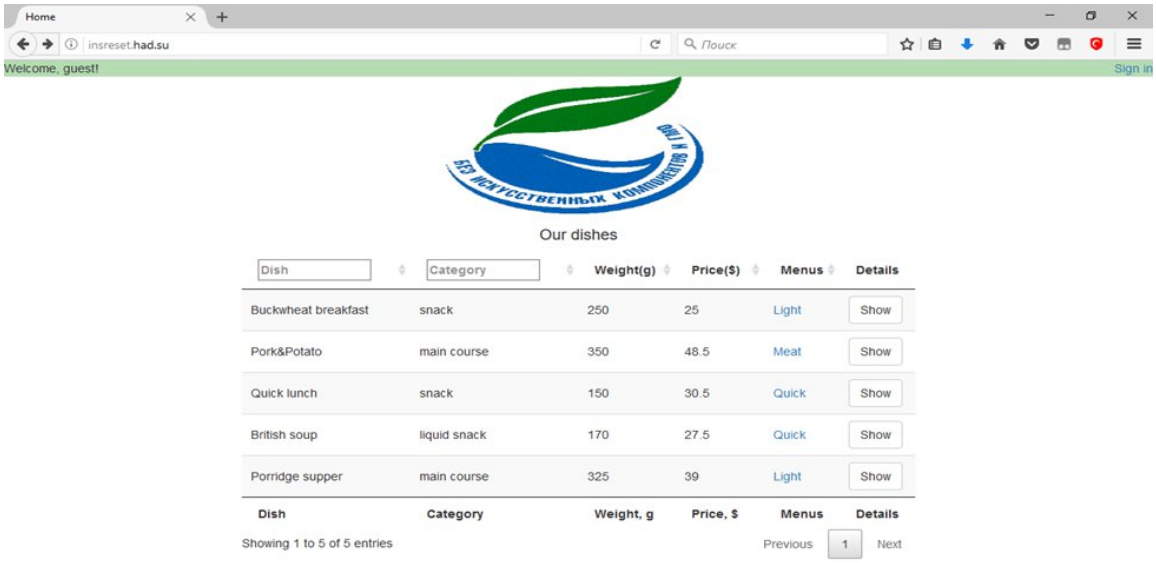

Рисунок 2.6 – Головна сторінка

Для перегляду доступних відомості про замовлення, в програмному забезпеченні є головна сторінка, та сторінки авторизованих користувачів: офіціантів, кухарів, адміністраторів.

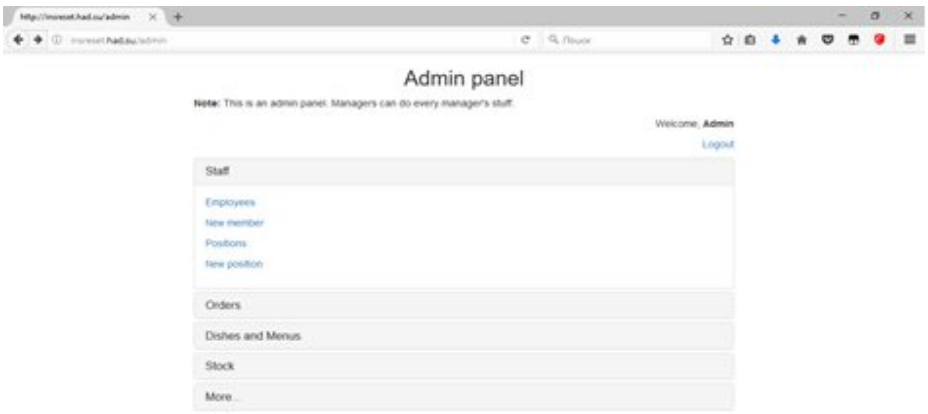

Рисунок 2.7 – Сторінка адміністрації

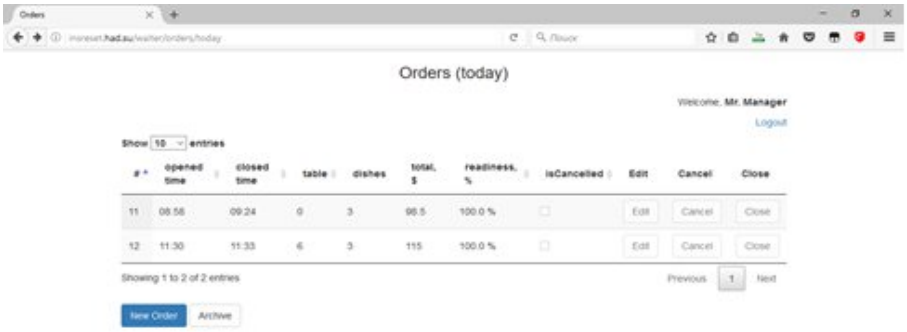

Рисунок 2.8 – Сторінка офіціанта

Офіціант може додати нове замовлення до бази даних. Кожна страва має характеристики, що вказані на сайті. Адміністратор може редагувати характеристики. Також є можливість додати товар до кошика, додавши нове блюдо. Після того, як офіціант обрав необхідні страви, він можете переглянути зроблене замовлення у кошику.

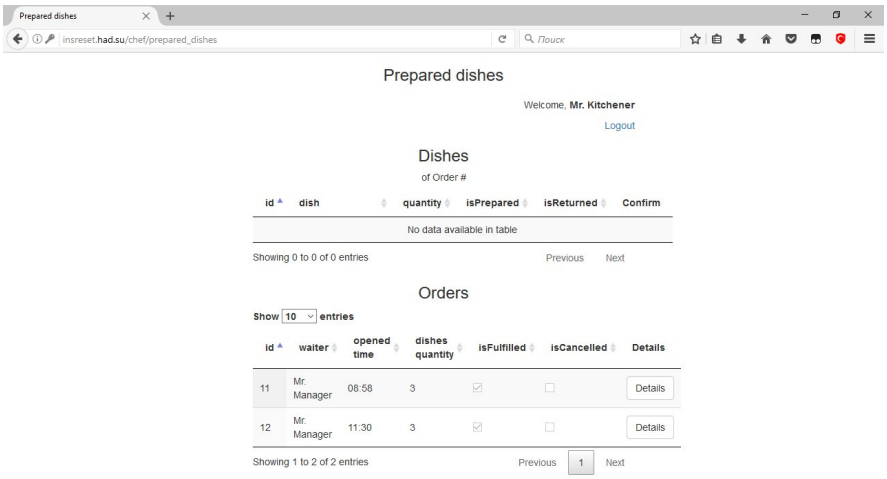

Рисунок 2.9 – Сторінка кухаря

Для підтвердження зробленого клієнтом замовлення, офіціанту потрібно заповнити певні дані, а саме номера столика на який офіціанту потрібно буде доставити замовлення та списку страв. Офіціант може додати нове замовлення. Кожна страва має характеристики, що вказана на сайті. Також є можливість додати товар до кошика, додавши нове блюдо. Після того, як офіціант обрав необхідні страви, він можете переглянути зроблене замовлення у кошику.

Програмне забезпечення вимагає затвердити всі дані, що були введеними користувачем під час заповнення запитів, а саме перевірка формату пошти, номера контактного телефону, а також інших даних, що вводяться користувачами.

Програмне забезпечення робить обробку всіх помилок задля коректної роботи. Це досягається завдяки своєчасному повідомлення користувача про можливі помилки, що були зафіксовані під час роботи. Тому користувач буде знати, якщо в процесі виконання роботи (запиту) щось пішло не так.

Для запобігання можливого виникнення проблем під час роботи у користувача з використанням програмного забезпечення, у користувача повинна бути детальна документація, а також користувачу потрібно надати можливість зв'язатися з технічною підтримкою за допомогою телефону, пошти, або інших форм контакту.

2.5 Розробка моделі бази даних

Модель даних – це засіб для визначення логічного зображення фізичних даних, що відносяться до деякого додатку. В процесі розробки засад створення баз даних вибір моделі в основному залежить від об'єкту впровадження (від регіонального представництва до центрального органу управління), а також від етапу впровадження.

Модель даних займається вивченням орієнтованих на дані структур. Він має на меті визначити структуру об'єктів даних, їх взаємозв’язки та пов’язану

інформацію, що описують об’єкти та взаємозв’язки. Модель даних важлива, оскільки кожному додатку в певний момент потрібно буде зберігати свої дані до певного типу сховища. Якщо структура даних не буде ретельно розроблена, це вплине на ефективність пошуку даних. Розробка моделі даних тісно пов'язана з рішеннями структур даних. Перш ніж цей процес може навіть розпочатися, тип даних та стратегію зберігання потрібно дослідити та ретельно підібрати, щоб супроводжувати обробку даних у системі.

Діаграма взаємовідносин широко застосовувана модель даних для представлення реляційної моделі даних бази даних. Є три основні концепції в ER діаграмах. По-перше, кожен об'єкт даних в ER діаграмах є іменованим об'єктом, який часто відображається у таблиці в СУБД. По-друге, інформація з суттю представлена набором атрибутів - які фіксують сегмент даних (наприклад, заголовок рецепта) об'єкта даних (наприклад, рецепт). Нарешті, асоціація між суб'єктами господарювання, також відома як відносини, зображує ділові правила системи високого рівня.

AllFusion ERwin Data Modeler r7 пропонує повний пакет для розробки баз даних з можливостями та функціями, які переміщують його до керівника класу. На ринку існує безліч чудових інструментів моделювання баз даних, включаючи ER / Studio Embarcadero Technologies, DBDesigner fabForce (з відкритим кодом), QDesigner Quest Software, PowerDesigner Sybase та програмне забезпечення Microsoft Visio. IT-менеджери вимагають, щоб такі продукти стали невід'ємною частиною всього процесу розробки баз даних, і AllFusion ERwin DM із повністю переробленим механізмом моделювання даних забезпечує вимоги.

AllFusion ERwin Data Modeler дозволяє розробникам баз даних негайно стати продуктивними, забезпечуючи при цьому середовище для збереження бізнес-правил, даних, метаданих, конструкцій баз даних, об’єктів баз даних та стандартів магазину. CA створила цей випуск на своєму переробленому механізмі узагальненого моделювання даних (GDM) із новими функціями, такими як "Повне порівняння" та вдосконаленими можливостями скасування / повторення. Функція порівняння дозволяє дизайнерам баз даних порівнювати

будь-які дві версії моделі або отриманої бази даних; функція скасування / повторення дозволяє користувачеві вносити зміни в будь-яку точку моделі, щоб зрозуміти вплив цих змін у базі даних.

Логічна модель бази даних представлена на рис. 2.10.

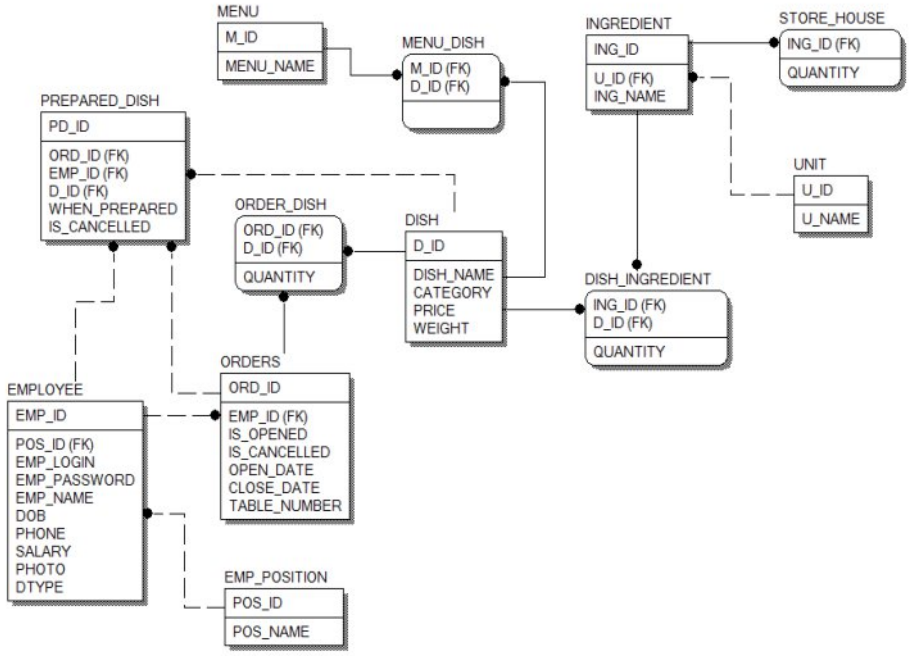

Рисунок 2.10 – Логічна модель бази даних

Зберігання даних - це контейнерний ресурс для об’єктів даних. Структури даних, які зберігатимуться в системі, потрібно зберігати одним або кількома методами зберігання даних.

Методом зберігання, вибраним для постійних файлів є СУБД. Основним обґрунтуванням був суб'єкт господарювання, первинна структура даних, що вимагала широких перехресних посилань. Наприклад, замовлення повинно знати, які рецепти були обрані, а рецепти повинні знати, які матеріали потрібно споживати. Введення цих даних в ієрархічні моделі призведе до надлишкових даних скрізь. Це також має суворі заходи безпеки (наприклад, контроль доступу), і вони важливі для налаштування додатків, що містить критичні дані, які можуть вплинути на всю поведінку системи.

2.6 Діаграми варіантів використання

Моделювання випадків використання є одним із часто застосовуваних методів моделювання при моделюванні вимог. Його основне використання полягає у захопленні взаємодії користувачів із системою. Взаємодія, яка відбувається в системі, може бути взаємодією користувачів, наприклад, жестом введення, спілкуванням із зовнішніми системами або співпрацею між компонентами системи

Знання переважних способів взаємодії користувачів із системою також дозволяє розробникам чітко визначити вимоги та побудувати більш корисну систему.

Діаграма використання - це просте представлення того, які функції система дозволяє виконувати акторам. Це забезпечує уявлення на високому рівні про взаємозв'язок між суб'єктами та функціональними можливостями. Кожен варіант використання представлений у вигляді овальної форми, а кожен актор - у вигляді фігури. Актор міг би надавати вхідні дані та отримувати результати з пов'язаних випадків використання, і ці асоціації зображуються рядком. На діаграмі показані всі актори, які безпосередньо взаємодіють із функціями системи.

Варіанти використання - це прості описи функцій системи з точки зору користувачів. Приклади використання також фіксують сценарії того, що користувач міг працювати із системою, та очікувану відповідь системи. Тим не менше, варіант використання часто використовується для врахування функціональних вимог системи та, як правило, не підходить для нефункціональних вимог. Моделювання випадків використання включає два основні артефакти: схему використання та опис справи.

Приклад використання описує, як користувач використовує систему для досягнення певної мети. Діаграма випадків використання складається із системи, відповідних випадків використання та суб'єктів і пов'язує їх між собою, щоб наочно уявити: що описується, хто використовує систему і чого хочуть досягти

актори, таким чином, випадки використання допомагають забезпечити розробку правильної системи, враховуючи вимоги з точки зору користувача.

Діаграма випадків використання зазвичай проста. Він не відображає деталей випадків використання, лише узагальнює деякі взаємозв'язки між випадками використання, суб'єктами та системами. Діаграма випадків не відображає порядок, у якому виконуються кроки для досягнення цілей кожного випадку використання.

На рис. 2.11 представлена діаграма варіантів використання системи адміністратора.

Рисунок 2.11 – Діаграма варіантів використання системи адміністратором

На рис. 2.12 представлена діаграма варіантів використання системи офіціанта.

Рисунок 2.12 – Діаграма варіантів використання системи офіціантом

На рис. 2.13 представлена діаграма варіантів використання системи кухаря.

Рисунок 2.13 – Діаграма варіантів використання системи кухарем

Варіанти використання є описами типових взаємодій між користувачами системи (адміністратором, кухарем, офіціантом) і самою системою.

2.7 Діаграма класів

Клас — це проект об'єкта. Предмети та класи йдуть рука об руку. Не можливо акцентувати одне, не говорячи про інше. Вся суть об’єктно-орієнтованого проектування стосується не об’єктів, а класів, оскільки ми використовуємо класи для створення об’єктів. Отже, клас описує, яким буде об’єкт, але це не сам об’єкт.

Насправді класи описують тип об'єктів, тоді як об'єкти є придатними екземплярами класів. Кожен об'єкт був побудований з одного набору креслень і, отже, містить однакові компоненти (властивості та методи). Стандартне значення полягає в тому, що об’єкт є екземпляром класу та об’єкта - об’єкти мають стани та поведінку.

На рис. 2.14 представлена діаграма класів.

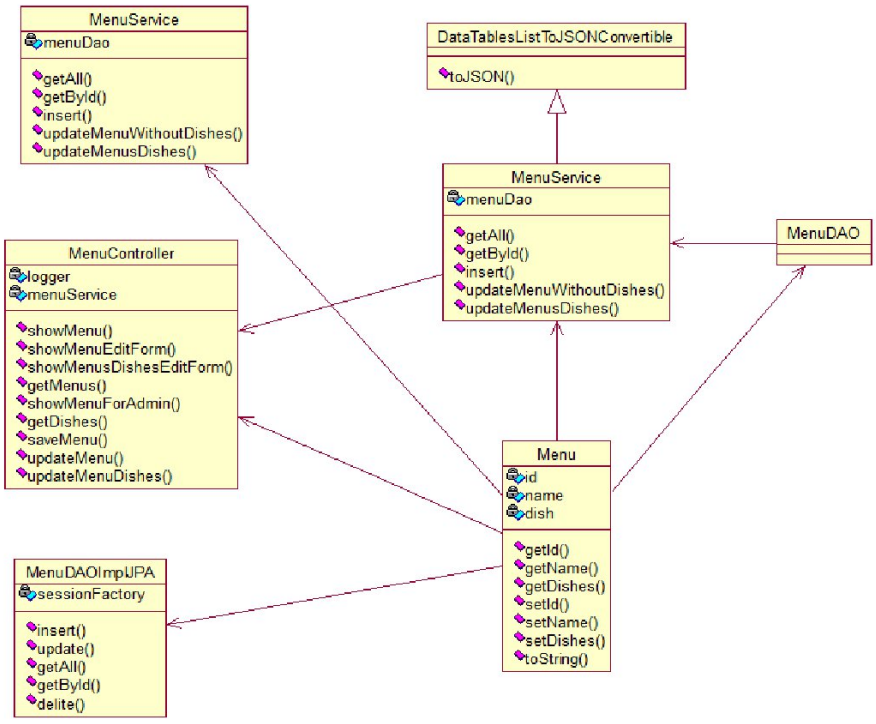

Рисунок 2.14 – Діаграма класів системи

2.8 Діаграма послідовності дій

Діаграми послідовності в UML просто зображує взаємодію між об’єктами в послідовному порядку. Призначення схеми послідовності в UML полягає у візуалізації послідовності потоку повідомлень у системі. Діаграма послідовності показує взаємодію двох життєвих ліній як упорядковану за часом послідовність подій. Діаграма послідовності показує реалізацію сценарію в системі, шляхи в системі, за якими актори беруть участь під час роботи системи. Потік повідомлень між двома або більше об’єктами представлений за допомогою вертикальної пунктирної лінії, яка простягається внизу сторінки. На схемі послідовності використовуються різні типи повідомлень та оператори. На діаграмі послідовності також використовуються ітерація та розгалуження.

На рис. 2.15 представлена діаграма послідовності.

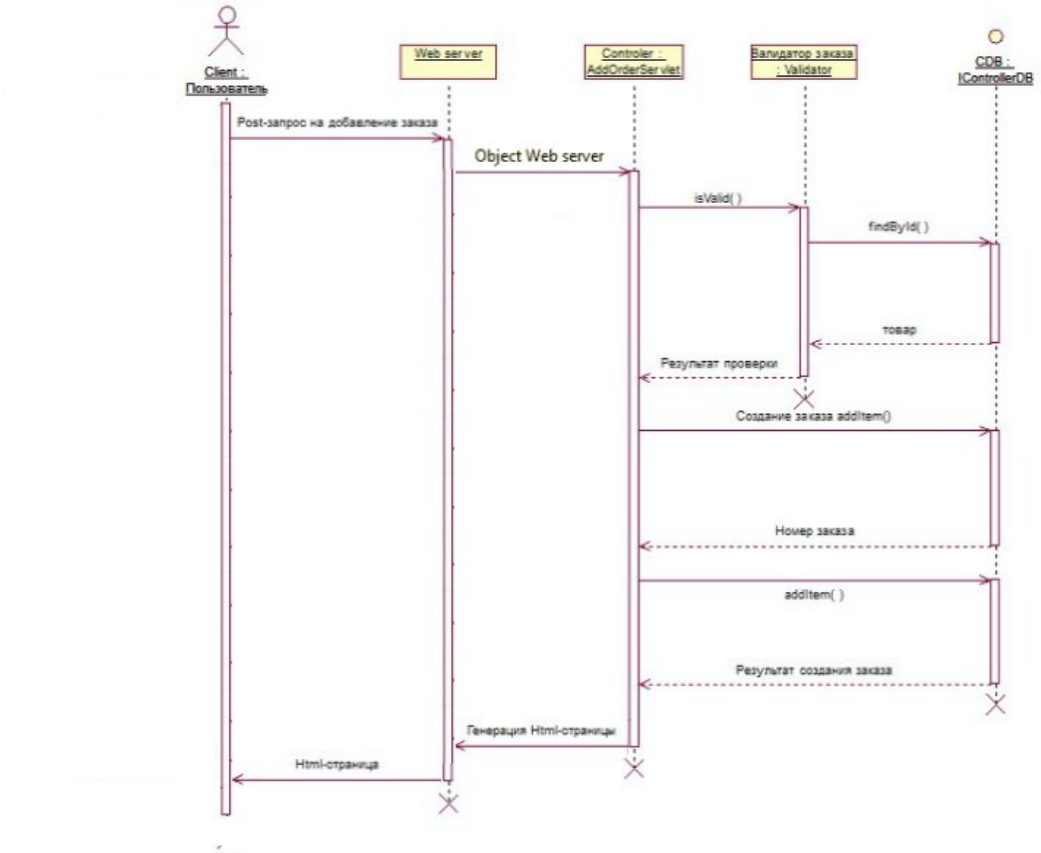

Рисунок 2.15 – Діаграма послідовності дій створення замовлення

2.9 Розробка алгоритму системи

Архітектура системи використовує шаблон проектування MVC.

Шаблон MVC стосується відповідальності трьох основних компонентів у побудові інтерфейсу. Модель стосується даних та поведінки програми. Він відповідає за надання поточного стану своїх даних та обробляє інструкції щодо зміни станів відповідно до запитів клієнта. Перегляд - це користувацький інтерфейс, який представляє стан даних і керує способом їх подання. Контролер фіксує введення даних користувача з інтерфейсу користувача та керує потоками для оновлення стану моделі та перегляду інформації.

Реалізація системи може легко врахувати зміни, прийнявши шаблон MVC. Чисте розділення проблем також дозволяє легко протестувати окрему деталь, отже, спрощує досвід налагодження. Однак шаблон MVC може вирішити проблему створення подання на стороні сервера, але не на стороні клієнта. Як зазначено у п. 5.2.2, після подання подання клієнтській стороні. JavaScript використовується для представлення поведінки у поданні. JavaScript часто потрібно націлювати на певний елемент HTML у поданні, і зміни в структурі представлення легко порушать існуючу реалізацію. Проблема тісного зв’язку між поведінкою презентації та програми все ще існує, якщо вони змішані на стороні клієнта. Отже, другий шаблон дизайну, шаблон MVVM, використовується для вирішення цієї проблеми.

На рис. 2.16 представлено алгоритм роботи серверного забезпечення.

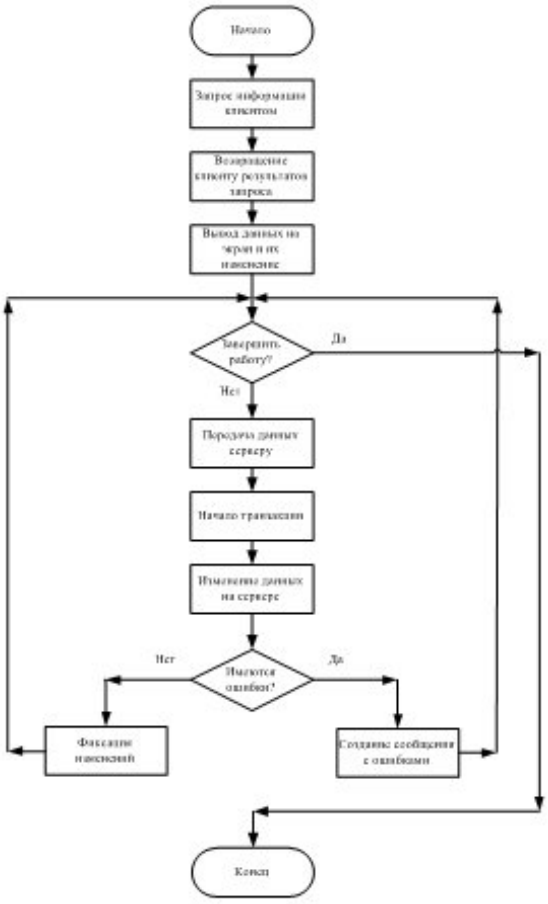

Рисунок 2.16 – Алгоритм роботи інформаційної системи

3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка серверної частини

Процес проектування довів вимоги та обмеження, яким потрібно дотримуватися. Моделі дизайну є орієнтирами, але не призначені для встановлення деталей реалізації. Архітектурне проектування створило бачення високого рівня для проектування всіх компонентів системи. Системне моделювання дозволило абстрактно подати рішення, яке могло би керувати діяльністю впровадження. Нарешті, дизайн інтерфейсу вирішив зручність використання та презентацію прототипу програмного забезпечення.

Структурна схема представлена на рис. 3.1.

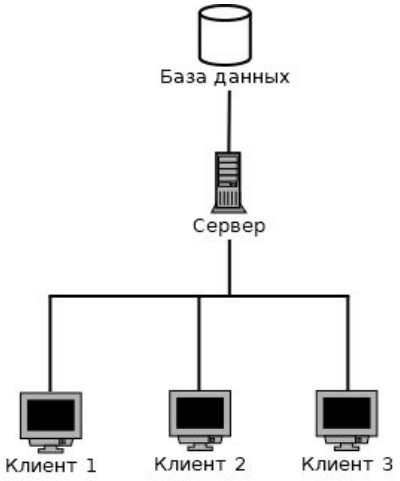

Рисунок 3.1 – Структурна схема системи

Сервер генерує веб-сторінки з HTML, CSS та Javascript, надісланими вам через інтернет. Інтернет складається з купу ресурсів, розміщених на різних серверах. Термін "ресурс" відповідає будь-якій сутності в інтернеті, включаючи файли HTML, таблиці стилів, зображення, відео та сценарії. Щоб отримати доступ до вмісту в інтернеті, ваш браузер повинен запитати у цих серверів

потрібні ресурси, а потім відобразити ці ресурси вам. Цей протокол запитів та відповідей дозволяє переглядати цю сторінку у вашому браузері.

HTTP використовується для структурування запитів та відповідей через Інтернет. HTTP вимагає передачі даних з однієї точки в іншу через мережу. Передача ресурсів відбувається за допомогою TCP (Transmission Control Protocol). Під час перегляду цієї веб-сторінки TCP керує каналами між вашим браузером та сервером. TCP використовується для управління багатьма типами підключень до мережі, при яких один комп’ютер або пристрій хоче надіслати щось іншому. HTTP - це мова команд, якої повинні дотримуватися пристрої з обох сторін з'єднання, щоб спілкуватися.

Коли користувач вводите у свій браузер адресу, програмне забезпечення наказуєте їй відкрити канал TCP для сервера, який відповідає на цю URL-адресу. URL-адреса схожа на вашу домашню адресу або номер телефону, оскільки вона описує, як з вами зв’язатися. У цій ситуації комп'ютер, який робить запит, називається клієнтом. URL-адреса, яку ви вимагаєте, — це адреса, яка належить серверу.

Після встановлення TCP-з'єднання клієнт надсилає HTTP-запит GET на сервер для отримання веб-сторінки. Після того, як сервер надіслав відповідь, він закриває TCP-з'єднання. Якщо ви знову відкриваєте веб-сайт у своєму браузері або якщо ваш браузер автоматично запитує щось із сервера, відкриється нове підключення, яке слідує тому ж процесу, що описаний вище. Запити GET — це один із методів HTTP, який може викликати клієнт.

Впровадження програмного продукту вимагає різних наборів програмного забезпечення (або фреймворків), таких як засоби програмування, система побудови та сховище вихідного коду. Використання відповідних інструментів значно покращить ефективність розробки, продуктивність та якість програмного забезпечення.

Spring Framework — це середовище додатків і інверсія контейнера управління для платформи Java. Основні функції платформи можуть використовуватися будь-яким додатком Java, але є розширення для створення

веб-додатків на основі платформи Java EE (Enterprise Edition). Хоча структура не вимагає будь-якої конкретної моделі програмування, вона стала популярною в спів товаристві Java як доповнення до моделі Enterprise JavaBeans (EJB). Spring Framework має відкритий вихідний код.

Hibernate ORM (або просто Hibernate) - це об’єктно-реляційний інструмент відображення для мови програмування Java. Він забезпечує основу для відображення об'єктно-орієнтованої моделі домену в реляційну базу даних. Hibernate вирішує проблеми невідповідності об’єктно-реляційного імпедансу, замінюючи прямі, постійні звернення до бази даних функціями обробки об’єктів високого рівня.Hibernate - це безкоштовне програмне забезпечення, яке розповсюджується за GNU Lesser General Public License 2.1. Основною особливістю Hibernate є зіставлення з класів Java на таблиці баз даних і зіставлення з типів даних Java на типи даних SQL. Hibernate також надає можливості запиту та пошуку даних. Він генерує виклики SQL і звільняє розробника від ручної обробки та перетворення об'єкта набору результатів.

Враховуючи, що цей проект був розроблений для розробки веб-додатків, він включає розробку як на стороні сервера, так і на стороні клієнта. Розробка на стороні сервера використовує об’єктно орієнтовну парадигму при реалізації необхідних алгоритмічних структур та структур даних. Таким чином, йому потрібна система веб-розробки, яка могла б подолати розрив та невідповідність між двома середовищами програмування; на стороні клієнта та сервера.

3.2 Розробка функції обробки замовлень

Для реалізації функції перегляду замовлень, редагування та створення нових було створено клас, що містить відповідні методи.

Програмний код класу меню:

```
@Controller
public class OrderController {
```

```
	private static final Logger LOGGER = (Logger)
LoggerFactory.getLogger(OrderController.class);
	private static final String NO_DISHES_MESSAGE = "Order must contain
dishes!";
	private static final String INGREDIENTS_SHORTAGE_MESSAGE = "Not
enough ingredients to fulfill the order!";
	private static final String FORBIDDEN_ACTION_MESSAGE = "This action
is forbidden!";
	private static final String UNEXPECTED_ERROR_MESSAGE ="Unexpected
error happened";
	@Autowired
	private OrderService orderService;
	// Auxiliary data source for fetching Order's existing dishes
	@GetMapping("/waiter/get_orders_dishes")
	@ResponseBody
	public DataTablesOutputDTOCollectionWrapper
getOrdersDishes(@RequestParam Long orderId) {
	DataTablesOutputDTOCollectionWrapper data = new
DataTablesOutputDTOCollectionWrapper();
	Order order = orderService.getById(orderId);
	Map<Dish, Integer> dishes = new HashMap<>();
	if (order!=null) dishes = order.getDishes();
	data.setData(orderService.toJSON(dishes));
	return data;
	}
	@PostMapping("/waiter/get_tables")
	@ResponseBody
	public List<Integer> getTables() {
	return Arrays.asList(orderService.getTables());
	}
	@PostMapping("/waiter/get_orders_table")
	@ResponseBody
	public String getOrdersTable(@RequestParam Long orderId) {
	String table = orderService.getById(orderId).getTable();
```

```
        return table;
        }
        @GetMapping("/waiter/new_order")
        public String showOrderForm() {
        return "th/waiter/new_order";
        }
        @GetMapping(value = "/admin/orders")
        public String showOrdersTableManager() {
        return "th/manager/orders";
        }
        @GetMapping(value = "/waiter/orders/today")
        public String showOrdersTableWaiter() {
        return "th/waiter/orders_today";
        }
        @GetMapping(value = "/waiter/orders/archive")
        public String showOrdersTableWaiterArchive() {
        return "th/waiter/archive/orders_archive";
        }
        @PostMapping(value = "/admin/get_orders")
        @ResponseBody
        public DataTablesOutputDTOUniversal<Order>
getOrders(DataTablesInputExtendedDTO input) {
        DataTablesOutputDTOUniversal<Order> data =
orderService.getAllOrders(input);
        return data;
        }
        @PostMapping(value = "/waiter/get_order")
        @ResponseBody
        public Order getOrder(@RequestParam Long orderId) {
        LOGGER.info("Order: "+orderService.getById(orderId));
        return orderService.getById(orderId);
        }
        @PostMapping(value = "/waiter/get_orders")
        @ResponseBody
```

```
	public DataTablesOutputDTOListWrapper<Order>
getWaiterOrders(@RequestParam Long waiterId) {
	DataTablesOutputDTOListWrapper<Order> data = new
DataTablesOutputDTOListWrapper<>();
	data.setData(orderService.getAllWaiterToday(waiterId));
	return data;
	}
	@PostMapping(value = "/waiter/get_archive")
	@ResponseBody
	public DataTablesOutputDTOUniversal<Order>
getWaiterOrdersArchive(@RequestParam Long waiterId,
	  DataTablesInputExtendedDTO input) throws JsonProcessingException
{
	DataTablesOutputDTOUniversal<Order> data =
orderService.getAllOrdersByWaiter(input, waiterId);
	return data;
	}
	@GetMapping("/waiter/edit_order")
	public ModelAndView editOrder(@RequestParam Long id) {
	ModelAndView modelAndView = new
ModelAndView("th/waiter/edit_order");
	modelAndView.addObject("id", id);
	return modelAndView;
	}
	@PostMapping("/waiter/check_order")
	public ResponseEntity checkOrder(@RequestParam Long orderId) {
	Order order = orderService.getById(orderId);
	ObjectMapper mapper = new ObjectMapper();
	ObjectNode params = mapper.createObjectNode();
	params.put("orderId", orderId);
	params.put("hasPrepared", order.hasPreparedDishes());
	params.put("isCancelled", order.isCancelled());
	params.put("isClosed", !order.isOpened());
	params.put("isFulfilled", order.isFulfilled());
```

```
	return new ResponseEntity(params, HttpStatus.OK);
	}
	@PostMapping(value="/waiter/edit_order")
	public ResponseEntity<Order> createOrUpdateOrder(@RequestBody Order
order) {
	if (order.getDishes().isEmpty()) {
	LOGGER.error("Error: "+NO_DISHES_MESSAGE);
	return new ResponseEntity(NO_DISHES_MESSAGE,
	HttpStatus.FORBIDDEN);
	}
	long startTime = System.currentTimeMillis();
	try {
	orderService.processOrder(order);
	} catch (UnsupportedOperationException e) {
	LOGGER.error("Error: "+ FORBIDDEN_ACTION_MESSAGE);
	return new ResponseEntity(FORBIDDEN_ACTION_MESSAGE,
	HttpStatus.FORBIDDEN);
	} catch (NotEnoughIngredientsException e) {
	LOGGER.error("Error: "+INGREDIENTS_SHORTAGE_MESSAGE);
	return new ResponseEntity(INGREDIENTS_SHORTAGE_MESSAGE,
	HttpStatus.FORBIDDEN);
	} catch (Exception e) {
	LOGGER.error("Error: "+e.getMessage()+
	     "Cause: "+e.getCause()+ "Class: "+e.getClass());
	return new ResponseEntity("Error: "+UNEXPECTED_ERROR_MESSAGE,
	HttpStatus.FORBIDDEN);
	}
	long endTime   = System.currentTimeMillis();
	LOGGER.info("VALIDATION RUNTIME: "+(endTime - startTime)+" ms");
	LOGGER.info("Created/Edited order: "+order);
	return new ResponseEntity(order, HttpStatus.OK);
	}
	@GetMapping("/waiter/delete_order")
```

```
	public String deleteOrder(@RequestParam Long id, HttpServletRequest
request, RedirectAttributes ra) {
	URL url;
	try {
	url = new URL(request.getHeader("referer"));
	orderService.deleteById(id);
	LOGGER.info("Deleted Order #: "+id);
	} catch (Exception e) {
	setErrorMessages(id, ra,
	HttpStatus.FORBIDDEN.toString(),
	e.getMessage(),
	"Error: Failed to deleteById the Order #: " +id);
	return "redirect:/error";
	}
	return "redirect:"+url.getPath();
	}
	@GetMapping("/waiter/close_order")
	public ResponseEntity closeOrder(@RequestParam Long id) {
	try {
	orderService.closeOrder(id);
	} catch (Exception e) {
	LOGGER.error("Failed to close Order# "+id);
	return new ResponseEntity(HttpStatus.EXPECTATION_FAILED);
	}
	LOGGER.info("Closed Order #: "+id);
	return new ResponseEntity(HttpStatus.OK);
	}
	@GetMapping("/waiter/cancel_order")
	public ResponseEntity cancelOrder(@RequestParam Long id) {
	try {
	orderService.cancelOrder(id);
	} catch (Exception e) {
	LOGGER.error("Error: Failed to cancel the Order #: "+id);
	return new ResponseEntity(HttpStatus.EXPECTATION_FAILED);
```

```
    }
    LOGGER.info("Cancelled Order #: "+id);
    return new ResponseEntity("{\"orderId\":" +id+"}", HttpStatus.OK);
    }
    private void setErrorMessages(Long id, RedirectAttributes ra,
String... messages) {
    LOGGER.error("ERROR: status: "+messages[0]+" / error:
"+messages[1]+"/ message: "+messages[2]);
    ra.addFlashAttribute("status", messages[0]);
    ra.addFlashAttribute("error", messages[1]);
    ra.addFlashAttribute("message", messages[2]);
    }
    }
```

Клас забезпечує контроль виконання функції перегляду замовлених страв за меню, редагування та додавання нових замовлень. Клас містить методи призначені для представлення списку замовлень.

3.3 Реалізація функції оформлення замовлення

Для реалізації функції оформлення замовлення були створені методи, що дозволяють робити запити та відтворювати відповідь.

Програмний код функції приведено у лістингу 3.3.

Лістинг 3.3 – Програмний код функцій обробки замовлення

```
    public class OrderServiceImplFast implements OrderService {
    private static final Logger logger = (Logger)
LoggerFactory.getLogger(OrderServiceImplFast.class);
    @Autowired
    private OrderDAO orderDAO;
    @Autowired
    private StockService stockService;
    @Autowired
    private Service service;
```

```
	@Autowired
	private OrderServerSideProcessing orderServerSideProcessing;
	@Autowired
	private OrderByWaiterServerSideProcessing
orderByWaiterServerSideProcessing;
	@Override
	public List<Order> getAll() {
	return orderDAO.getAll();
	}
	@Override
	public Long insert(Order order) {
	return orderDAO.insert(order);
	}
	@Override
	public void update(Order order) {
	orderDAO.update(order);
	}
	@Override
	public Order getById(Long id) {
	return orderDAO.getById(id);
	}
	@Override
	public void delete(Order order) {
	orderDAO.delete(order);
	}
	@Override
	public void deleteById(Long orderId) {
	orderDAO.delete(getById(orderId));
	}
	@Override
	public List<Order> getAllWaiterToday(Long waiterId) {
	return orderDAO.getAllWaiterToday(waiterId);
	}
	@Override
```

```
	public List<Order> getAll(DataTablesInputExtendedDTO dt) {
	return orderDAO.getAll(dt);
	}
	@Override
	public List<Order> getAllToday() {
	return orderDAO.getAllToday();
	}
	@Override
	public Integer[] getTables() {
	return Order.TABLE_SET;
	}
	@Override
	public void cancelOrder(Long id) {
	Order order = getById(id);
	order.setCancelled(true);
	update(order);
	}
	@Override
	public Long count() {
	return orderDAO.count();
	}
	@Override
	public Long countWaiter(Employee waiter) {
	return orderDAO.countWaiter(waiter);
	}
	@Override
	public void closeOrder(Long orderId) {
	Order order = getById(orderId);
	if (order.isCancelled()) throw
	new UnsupportedOperationException("You cannot close a cancelled
order!");
	if (!order.isOpened()) throw
	new IllegalStateException("This order has already been closed!");
	order.setOpened(false);
```

```
order.setClosedTimeStamp(new Date());
update(order);
}
```

3.4 Реалізація функції захисту інформації

Інформаційна безпека — це набір практик, призначених захистити дані від несанкціонованого доступу чи змін, як коли вони зберігаються, так і коли вони передаються з однієї машини або фізичного місця в іншу. Іноді ви можете побачити, що це називають безпекою даних.

Основними компонентами інформаційної безпеки є конфіденційність, цілісність та доступність.

Дані є конфіденційними, коли їх можуть використовувати лише ті люди, які мають до них доступ; щоб забезпечити конфіденційність, програмне забезпечення повинні мати можливість визначити, хто намагається отримати доступ до даних, і блокувати спроби тих, хто не має дозволу. Паролі, шифрування, автентифікація та захист від атак проникнення є методами, призначеними для забезпечення конфіденційності.

Цілісність означає підтримку даних у правильному стані та запобігання їх неправильному зміненню, випадково чи зловмисно. Багато методів, що забезпечують конфіденційність, також захищають цілісність даних. Зловмисники не можуть змінити дані, до яких вони не мають доступу, але існують інші інструменти, які допомагають забезпечити глибокий захист цілісності: контрольні суми можуть допомогти перевірити дані на цілісність. Наприклад, програмне забезпечення для контролю версій та часті резервні копії можуть допомогти підприємству відновити дані до правильного стану, якщо це необхідно. Цілісність також охоплює поняття відмови, система повинна мати можливість довести, що зберегла цілісність своїх даних, особливо в юридичному контексті.

В той час як програмне забезпечення повинно переконатися, що дані не можуть бути доступні неавторизованих користувачам, також повинно переконатися, що користувачі можуть отримати доступ до таких, якщо має відповідні права. Забезпечення доступності даних означає узгодження мережевих та обчислювальних ресурсів із обсягом доступу до даних, яких власник очікує, та впровадження належної політики резервного копіювання для цілей аварійного відновлення.

В ідеальному світі ваші дані завжди повинні залишатися конфіденційними, у правильному стані та доступними; на практиці, звичайно, часто потрібно робити вибір, на яких принципах інформаційної безпеки треба наголосити, а для цього потрібно оцінити ці дані.

3.5 Розробка системи зберігання даних

Цей проект використовує СУБД як метод зберігання даних. Проект прямо не обмежує вибір систему певним постачальником. Насправді використання СУБД може бути особливо різним, якщо розглядати його з етапу розробки та етапу розгортання. На стадії розробки дані часто є фіктивними та легко генеруються. Схема даних також швидко змінюється в міру зміни проектних рішень. Таблиці бази даних потрібно постійно скидати, змінювати, відтворювати; і випробовування. Таку необхідну функціональність було б краще реалізувати за допомогою легкої СУБД; що дозволить миттєво створити базу даних з нуля. З іншого боку, продуктивність та функціональність СУБД є більш вимогливими, коли система розгортається в реальному середовищі.

Для проекту обрана СУБД - PostgerssSQL, яка використовувалася на етапі розробки та остаточної повної версії на етапі розгортання та можуть бути легко інтегровані в систему. Локальна БД підтримує мінімальну конфігурацію для запуску механізму баз даних, що дозволяє розробнику зосередитися на обробці даних, а не на налаштуванні сервера. PostgerssSQL має широкий спектр інструментів для управління, моніторингу та аналізу даних, що зберігаються в

базі даних. База даних використовується для зберігання оперативних даних ресторану при їх розміщенні у виробничому середовищі.

Власник системи може розглянути можливість заміни постачальника СУБД, якщо це необхідно, після розгортання системи. Рішення ORM допомагає вирішити проблему доступу об’єктно орієнтованої моделі та моделі відношень до своїх даних. Об’єктно орієнтована парадигма проходить об'єкти через їх посилання та зв'язки з іншими об'єктами, тоді як реляційна парадигма передбачає об'єднання (або об'єднання) рядків даних таблиць. Це створює тісний зв'язок між сутностями домену та стратегіями збереження.

Використання рішення ORM зменшує велику кількість коду, що накладається на витрати при використанні попереднього методу, особливо при реалізації функціональних можливостей SQL. ORM має очевидні переваги щодо прискорення процесу розробки за рахунок зменшення перевантаження при написанні SQL-запитів. Це дозволяє розробникам зосередитись на бізнес-проблемах, а не на повторюваній логіці CRUD у SQL. Крім того, рішення ORM кешує завантажену сутність в пам’яті, зменшуючи тим самим зворотній шлях до бази даних.

Програмний код сторінки замовлення приведено у лістингу 3.6.

Лістинг 3.5 – Програмний код сторінки замовлень

```
@Transactional
public class OrderDAOImplJPA implements OrderDAO {
private static final Logger logger = (Logger)
LoggerFactory.getLogger(OrderDAOImplJPA.class);
@Autowired
private SessionFactory sessionFactory;
@Autowired
private EmployeeDAO employeeDAO;
@Autowired
private OrderServerSideDAOProcessing orderServerSideDAOProcessing;
@Autowired
```

```
	private OrderByWaiterServerSideDAOProcessing
orderByWaiterServerSideDAOProcessing;
	@Override
	public Long insert(Order order) {
	return (Long) sessionFactory.getCurrentSession().save(order);
	}
	@Override
	public void update(Order order) {
	sessionFactory.getCurrentSession().update(order);
	}
	@Override
	public Order getById(Long id) {
	return sessionFactory.getCurrentSession().get(Order.class, id);
	}
	@Override
	public void delete(Order order) {
	sessionFactory.getCurrentSession().delete(order);
	}
	@Override
	public Long count() {
		return ((Long)
sessionFactory.getCurrentSession().createQuery("select count(*) from
Order").uniqueResult());
	}
	public Long countWaiter(Employee waiter) {
	    return ((Long)
sessionFactory.getCurrentSession().createQuery("select count(*) from
Order o where o.waiter=:waiter")
	        .setParameter("waiter", waiter)
	        .uniqueResult());
	}
	@Override
	public List<Order> getAll() {
```

```
	return sessionFactory.getCurrentSession().createQuery("select
distinct o from Order o order by o.id").list();
	}
	@Override
	public List<Order> getAllWaiterToday(Long waiterId) {
	Employee waiter = employeeDAO.getById(waiterId);
	Calendar today = Calendar.getInstance();
	today.set(Calendar.HOUR_OF_DAY, 0);
	today.set(Calendar.MINUTE, 0);
	today.set(Calendar.SECOND, 0);
	return sessionFactory.getCurrentSession().createQuery("select o
from Order o where o.waiter = :waiter"+
	" and o.openedTimeStamp >= :today order by o.id desc")
	.setParameter("waiter", waiter)
	.setParameter("today", today, TemporalType.DATE)
	.list();
	}
	@Override
	public List<Order> getAllToday() {
	Calendar today = Calendar.getInstance();
	today.set(Calendar.HOUR_OF_DAY, 0);
	today.set(Calendar.MINUTE, 0);
	today.set(Calendar.SECOND, 0);
	return sessionFactory.getCurrentSession().createQuery("select o
from Order o where "+
	"o.openedTimeStamp >= :today order by o.id desc")
	.setParameter("today", today, TemporalType.DATE)
	.list();
	}
	@Override
	public List<Order> getAll(DataTablesInputExtendedDTO dt) {
	return orderServerSideDAOProcessing.getAllItems(dt);
	}
	@Override
```

```
    public List<Order> getAllOrdersByWaiter(DataTablesInputExtendedDTO
dt, String[] params) {
    return orderByWaiterServerSideDAOProcessing.getAllItems(dt,
params);
    }
    }
```

3.6 Розробка інтерфейсу клієнтської частини

При розробці веб-додатків важливим вибором є те, який двигун буде дбати про шар перегляду. Розробка на стороні клієнта в основному передбачає побудову інтерфейсів (переважно написаних у форматі HTML) для веб-браузера. Розробка інтерфейсу користувача здійснена з використанням Thymeleaf.

Сторінки Java Server раніше були дуже популярними, хоча накладні витрати та витрата часу були основними недоліками їх використання. Вони вимагали значних змін у HTML сторінок. У наш час Thymeleaf широко прийнятий і використовується як шаблонний двигун для додатків Spring та MVC. Він також може бути використаний для розширеного шаблонування електронної пошти HTML. Поки JSP компілюються до класів сервлетів Java, Thymeleaf аналізує прості файли шаблонів HTML. На основі виразів, наявних у файлі, він видає статичний вміст. Він здатний обробляти HTML, XML, JS, CSS тощо.

Програмний код сторінки замовлення приведено у лістингу 3.6.

Лістинг 3.6 – Програмний код сторінки замовлень

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8"/>
<title>Orders</title>
<link rel="stylesheet" type="text/css" media="all"
href="/css/lib/bootstrap-3.3.7.min.css" />
```

```
      <link rel="stylesheet" type="text/css"
href="//cdn.datatables.net/1.10.13/css/jquery.dataTables.css"/>
      <link rel="stylesheet" type="text/css"
href="/css/filterDisabled.css"/>
      <script src ="/js/lib/date.format-1.2.3.js"/>
      <script src ="/js/lib/moment.js"/>
      <script src ="/js/lib/jquery-1.12.1.js"/>
      <script src ="/js/lib/bootstrap-3.3.7.min.js"/>
      <script src ="/js/lib/jquery.dataTables-1.10.12.min.js"/>
      <script src ="/js/lib/jquery-ui-1.12.1.js"/>
      <script src="/js/lib/stomp.js"></script>
      <script src="/js/lib/sockjs.min.js"></script>
      <script src ="/js/waiter/new_or_edit_order.js"/>
      <script src ="/js/waiter/orders_today.js"/>
      <script src ="/js/waiter/orders_today_messaging.js"/>
      </head>
      <body>
      <div class="container">
      <div class="row" style="margin-top:20px;">
      <div class="panel panel-primary">
      <div class="panel-heading">
      <h3 class="panel-title">Orders on: <span id="date"
name="date"></span></h3>
      </div>
      <div class="panel-body">
      <div th:replace="/th/header :: header"/>
      <details>
      <summary>Messages</summary>
      <textarea class="form-control" rows="5" id="messages"
name="messages" disabled="true"/>
      </details>
      </div>
      <div class="panel-body">
      <div class="table-responsive">
```

```
<table id="ordsTable" name="ordsTable"
   class="display table table-hover" cellspacing="0" width="100%"
   style="overflow-x:auto">
<thead>
<tr>
<th>Id</th>
<th>isOpened</th>
<th>Opened</th>
<th>Closed</th>
<th>Table, #</th>
<th>Dishes</th>
<th>Total, $</th>
<th>Readiness, %</th>
<th>isFulfilled</th>
<th>isCancelled</th>
<th>Edit</th>
<th>Cancel</th>
<th>Close</th>
</tr>
</thead>
</table>
</div>
</div>
</div>
<div class="form-group">
<br/>
<!--<button onclick="reloadOrdersTable()"
class="btn">Update</button>-->
<button type="button" data-toggle="modal" data-target="#modal"
class="btn btn-primary">New Order</button>
<a target="_blank" href="/waiter/orders/archive">
<input type="button" class="btn btn-default" value="Archive"/>
</a>
</div>
```

```
      </div>
      </div>
      <!-- Modal Order -->
      <div class="modal" id="modal">
      <div class="modal-dialog modal-md">
      <div class="modal-content">
      <div class="modal-header">
      <button type="button" class="close" data-dismiss="modal" aria-
hidden="true">*</button>
      <h3 class="text-center">New Order</h3>
      </div>
      <div class="modal-body">
      <form id="orderForm" action="/waiter/edit_order" method="post"
onsubmit="return false;">
      <div class="form-group">
      <input type="hidden" id="id" name="id" value="0"/>
      </div>
      <div class="form-group">
      <label for="table">Table #</label>
      <select id="table" class="list-group" title="Select a table
number">
      </select>
      </div>
      <div class="form-group">
      <table id="odTable" class="display" cellspacing="0" width="100%"
style="overflow-x:auto">
      <thead>
      <tr>
      <th hidden="true">Id</th>
      <th>Dish</th>
      <th>Quantity</th>
      <th>Price, $</th>
      <th>Del</th>
      </tr>
```

```
      </thead>
      </table>
      </div>
      <div class="form-group">
      <br/>
      <button class="btn btn-default" type="button" data-toggle="modal"
data-target="#myModal">Add
      Dish
      </button>
      <button class="btn btn-primary" type="submit">Confirm</button>
      </div>
      <div id="feedback"></div>
      </form>
      </div>
      <div class="modal-footer">
      <a href="#" data-dismiss="modal" class="btn">Close</a>
      </div>
      </div>
      </div>
      </div>
      <!-- Second modal Dish -->
      <div class="modal" id="myModal" data-backdrop="static">
      <div class="modal-dialog">
      <div class="modal-content">
      <div class="modal-header">
      <button type="button" class="close" data-dismiss="modal" aria-
hidden="true">*</button>
      <h4 class="modal-title">Dishes</h4>
      </div>
      <div class="modal-body">
      <table id="dTable" class="display" cellspacing="0" width="100%"
style="overflow-x:auto">
      <thead>
      <tr>
```

```
<th>Id</th>
<th>Dish</th>
<th>Category</th>
<th>Price, $</th>
<th>Weight, g</th>
<th>Add</th>
</tr>
</thead>
</table>
</div>
<div class="modal-footer">
<button type="button" class="btn btn-default" data-
dismiss="modal">Close</button>
</div>
</div>
</div>
</div>
</body>
</html>
```

3.7 Тестування розробленого програмного забезпечення

Тестування програмного забезпечення - це процес перевірки роботи впровадження програмного забезпечення на відповідність його вимогам, є процесом оцінки та перевірки того, що програмний продукт або додаток виконує те, що йому слід робити. Це дозволяє повідомити розробника про те, що можливі помилки існують до того, як програмне забезпечення буде використане фактичним користувачем (кінцевою зацікавленою стороною або клієнтом). Переваги тестування включають запобігання помилкам, зменшення витрат та часу на розробку, підвищення продуктивності.

Якщо в програмному забезпеченні є будь-які помилки, вони будуть виявлені на ранньому етапі і можуть бути швидко усунені. Правильно

протестований програмний продукт забезпечує надійність, безпеку і високу продуктивність, що в подальшому призводить до економії часу, рентабельності та задоволеності клієнтів.

Тестування програмного забезпечення є повторюваним та одноманітним завданням, оскільки воно передбачає повторюваний цикл виконання дії та перевірку результату. Зокрема, коли кількість елементів, необхідних для тестування, зростає, проведення тестування вручну може бути трудомістким та неефективним. Отже, більшість тестувань програмного забезпечення використовують інструменти для автоматизації процесу тестування. Засіб автоматизації тестів може зробити тестування більш ефективним та швидшим, автоматично виконуючи набір визначених тестових випадків та перевіряючи їх результат. Це передбачає написання сценаріїв для визначення цих кроків.

Існує багато різних типів тестів програмного забезпечення, кожен із яких має певні цілі та стратегії. Приймальне тестування є перевіркою того, чи вся система працює належним чином. Блокове тестування: перевірка того, що кожен програмний блок працює належним чином. Функціональне тестування забезпечує перевірку функцій шляхом емуляції бізнес-сценаріїв на основі функціональних вимог. Тестування чорної скриньки є поширеним способом перевірки функцій. Тестування продуктивності перевіряє роботу програмного забезпечення при різних робочих навантаженнях. Наприклад, тестування на навантаження використовується для оцінки продуктивності в реальних умовах навантаження. Стрес-тестування визначає наскільки велика напруга може сприйняти систему до її виходу з ладу, та вважається різновидом нефункціонального тестування. Тестування зручності використання перевіряє наскільки добре клієнт може використовувати систему або веб-додаток для виконання завдання. Регресійне тестування перевіряє порушують чи погіршують функціональність нові функції. Перевірка розумності може бути використана для перевірки меню, функцій і команд на поверхневому рівні, коли немає часу на повний тест регресії.

У кожному випадку перевірка базових вимог є критичною оцінкою. Не менш важливим є те, що дослідницьке тестування допомагає тестувальнику або команді тестувальників розкрити важко передбачувані сценарії та ситуації, які можуть призвести до помилок програмного забезпечення.

Навіть просте застосування може пройти велику кількість та різноманітність тестів. План управління тестами допомагає визначити пріоритети, які типи тестування забезпечують найбільше значення з урахуванням наявного часу та ресурсів. Ефективність тестування оптимізується шляхом проведення найменшої кількості тестів для виявлення найбільшої кількості дефектів.

Ітеративні розробки програмних продуктів часто вносять нові зміни до поведінки системи та інтерфейсів. Тестування інтеграції забезпечення взаємодії програмних компонентів або функцій. Однак ступінь їх впливу на існуючи системи може бути важко оцінити. Поточні функції можуть перестати працювати або після цих змін можуть з’явитися нові помилки. Це знову узгоджується з принципом безперервної інтеграції, згідно з яким кожні зміни повинні бути належним чином перевірені перед інтеграцією їх в основу коду.

Тестування системи — це тестування повністю інтегрованої системи в цілому. Це етап тестування після інтеграційного тестування; але здійснюється на всіх компонентах системи. Вона спрямована на виявлення небажаної поведінки, коли всі компоненти працюють разом. Він також перевіряє, чи відповідає система вимогам та очікуванням. Крім того, це також допомагає зрозуміти межі системи та гарантує надійність системи. На цьому етапі модульне тестування та інтеграційне тестування вирішили більшість проблем функціональності.

Інтеграційне тестування — це тестування декількох компонентів, які працюють разом. Часто ці компоненти тестувались окремо перед тестом інтеграції. Проблеми інтеграційного тестування пов'язані з тестуванням інтерфейсів компонентів та їх взаємодії. Це також досліджує методи, що використовуються для обміну даними між компонентами. Це допомагає

розкрити такі проблеми, як викриття недійсного інтерфейсу або те, що передані дані не мають сумісного формату.

Модульне тестування — це метод тестування, які фокусується на найменшій чи окремій одиниці програмного забезпечення. У середовищі розробки найменша одиниця часто відноситься до класу та їх методів. Тест перевіряє шляхи управління в методі для забезпечення максимального охоплення та виявлення помилок. Він перевіряє, чи методи дають правильний результат при тестуванні для певних наборів вхідних даних. Крім того, він перевіряє, що поведінка та функціональність методу відповідають його цілі.

Задля уникнення помилок, під час розробки інформаційного забезпечення використовувався метод модульного тестування. Це допомогло виправити помилки на початку циклу розробки та заощадити час. У майбутньому, це дозволяє швидко вносити зміни та повторно використовувати код.

JUnit - це модульна система тестування для мови програмування Java. JUnit мав важливе значення в розробці прототипів і є одним із сімейства модульних фреймворків тестування, спільно відомих як xUnit, що походять з JUnit. Платформа JUnit служить основою для запуску тестових платформ на JVM. Він також визначає API TestEngine для розробки тестової основи, яка працює на платформі. Крім того, платформа надає панель запуску консолі для запуску платформи з командного рядка та Runner на основі JUnit 4 для запуску будь-якого TestEngine на платформі в середовищі на основі JUnit 4. Першокласна підтримка платформи JUnit також існує в популярних середовищах розробки (див. IntelliJ IDEA, Eclipse, NetBeans та Visual Studio Code) та інструментах побудови (див. Gradle, Maven та Ant).

ВИСНОВКИ

Проект пройшов ряд заходів з розробки комплексного рішення для інформаційного забезпечення закладів громадського харчування, а саме ресторанів. Після аналізу мети проекту та напрямку дослідження було встановлено набір цілей. Всі заходи, здійснені під час проекту, були спробами реалізації цих цілей. Наприкінці проекту був розроблений прототип програмного забезпечення.

Застосування гнучкого методу розробки також виявилось корисним для управління процесом розробки програмного забезпечення. Прототип програм постійно розвивався завдяки поступовому та повторюваному циклу розробки. Його перевіряли на кожній ітерації, отже більшість дефектів були усунені на початку проекту. План проекту постійно переглядався з урахуванням прогресу та можливостей на основі цих методів.

Проект був амбіційним та трудомістким, впроваджуючи якомога більше можливостей за дуже обмежені терміни. Проект успішно задовольнив функціональні вимоги системи. Ці вимоги мають головний пріоритет і відображають найбільш необхідні функції, які вимагають зацікавлені сторони. Розроблена система може використовуватися з урахуванням потреб зацікавлених сторін та оперативних проблем.

Результатом роботи є інформаційне забезпечення, що задовольняє зазначеним вимогам, вимогам законодавства, правилам приготування страв та обслуговування клієнтів закладу, технічного забезпечення та іншим факторам.

Проект успішно реалізував робочий комплексний прототип інформаційного забезпечення системи закладів громадського харчування. Впроваджений прототип програмного забезпечення був повністю протестований на всіх етапах проекту і продемонстрував прийнятну продуктивність.

Цей звіт також задокументував усі відповідні деталі дослідження та процеси прийняття рішень. Якщо будуть розроблені майбутні розширення

системи, звіт буде корисним для доповнення решти вимог та майбутніх удосконалень, які можуть бути залучені. Підводячи підсумок, проект виконав свої цілі та є таким, що сприяти діяльності ресторанів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sitnikov D. Analysis of ways for exchanging data in networks with package commutation / D.Sitnikov // Radio Electronics, Computer Science, Control. – December 2018. (Web of Science Core Collection)

2. M. Vyshniak, I. Klymova The basic issues of knowledge management bionics. // Bionics of intelligence, 2018, № 1 (90), pp. 24-30

3. Капустин, Н. М. Автоматизація виробничих процесів у машинобудуванні: Учеб. для втузів [Текст]/ Под ред. Н. М. Капустина. — М.: Вища школа, 2004. — 415 с.

4. Автоматизація транспортної галузі [Електронний ресурс]. - Режим доступу: http: //www.controlengrussia.com - Загол.с екрану.

5. Менеджмент готелів та ресторанів: Посібник. [Текст]/ - 2-е вид. - Мн .: Нове знання, 2001. - 216с

6. Ресторанний персонал та автоматизація [Текст]/ д. Солдатенков Ресторанні відомості, 2005. - 192с

7. Підручник ресторатора. Катсігріс Костас, Томас Кріс./ Ресторанні відомості, 2009. - 576с

8. IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/IDEF0.

9. IDEF0: функціональне моделювання процесів [Електронний ресурс] – Режим доступу до ресурсу: http:// www.info-system.ru/ designing / methodology/ sadt/ sadt_for_bp.html.

10. The Complete Guide to UML Diagram Types with Examples [Електронний ресурс] – Режим доступу до ресурсу: http://creately.com/blog/diagrams/uml-diagram-types-examples/#Activity Diagram.

11. Эспозито Д. Г. Разработка современных веб-приложений. Анализ предметных областей и технологий / за ред. Вильямс. 2016. 465 с.

12. Каледина М. А. Как правильно подобрать интерфейс / за ред. Московского государственного университета. 2015. 193 с.

13. Базалева О.Н. Мастерство визуализации / за ред. Диалектика 2016. 192 с.

14. Хмарні обчислення // Google Sites. URL: https://sites.google.com/site/hmaarniobcislenna/

15. What is cloud computing? A beginner’s guide // MS Azure. URL: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

16. Cloud Computing // International Business Machines. URL: https://www.ibm.com/cloud/learn/cloud-computing/

17. Загальні методичні вказівки з дипломного проектування в університет / П.С. Ковтун, З.В. Дудар, В.Я. Журавльов, О.С. Шкіль. - Харків: ХНУРЕ, 2015. 44 с.