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 вебсайту бронювання місць в барбершопі з використанням технології Ruby on Rails

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

_____ Петльовий Б.С.
(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

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)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Петльовий Богдан Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебсайту бронювання місць в барбершопі з використанням технології Ruby on Rails

Керівник роботи к.т.н., ст. викладач каф. ПІ Стоянов Ю.М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2023 року № _____

2. Термін подання студентом завершеної роботи 20 червня 2023р.

3. Вихідні дані до роботи вимоги до фунціоналу додатка, використання бази даних MongoDB, використання фреймворку Ruby on Rails

4. Зміст роботи (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С., к.т.н., доцент кафедри МТ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи		
2.	Підбір джерел інформації щодо створювання веб додатків на базі Ruby on Rails		
3.	Переклад та опрацювання інформації з вибраних джерел		
4.	Розробка веб додатку на основі отриманої інформації		
5.	Оформлення звіту по виконаній роботі		
6.	Виконання завдання до підрозділу «Безпека життєдіяльності»		
7.	Виконання завдання до підрозділу «Основи охорони праці»		
8.	Підготовка презентації і доповіді		
9.	Нормоконтроль		
10.	Перевірка на плагіат		
11.	Попередній захист кваліфікаційної роботи		
12.	Захист кваліфікаційної роботи		

Студент _____ Петльовий Б.С.
(підпис) (прізвище та ініціали)

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

РЕФЕРАТ

Розробка вебсайту бронювання місць в барбершопі з використанням технолії Ruby On Rails// Кваліфікаційна робота освітнього рівня «Бакалавр» // Петльовий Богдан Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра прогограмної інженерії, група СП-41 // Тернопіль, 2023 // С. 44, рис. – 38, табл. – 2 , кресл. – , додат. – 1, бібліогр. – 24.

Ключові слова: RubyOnRails, MongoDB, Visual Studio Code, Github, вебсайт, адміністратор, користувач, власник салону, майстер, база даних, лістинг, діаграма.

В першому розділі кваліфікаційної роботи проаналізовано готові програмні рішення конкурентів, розглянуто актуальність розробки вебсайту для бронювання місць в барбершопі та технічний аспект проблеми.

В другому розділі кваліфікаційної роботи здійснено розробку загальної структури та вигляду вебсайту, спроектовано архітектуру вебсайту.

В третьому розділі кваліфікаційної роботи реалізовано ключові класи вебдодатку, проведено тестування додатку.

ANNOTATION

Development of a website for booking appointments in a barbershop using Ruby On Rails technology // Qualification work of the Bachelor's degree // Petliovyi Boghdan Sergiyovich // Ivan Puluj National Technical University of Ternopil, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SP-41 group // Ternopil, 2023. // Pages - 44, figures - 37, tables - 2, addition - 1, literature - 24.

Keywords: RubyOnRails, MongoDB, Visual Studio Code, GitHub, website, administrator, user, salon owner, master, database, listing, diagram.

The first section of the qualification work analyzes existing software solutions of competitors, discusses the relevance of developing a website for booking appointments in a barbershop, and examines the technical aspect of the problem.

The second section of the qualification work focuses on the development of the overall structure and appearance of the website, and designs the architecture of the website.

The third section of the qualification work implements the key classes of the web application and conducts testing of the application.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ,СКОРОЧЕНЬ І ТЕРМІНІВ

ПК – персональний комп’ютер.

CSS – cascading style sheets (каскадні таблиці стилів).

HTML – hypertext markup language (мова розмітки гіпертекстових документів).

JS – це невибаглива до ресурсів мова програмування з функціями першого класу, код якої інтерпретується та компілюється під час виконання.

Ruby (англ. «Рубін») — це повністю об'єктноорієнтована мова програмування з чіткою динамічною типізацією.

Ruby on Rails – це фреймворк для створення додатків, написаний на мові Ruby

UML – мова візуального представлення ідей розробника

DB (database) – це організована структура, яка призначена для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів.

API (Application Programming Interface) - інтерфейс програмування додатків, програмний інтерфейс програми.

IDE(Integrated Drive Electronics) - комплексне програмне рішення для розробки програмного забезпечення.

Android – мобільна платформа створена компанією Google.

IOS - це операційна система Apple.

Visual Studio Code - засіб для створення, редагування та зневадження сучасних вебзастосунків і програм для хмарних систем.

Github – це один з найбільших вебсервісів для спільної розробки програмного забезпечення.

MongoDB – це база даних, яка не потребує опису схеми таблиць.

ЗМІСТ

ВСТУП

У сучасному світі інтернет відіграє важливу роль у бізнес-середовищі. Веб-додатки стали необхідним інструментом для підтримки діяльності різних підприємств та організацій. Вони надають можливість клієнтам отримувати інформацію, здійснювати покупки та взаємодіяти з певними послугами безпосередньо зі свого домашнього комп'ютера чи мобільного пристрою.

Однією зі сфер бізнесу, які активно використовують інтернет ресурси є сфера послуг, в яку входять барбершопи. Барбершопи — це спеціалізовані заклади, які надають послуги з чоловічого стрижки, гоління та догляду за волоссям та бородою. Зростання популярності таких перукарень останніми роками привело до збільшення конкуренції в цій галузі. У таких умовах необхідною стає наявність ефективних та інноваційних інструментів для залучення клієнтів та полегшення процесу бронювання місць.

Метою цієї бакалаврської роботи є розробка веб-додатку для бронювання місць в барбершопі з використанням технології Ruby on Rails[1]. Використання фреймворку Ruby on Rails дозволить швидко та ефективно створити веб-додаток. Також це дозволить розробити потужну та функціональну платформу для бронювання місць, що забезпечить зручність для клієнтів та полегшить управління бізнес-процесами для власників.

Планується використати інтерактивний дизайн та зручний інтерфейс, щоб забезпечити користувачам простоту та зручність взаємодії з додатком. Крім того, розробка буде включати функціонал для керування записами клієнтів, розкладом роботи майстрів, оглядом послуг та їх цінами, а також можливістю здійснювати онлайн-платежі.

Очікується, що даний проект сприятиме покращенню процесу резервування місць для клієнтів та збільшенню кількості замовлень. Крім того, розробка такого додатку може сприяти покращеню умов бронювань місць у сфері послуг.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДЛЯ РОЗРОБКИ ВЕБСАЙТУ «CHICK-CHICK»

1.1 Огляд конкурентів

Одним із можливих конкурентів в сфері послуг, а саме перукарень є компанія із Сполучених Штатів Америки ‘StyleSeat’(рис. 1.1), вони є популярною платформою для бронювання послуг. Також вони співпрацюють з широким колом перукарень по всьому світі, завдяки цьому вони можуть пропонувати широкий вибір перукарень та майстрів. Немало важливим для клієнтів є те, що на цій платформі є досить відгуків про будь-які салони.

Рис. 1.1 – Логотип компанії конкурента StyleSeat

Клієнти StyleSeat можуть зручно використовувати для бронювання як їх вебсайт(рис. 1.2) так і мобільний додаток, який представлений для Android і для IOS версії. Користувачі також мають змогу самим обирати зручний для них час прийому, після чого система буде провіряти доступність цього відрізку часу.

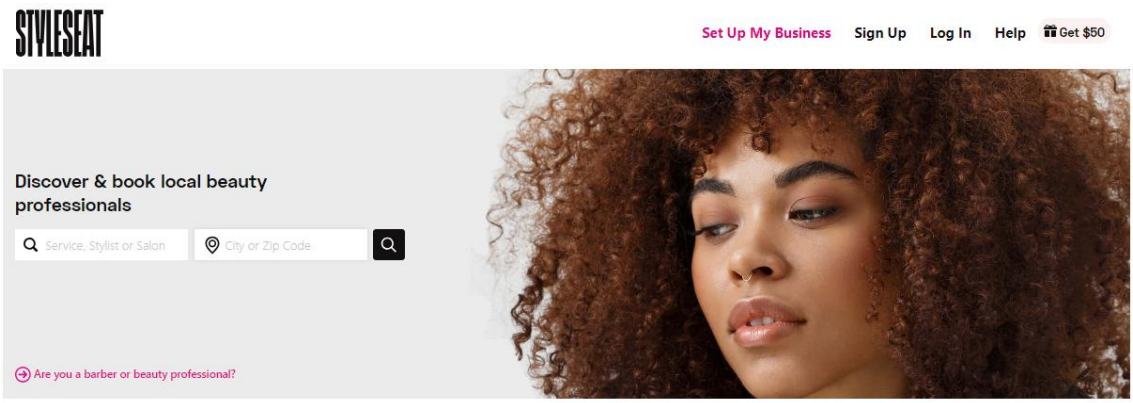

Рис. 1.2 – Вебсайт компанії StyleSeat

Ця система також досить зручна не тільки для клієнтів, а й для майстрів салонів представлених в базі даних. Кожен майстер має свій профіль в системі, в якому він може презентувати свої послуги, ціни, портфоліо та отримувати відгуки від клієнтів. Це дає змогу клієнту отримати повну інформацію про того чи іншого майстра і відповідно прийняти більш обґрунтоване рішення.

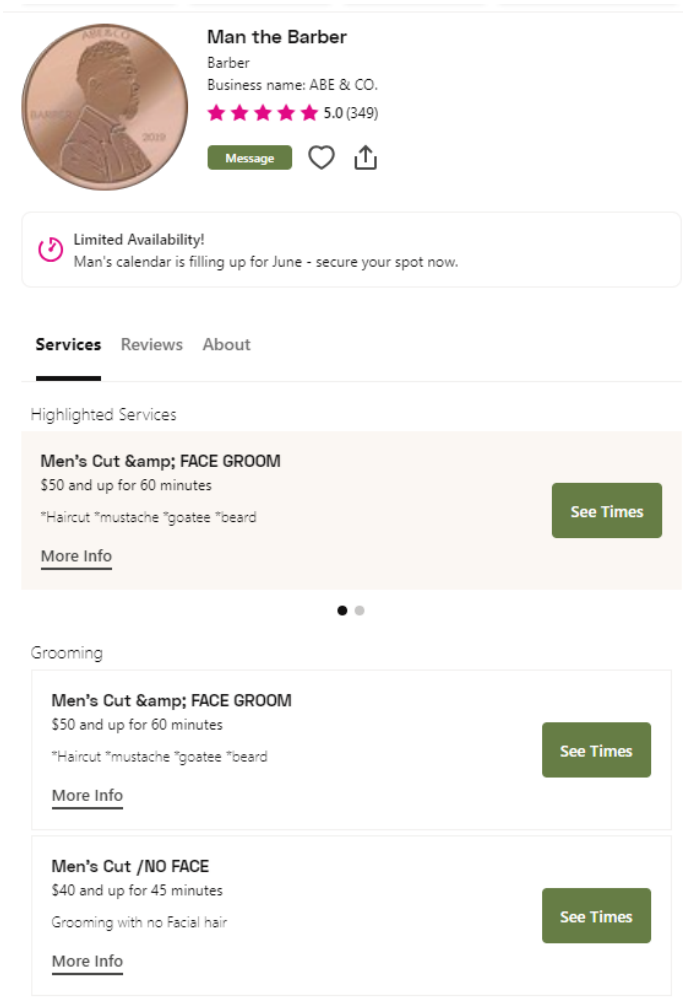

Рис. 1.3 – Приклад оформлення профілю майстра

Також платформа робить нагадування для клієнтів і майстрів про броювання надсиланням повідомлення на електрону почту або в вигляді сповіщення на телефоні. Важливою складовою цього продукту є також можливість для власників салонів ефективно керувати своїм бізнесом включаючи інтегрований календар, можливість обліку фінансів та можливість маркетингу через внутрішню аудиторію продукту.

Підсумовуючи все описане можна сказати, що компанія ‘StyleSeat’ є досить серйозним конкурентом на ринку Сполучених Штатів Америки, адже вона створює зручні в користуванні застосунки як для клієнтів, так і для фахівців з краси.

1.2 Обґрунтування вибору напрямку дослідження

Мною було обрано саме цей напромок, адже в нас час є досить актуальної тема економії часу, саме для цього і було обрано даний напрямок, а саме напрям сфери послуг. Досить частим явищем в наш час є різного роду доставки, бронювання машин, номерів готелів, інтернет магазини, електроні направлення в лікарні чи отримання паспорта, або водійського посвідчення онлайн. В зв’язку з цим мені було цікаво чи є якийсь вебдодаток для бронювання місць в чоловічі перукарні, виявилось що в Україні їх досить мало, і це може стати досить корисною і можливо прибутковою справою.

В зв’язки з тим, що розробка потребує малої кількості ресурсів це дасть мені змогу додати більше зручних і корисних функцій в сам додаток. В подальшому розвитку вебдодатку в мене буде змога зробити його мультиплатформеним, щоб користувачі смартфонів, планшетів змогли користуватись додатком.

Рис. 1.4 – Приклад мультиплатформеності

З додаванням мультиплатформеності в додаток мені вдасться розширити спільноту користувачів, які зможуть користуватись додатком з будь-якого місця чи пристрою.

Онлайн бронювання місць дозволить уникнути більшості помилок чи непорозумінь, які могли б виникнути при взаємодії з персоналом перукарень. Це забезпечить точність даних, що буде позитивно впливати на репутацію перукарні.

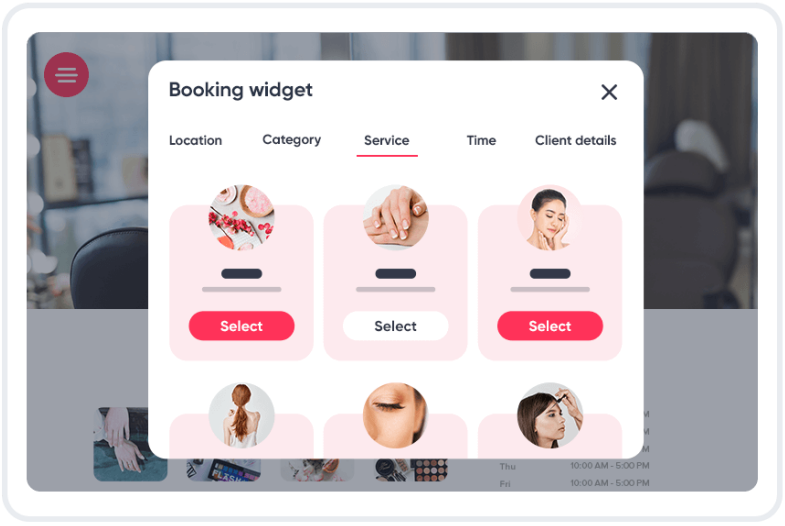

Рис. 1.5 – Приклад онлайн бронювання місць в перукарні

Також такий додаток дозволить зняти додаткову нагрузку з адміністраторів, адже їм буде достатньо просто перевірити дату та час бронювання, для отримання повної інформації на яку послугу і до кого записався клієнт.

1.3 Технічний аспект проблеми

Основуючись на цих даних мені стало цікаво використати фреймворк Ruby on Rails, адже він є досить ефективним і швидкісним фреймворком для розробки веб додатків. Він має досить вбудованих інструментів, шаблонів та велику кількість готових рішень, що є досить зручним для розробників.

Рис. 1.6 – Логотип Ruby on Rails

В Ruby on Rails є велика спільнота, яка може допомогти початківцям з базовими речами, також регулярні обновлення та виправлення помилок роблять цей фреймворк досить актуальним на даний час. Це дає змогу використовувати різні актуальні ресурси для підтримання і поліпшення додатків на базі Ruby on Rails.

Фреймворк Ruby on Rails[2] дозволяє легко інтегрувати в свої застосунки різні зовнішні сервіси, API та розширення. Це особливо важливо для мого додатку, адже це дозволить доволі легко підключити платіжні системи, інтегрований календар для бронювання, перевірку актуальності вибраної дати, систему сповіщень та багато іншого для зручності користування вебдодатком.

На сьогоднішній день фреймворк Ruby on Rails є відомим далеко не в останню чергу за рахунок своєї стабільності та надійності. В ньому є вбудовані механізми для автоматичного тестування, що дозволяють забезпечити якість коду та запобігти появі помилок під час роботи сайту.

Також в якості IDE було вибрано Visual Studio Code[3] так, як цю IDE можна використовувати на будь-якій системі. Visual Studio Code має інтуітивно зрозумілий інтерфейс в якому зручно проектувати проекти будь-якого розміру, в цьому інтерфейсі є такі функції, як автодоповнення, підказки, можливість налаштування інтерфесу під свої потреби.

Рис. 1.7 – Логотип Visual Studio Code

На даний момент Visual Studio Code пітримує інтеграцію готових Github[4] проектів і подальшу взаємодію з ними, тобто можна виконувати пуші, коміти та пуші без необхідності постійно виходити з редактора. Немало важливим є те, що це IDE має підтримку багатьох сучасних мов програмування, в тому числі Ruby і його фреймворка Ruby on Rails, що є необхідною умовою для роботи над проектом.

В якості бази даних мною було використано базу даних MongoDB[5], адже вона є досить гнучкою в плані визначення структури ти схем даних.

Використовуючи цю базу даних можна не дотримуватися жорсткої схеми, що дозволяє сильно полегшити подальший розвиток додатку.

Рис. 1.8 – Логотип MongoDB

Ця база даних підтримує горизонтальне масштабування, що дозволяє їй краще підтримувати масштабні додатки. Важливим моментом при виборі саме цієї бази даних була її змога обробляти великі обсяги даних, це важливо адже під час використання її в моєму додатку мені потрібно буде обробляти постійні запити на додавання тої чи іншої інформації.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБСАЙТУ «CHICK-CHICK»

2.1 Розробка загальної структури та вигляду вебсайту

Вебсайт барбершопу «CHICK-CHICK»[6] складається з семи сторінок зв’язаних між собою, доступ до яких можна отримати ввівши правильні дані. На головній сторінці сайту користувач буде бачити вікно для реєстрації, де він побачить вибір між ключовими ролями додатку. А саме користувач, майстер, власник салону. Після реєстрації і вибору ролі користувач буде направлений в залежності від вибору на стартову сторінку для кожної ролі. Після реєстрації звичайний користувач зможе налаштувати свій профіль, або відразу перейти до бронювання місця в доступних перукарнях.[7]

Основними кольорами дизайну вебдодатку є синій, його відтінки, чорний і білий. Вебсайт дотримується певного стилю оформлення. Тому фон усього вебдодатку має вигляд як на рисунку 2.1.

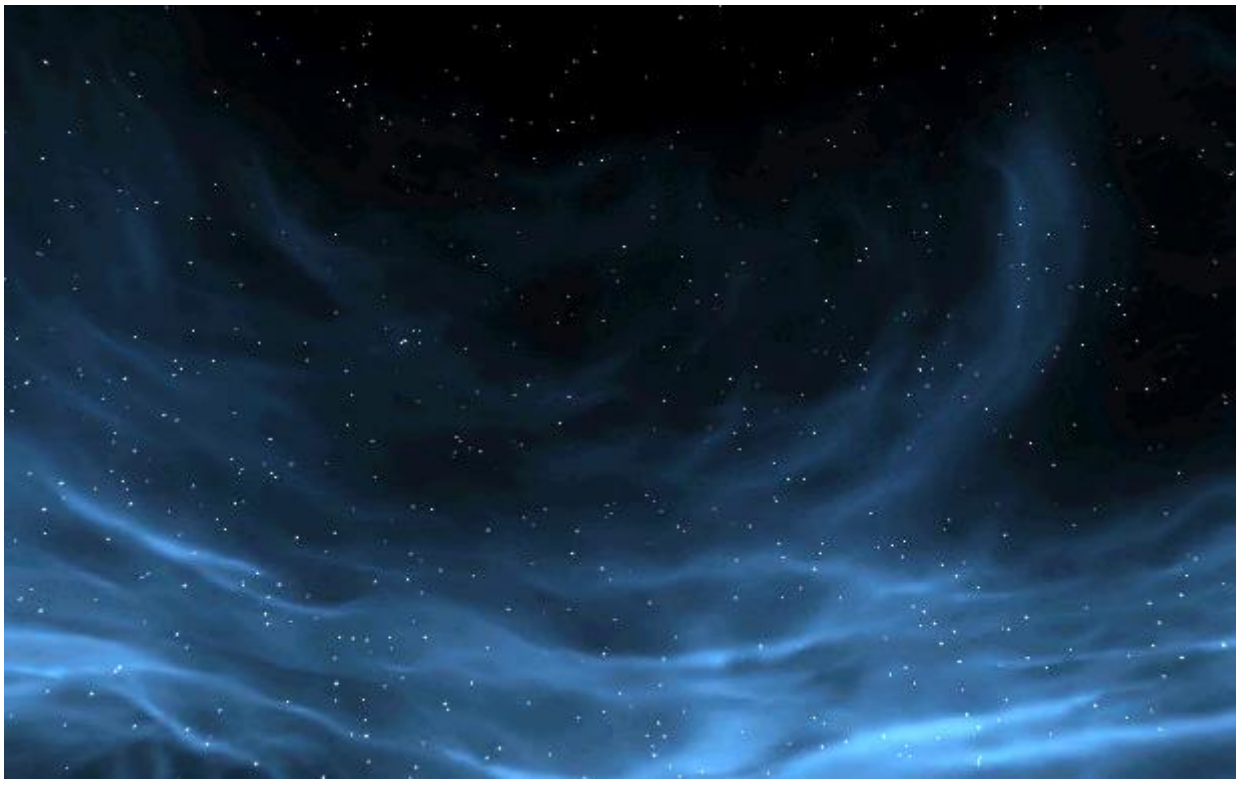

Рис. 2.1 – Дизайн фону

Коли користувач завершив всі базові налаштування він має змогу перейти до меню вибору послуги в якому має вибрати барбершоп, майстра й послугу на яку він хотів би записатись. Після підтвердження вибору користувачу буде запропоновано варіанти оплати послуги.

Рис. 2.2 – Меню вибора майстра

На рисунку 2.2 представлено приклад меню вибора майстра. На ньому можна дізнатись ім’я майстра, де він працює, його посаду і побачити коротко що він про себе написав. Меню представлено в круговому вигляді, тобто при виборі майстра буде запускатися анімація переключення на іншого майстра.

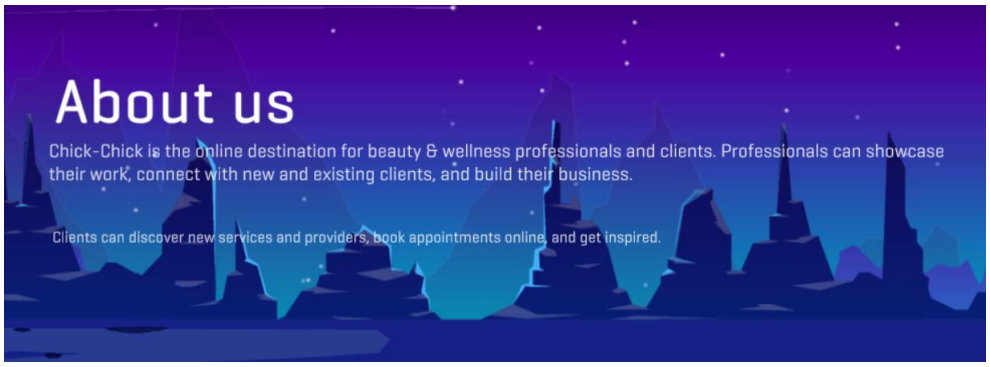

Рис. 2.3 – Сторінка «Про додаток»

Для зручності використання вебсайту було додано панель навігації, яка є індивідуальною для кожної ролі. Тобто якщо власник салону має на своїй панелі доступ як до фінансової частини, так і до даних відносно продуктивності майстрів, то майстер має на своїй панелі лише доступ до свого профілю і може переглянути свої успіхи.

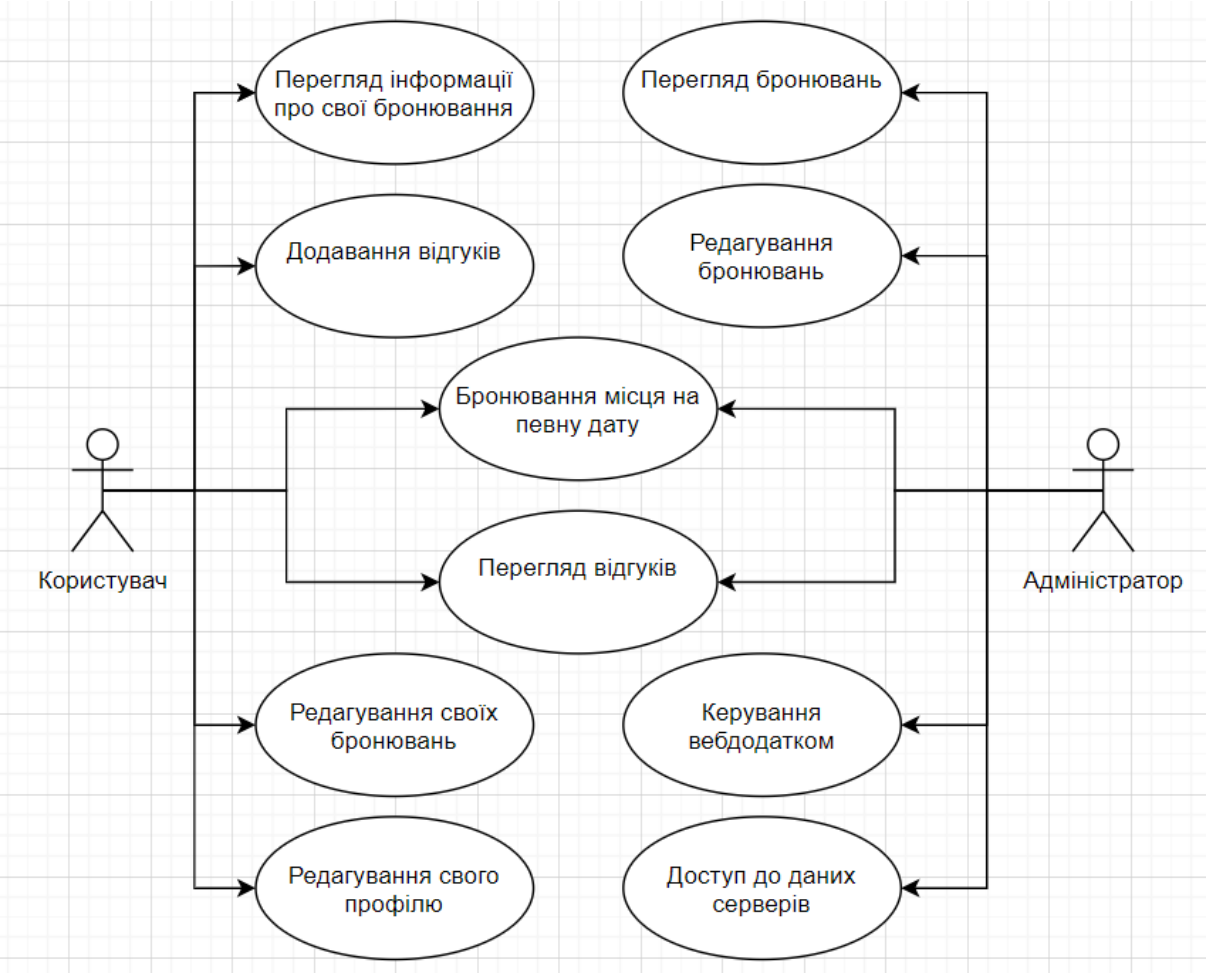

Рис. 2.4 – Діаграма варіантів використання вебсайту користувачем та адмністратором

Як можна побачити на рисунку 2.4 користувач має п’ять основних можливостей серед яких бронювання місця, перегляд інформації про свої бронювання, редагування своїх бронювань, редагування свого профілю, також немало важливим є те, що користувач може залишати відгуки про майстрів та салони і можеш проводити онлайн оплату послуг напряму через вебдодаток.

У ролі адміністратора є найбільше прав, як можна побачити на рисунку 2.4 адмністратор додатку має повний доступ до бронювань клієнтів, він може як бронювати сам, так і по запиту клієнта редагувати або видаляти бронювання. Адміністратор також може видавати ролі іншим користувачам якщо це потрібно. Для покращення стабільності додатку адміністратор має повний доступ до даних серверів, щоб у разі виявлення неполадки швидко полагодити її.

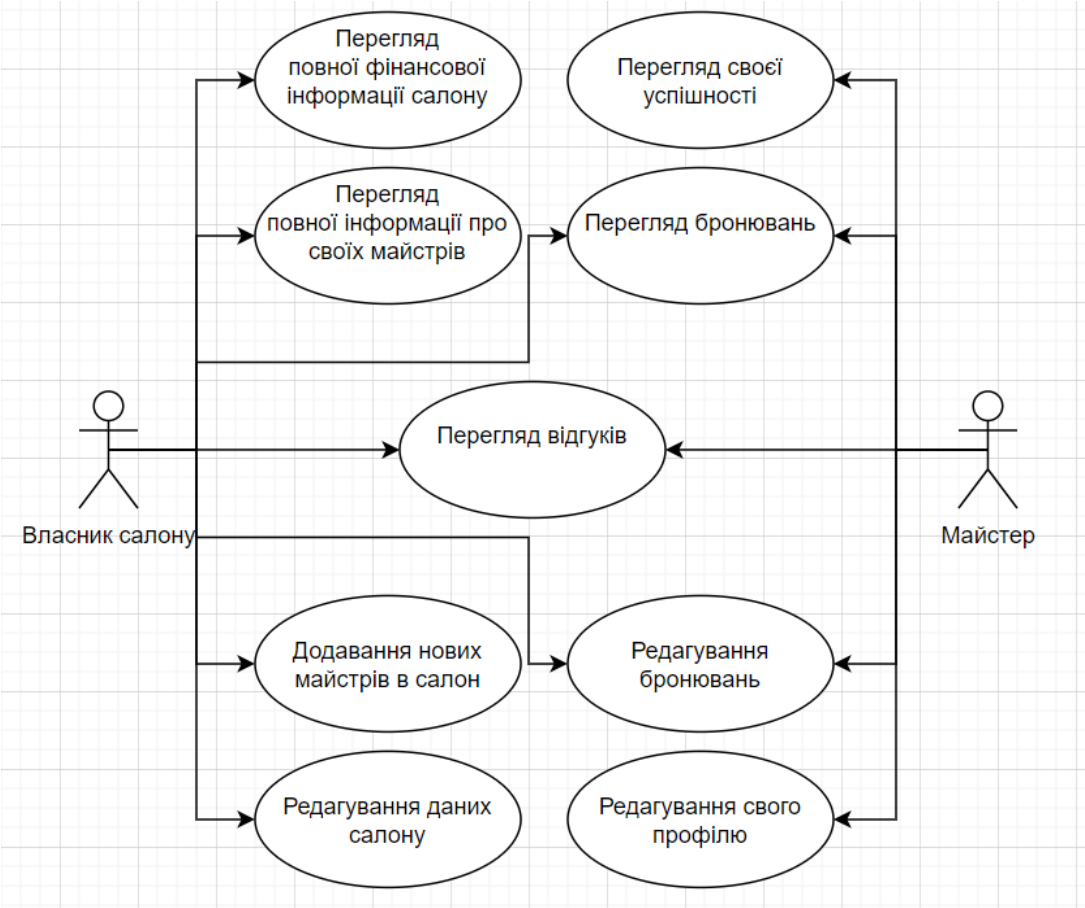

Рис. 2.5 – Діаграма варіантів використання вебсайту власником салону і майтром

На рисунку 2.5 представлені можливості ролі майстра і власника салону, а саме майстер може переглядати на які дати у нього є бронювання, редагувати заброньовані дати та послуги, майстер може редагувати свій профіль, в якому може відкрити доступ до відгуків про себе, щоб будь-який клієнт мав змогу дізнатись як інші клієнти оцінюють його. Також у майстра є можливість переглянути свою успішність де він може дізнатись про свої успіхи за весь час.

Як було зазначено раніше роль власника салону має більше прав ніж дві попередні ролі, посилаючись на рисунок 2.5 можна побачити, що власник салону має повну інформацію про успішність своїх майстрів, також він може додавати нових майстрів до сторінки свого салону. Власник може редагувати сторінку свого салону по власному бажанню, також для цієї ролі був доданих функціонал перегляду повної фінансової інформації салону. Для покращення комунікації з клієнтами було додано можливість перегляду відгуків про салон.

Також можливості ролей можна переглянути в таблиці 2.1.

Таблиця 2.1 – Варіанти використання вебдодатку.

Роль	Можливості
Користувач	Редагування свого профілю
	Бронювання місця на певну дату
	Перегляд інформації про свої бронювання
	Додавання відгуків
	Редагування своїх бронювань
Адміністратор	Бронювання місць
	Керування функціями вебдодатку
	Редагування бронювань
	Редагування профілів користувачів
	Доступ до даних серверів
Власник салону	Перегляд повної інформації про своїх майстрів
	Додавання нових майстрів
	Редагування даних салону
	Перегляд фінансової інформації салону
	Перегляд відгуків про салон
Майстер	Перегляд бронювань
	Редагування бронювань
	Перегляд відгуків
	Редагування свого профілю
	Перегляд своєї успішності

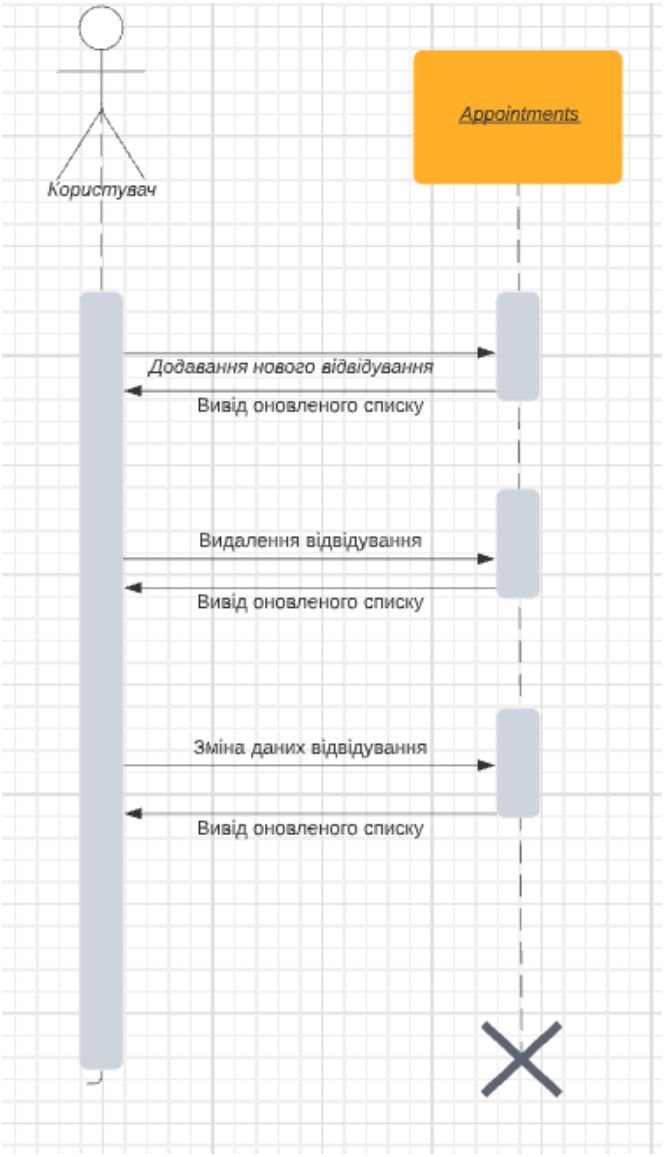

Рис. 2.6 – Діаграма послідовності

На рисунку 2.6 зображено діаграму послідовності для ролі користувача. Як можна побачити з рисунку 2.6 спочатку користувач робить запит до класу Appointments, після отримання запиту система видає користувачу форму, яку він повинен буде заповнити коректними даними, для успішного додавання даних у базу даних.

Після створення бронювання користувач має змогу змінювати або видаляти його за потреби. Також користувач може переглянути список своїх відвідувань зробивши відповідний запит до класу Appointments.

2.2 Проектування архітектури

Маючи відомості про систему з попередніх розділів, можна приступити до проектування архітектури додатку, а саме визначенням, які класи будуть присутні в системі.

Отже маючи представлення про те яким має бути додаток можна виділити певні класи які будуть потрібні системі, а саме клас Users в якому буде описано ролі користувачів та їх рівні допуску, клас Cities де будуть представлені міста з якими працює додаток, клас Companies де буде інформація про компанії, відповідно до компанії з класу Services будуть братись дані про послуги які надає компанія, інформація про майстрів надходитиме з класу Specialists і немало важливим є клас Appointments в якому мають бути дані про те де в кого і на яку годину було заброньоване місце.

Визначившись з класами вебдодатку можна буде приступити до побудови діаграми класів. Для побудови діаграми буде використано мову візуалізації додатків UML. На сьогоднішній день ця мова є основною для представлення додатків. В цій мові є певний набір графічних інструментів, який допомагає розробникам у передачі своїх ідей, та концепцій.

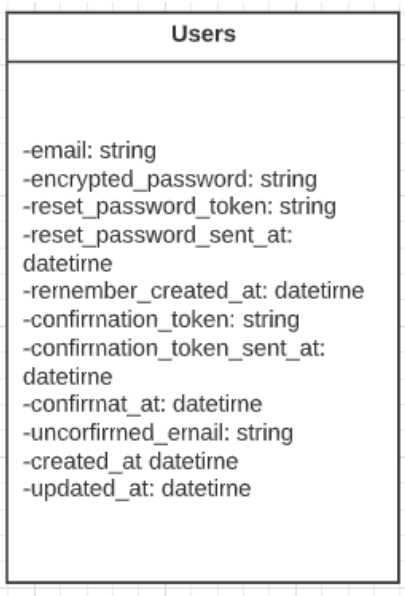

Рис. 2.7 – Клас Users

Клас Users, як видно з рисунку 2.7 відповідає за збереження інформації про користувачів. Також в цей клас додано функції авторизації користувачів, підтвердження електронної пошти.

Для успішної реєстрації користувача потрібно ввести необхідні дані які зображені на таблиці 2.2. Також ці дані можуть бути використаними для авторизації, або зміни паролю, якщо користувач забув його.

Таблиця 2.2 – Дані для успішної реєстрації користувача

Найменування поля	Тип даних	Опис даних
Email	string	Електронна пошта користувача
Password	string	Пароль аккаунта користувача
Confirmation token	string	Токен підтвердження пошти

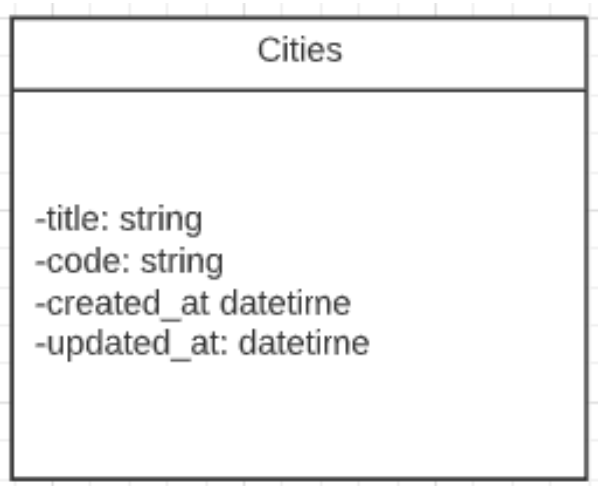

Рис. 2.8 – Клас Cities

На рисунку 2.8 зображено клас Cities, який відповідає за збереження інформації про міста з якими працює додаток, а саме назву міста та його код.

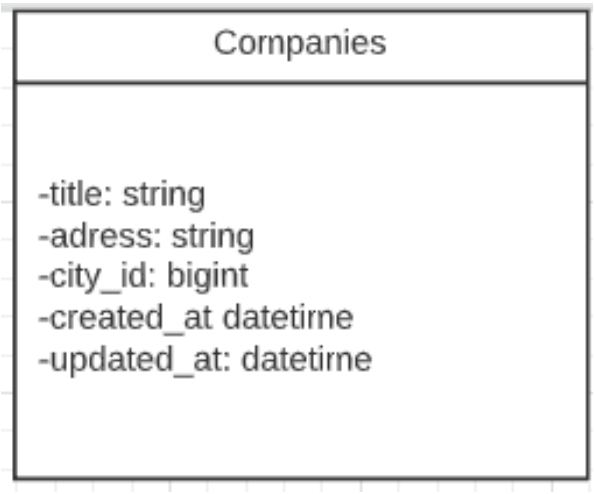

Рис. 2.9 – Клас Companies

Як можна побачити на рисунку 2.9 клас Companies зберігає інформацію про компанії які належать певному місту з урахуванням їх назви та адреси. Для створення салону користувачем ролі власник салону потрібно ввести такі дані, як назва, адреса, та ід міста в якому знаходиться салон.

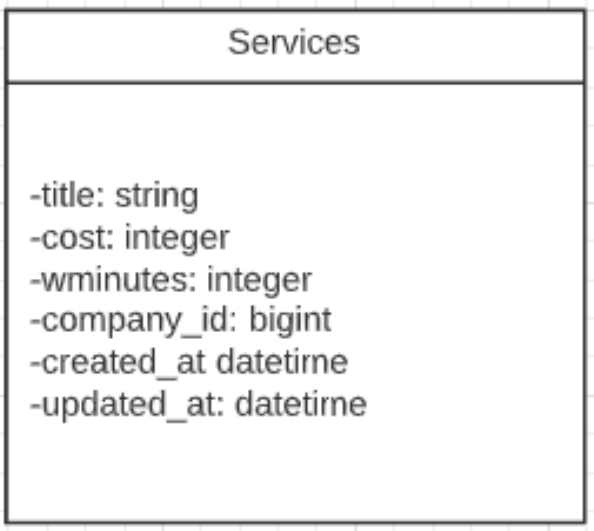

Рис. 2.10 – Клас Services

Відповідно до рисунку 2.10 клас Services зберігає у собі інформацію про послуги які надає салон з зазначеною вартістю та приблизним часом який може зайняти та чи інша послуга.

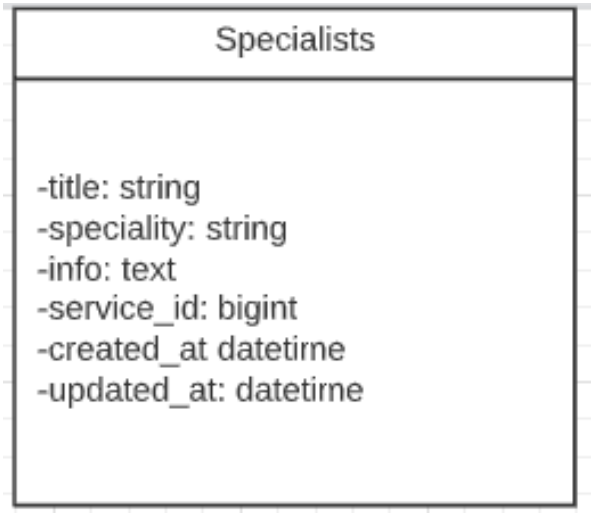

Рис. 2.11 – Клас Specialists

Зсилаючись на рисунок 2.11 клас Specialists зберігає інформацію про майстрів, і показує користувачу майстрів які відповідають його вимогам.

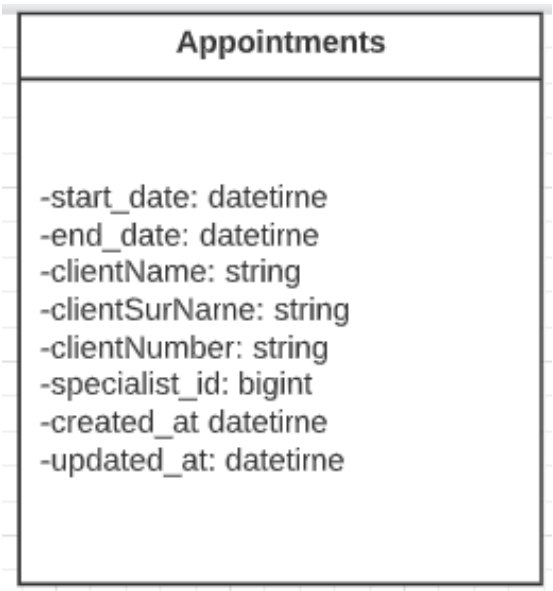

Рис. 2.12 – Клас Appointments

Як видно з рисунка 2.12 клас Appointments зберігає інформацію про бронювання, а саме ім’я клієнта, його номер телефону, інформацію про майстра, дату та час проведення послуги.

Описавши основні класи додатку можна перейти до побудови діаграми класів, до якої можна звертатись надалі.

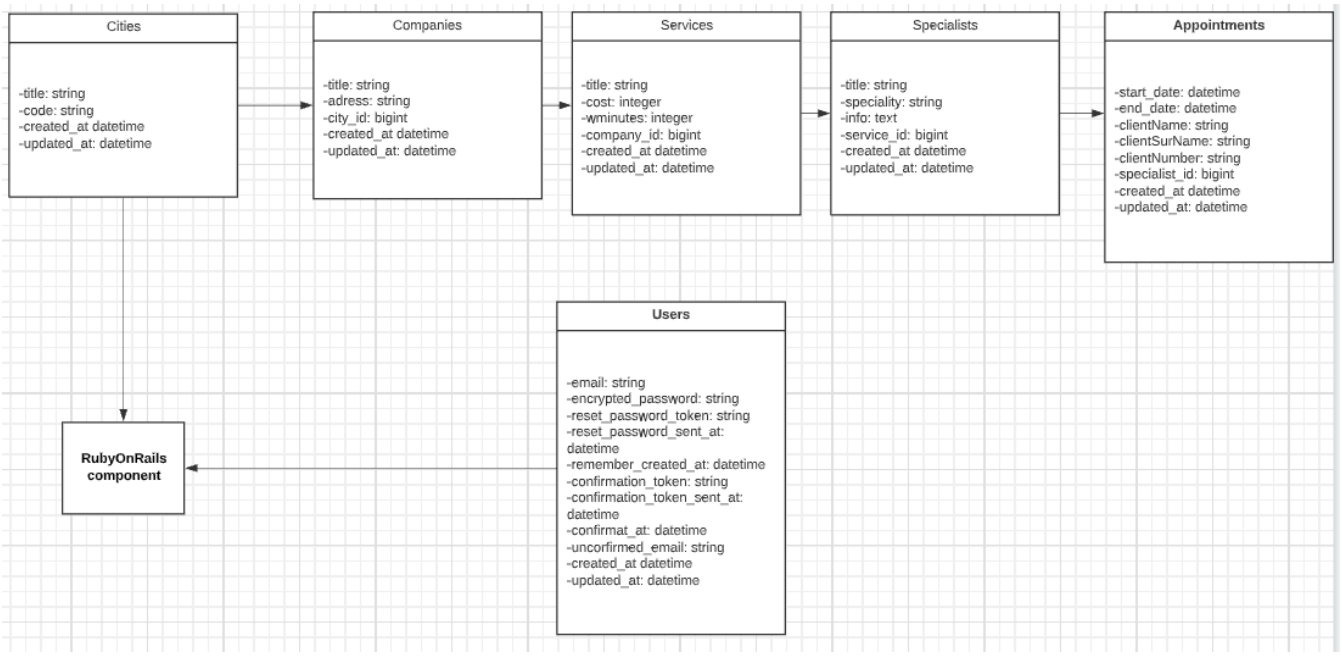

Рис. 2.13 – Діаграма класів вебдодатку «Chick-Chick»

На даній діаграмі зображено відношення між основними класами додатку, а також компоненту RubyOnRails[8] з якого наслідуються класи. Зсилаючись на рисунок 2.13 можна побачити, що більшість класів успадковуються з класу Cities.

РОЗДІЛ 3. КОНСТРУЮВАННЯ ТА ТЕСТУВАННЯ ВЕБСАЙТУ «CHICK-CHICK»

3.1 Реалізація ключових класів

Так як для розробки додатку було обрано RubyOnRails[9] потрібно для початку реалізувати ключові класи які буде використовувати вебдодаток, а саме клас Users, Cities, Companies, Services, Specialists, Appointments. Мною було використано технології RubyOnRails тому для початку мені потрібно було реалізувати авторизацію користувачів.

```
<h2>Sign up</h2>
<p class="alert"><%= alert %></p>
<%= form_for(resource, as: resource_name, url: registration_path(resource_name)) do |f| %>
  <%= render "users/shared/error_messages", resource: resource %>

  <div class="field">
    <%= f.label :email %><br />
    <%= f.email_field :email, autofocus: true, autocomplete: "email" %>
  </div>

  <div class="field">
    <%= f.label :password %>
    <% if @minimum_password_length %>
    <em>(<%= @minimum_password_length %> characters minimum)</em>
    <% end %><br />
    <%= f.password_field :password, autocomplete: "new-password" %>
  </div>

  <div class="field">
    <%= f.label :password_confirmation %><br />
    <%= f.password_field :password_confirmation, autocomplete: "new-password" %>
  </div>

  <div class="actions">
    <%= f.submit "Sign up" %>
  </div>
<% end %>
```

Рис. 3.1 – Код реєстрації нового користувача

Як можна побачити з рисунку 3.1 мною було представлено меню реєстрації користувача за допомогою вказування ним своєї електронної адреси та паролю, після чого йому потрібно буде підтверджувати свою пошту за допомогою функції

Action Mailer[10]. За допомогою Mailer стає доступна полегшена реалізація таких функцій як: підтвердження електронної адреси, зміна паролю, зміна адреси, відновленння профілю користувача.

```
<h2>Resend confirmation instructions</h2>

<%= form_for(resource, as: resource_name, url: confirmation_path(resource_name), html: { method: :post }) do |f| %>
  <%= render "users/shared/error_messages", resource: resource %>

  <div class="field">
    <%= f.label :email %><br />
    <%= f.email_field :email, autofocus: true, autocomplete: "email", value: (resource.pending_reconfirmation? ?
     resource.unconfirmed_email : resource.email) %>
  </div>

  <div class="actions">
    <%= f.submit "Resend confirmation instructions" %>
  </div>
<% end %>

<%= render "users/shared/links" %>
```

Рис. 3.2 – Код Action Mailer для підтвердження пошти

Як можна побачити Mailer використовує для підтвердження пошти вже заготоване повідомлення і пересилає його на пошту користувача від імені додатку. Також він використовує ці дані якщо користувач захоче змінити пошту або пароль.

Після реалізації функцій для авторизації і реєстрації користувачів мною було прийнято рішення перейти до реалізації логіки інших класів. Для початку було обрано клас Cities.

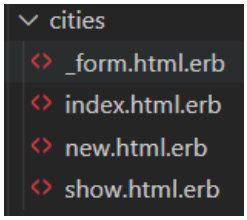

Рис. 3.3 – Функціонал класу Cities розподілений по файлах

Для класу Cities як і для інших було прийнято рішення розподілити певний функціонал на декілька файлів. Так у цьому випадку весь функціонал було розподілено на чотири файли, index.html.rb є основним файлом з якого потім

йдуть виклики для інших файлів з функціоналом показу доступних міст та додаванням нового міста.[11]

```
<h1>Listing cities</h1>
<%= link_to 'New City', new_city_path %>
<table>
  <tr>
    <th>Title</th>
    <th>Code</th>
    <th colspan="3"></th>
  </tr>

  <% @cities.each do |city| %>
    <tr>
      <td><%= city.title %></td>
      <td><%= city.code %></td>
      <td>| <%= link_to 'Show', city_path(city) %> |</td>
      <td><%= link_to 'Destroy', city_path(city),
                    method: :delete,
                    data: { confirm: 'Are you sure?' } %></td>
    </tr>
  <% end %>
</table>
```

Рис. 3.4 – Код файлу index.html.rb

Як можна побачити з рисунку 3.4 файл index.html.rb є стартовою сторінкою класу Cities. На цій сторінці представлено всі доступні міста в яких є барбершопи, також з цієї сторінки можна перейти на сторінку створення міста через файл new.html.rb.

Використовуючи цей файл адміністратор має змогу додати нове доступне місто в якому можуть бути перукарні. Як можна побачити з рисунку 3.5 для додавання нового міста необхідно ввести такі дані, як назву міста та його код. Після підтвердження інформації місто буде додано в перелік доступних.

```
<%= form_with model: @city, local: true do |form| %>

  <% if @city.errors.any? %>
    <div id="error_explanation">
      <h2>
        <%= pluralize(@city.errors.count, "error") %> prohibited
        this city from being saved:
      </h2>
      <ul>
        <% @city.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <p>
    <%= form.label :title %><br>
    <%= form.text_field :title %>
  </p>

  <p>
    <%= form.label :code %><br>
    <%= form.text_area :code %>
  </p>

  <p>
    <%= form.submit %>
  </p>

<% end %>
```

Рис. 3.5 – Код файлу new.html.rb

Опираючись на інформацію з діаграми класів можна зрозуміти те, що в моєму додатку класи наслідують один іншого, тому потрібно описувати їх по мірі наслідувань. Тому наступним описаним класом буде клас Companies оскільки він наслідує клас Cities.[12]

```
<p>
  <strong>Title:</strong>
  <%= @city.title %>
</p>

<p>
  <strong>Code:</strong>
  <%= @city.code %>
</p>

<h2>Companies list</h2>

<%= render @city.companies %>

<h2>Add a company:</h2>

<%= render 'companies/form' %>

<%= link_to 'Back', cities_path %>
```

Рис. 3.6 – Код файлу companies_index_html.rb

В цьому файлі представлений список салонів, які співпрацюють з додатком та находяться в вибраному місті. Як можна побачити з коду на рисунку 3.6 для виклику списку салонів використовується код міста по якому проходить виклик. Також за бажання на цій же сторінці власником салону може бути додано його салон за формою представленою на рисунку 3.7.

```
<%= form_with(model: [ @city, @city.companies.build ], local: true) do |form| %>
  <p>
    <%= form.label :title %><br>
    <%= form.text_field :title %>
  </p>
  <p>
    <%= form.label :about %><br>
    <%= form.text_field :about %>
  </p>
  <p>
    <%= form.label :adress %><br>
    <%= form.text_area :adress %>
  </p>
  <p>
    <%= form.submit %>
  </p>
<% end %>
```

Рис. 3.7 – Код для створення нового салону в додатку

В файлі для створення нового салону є такі змінні як назва, про салон, та адреса салону. Тобто все створення нового салону для підтвердженого власника є дуже простим, йому потрібно вказати ці дані після чого йому буде надано повний доступ до його сторінки де він зможе вказувати послуги їх ціну і час виконання, налаштовувати візуал, додавати нову інформацію про салон, додавати нових майстрів, та переглядати фінансову успішність салону за певний проміжок часу.

Для нормального функціоналу додатку потрібно буде також створити клас Services де будуть відображатись послуги які пропонує салон, клас Specialists в якому можна буде побачити майстрів, які можуть надати цю послугу, їх профілі з відгуками які їм залишили інші користувачі, та клас Appointments де буде відображено вже зайняті години в того чи іншого майстра на конкретну послугу.

```
<p>
  <strong>Company name:</strong>
  <tr>
      <td><%= @company.title %></td>
    </tr>
</p>

<p>
  <strong>Adress:</strong>
  <%= @company.adress %>
</p>

<h2>Services list</h2>

<%= render @company.services %>

<h2>Add a service:</h2>

<%= render 'services/form' %>

<%= link_to 'Edit', edit_city_company_path(@city, @company) %> |
<%= link_to 'Back', city_path(@city) %>
```

Рис. 3.8 – Код файлу services_index_html.rb

Оскільки класи наслідують один іншого кожне наступне вкладення буде відповіта певному ід, який передається через базу даних. Тому сторінка послуг ніколи не може бути одинаковою для всіх салонів, вона може бути схожа тільки у пипадку коли власники салонів між собою домовились про таке.[9]

```
<%= form_with(model: [ @city, @company, @city.companies.build.services.build ], local: true) do |form| %>
  <p>
    <%= form.label :title %><br>
    <%= form.text_field :title %>
  </p>
  <p>
    <%= form.label :cost %><br>
    <%= form.text_area :cost %>
  </p>
  <p>
    <%= form.label :wminutes %><br>
    <%= form.text_area :wminutes %>
  </p>
  <p>
    <%= form.submit %>
  </p>
<% end %>
```

Рис. 3.9 – Код для додавання нової послуги салону

З представленого рисунка 3.9 можна можна побачити, як у салон можна додавати послуги. Не мало важливим є те, що тут вперше з’являється поле wminutes, саме в цьому полі далі будуть передаватися дані для класу Appointments в якому ці дані будуть використані для бронювання місця на певний час.

Наступним у списку реалізації став клас Specialists в якому буде представлено майстрів які можуть виконати вибрані послуги, також в ньому реалізовується представлення профілю майстра з відгуками про нього.

```
<p>
  <strong>Service name:</strong>
  <tr>
      <td><%= @service.title %></td>
    </tr>
</p>

<p>
  <strong>Cost:</strong>
  <%= @service.cost %>
</p>
<p>
  <strong>Time:</strong>
  <%= @service.wminutes %>
</p>
<h2>Specialists list</h2>

<%= render @service.specialists %>
```

Рис. 3.10 – Код меню представлення списку майстрів

Саме в цей клас власники салонів можуть додавати своїх майстрів, які потім будуть представлені у вигляді списку по рейтингу. А самі майстри можуть оформляти свій профіль який в подальшому буде представлений через посилання у списку майстрів. Майстрів в списку будуть відображати ім’я, фото профілю, коротко про себе, та рейтинг.[10]

Крайнім в списку реалізації став клас Appointments. В ньому користувач буде обирати дату та час на яку він би хотів записатись. У цьому класі будуть представлені всі дані про бронювання, а саме салон, майстер, послуга, вартість, та проміжок часу який це може зайняти. Також клієнт може видалить бронювання за бажанням.

```
<p>
  <strong>Occupated working hours: <%= appointment.start_date %></strong>
</p>
<p>
  <strong>Client Name: <%= appointment.clientName %></strong>
</p>
<p>
  <strong>Client Surname: <%= appointment.clientSurName %> </strong>
</p>
<p>
  <strong>Client Number: <%= appointment.clientNumber %></strong>
</p>


<p>
  <%= link_to 'Remove appointment', city_company_service_specialist_appointment_path(@city, @company, @service, @specialist, appointment),
                method: :delete,
                data: { confirm: 'Are you sure?' } %>
</p>
```

Рис. 3.11 – код реалізації класу Appointments

Також важливу рольв проекті відіграє база даних MongoDB. Оскільки ця база даних не потребує прямого заповнення для неї було використано автозаповонення через схему. Схему для автозаповнення можна переглянути на рисунку 3.12.

```
create_table "appointments", force: :cascade do |t|
  t.datetime "start_date"
  t.datetime "end_date"
  t.string "clientName", default: ""
  t.string "clientSurName", default: ""
  t.string "clientNumber", default: ""
  t.bigint "specialist_id", null: false
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
  t.index ["specialist_id"], name: "index_appointments_on_specialist_id"
end

create_table "cities", force: :cascade do |t|
  t.string "title", default: ""
  t.string "code", default: ""
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
end

create_table "companies", force: :cascade do |t|
  t.string "title", default: ""
  t.text "adress", default: ""
  t.bigint "city_id", null: false
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
  t.index ["city_id"], name: "index_companies_on_city_id"
end
```

Рис. 3.12 – Схема автозаповнення бази даних MongoDB

Також приклад роботи схеми можна переглянути на рисунку 3.13.

```
# Examples:
#
cities = City.create([
    { title: 'Київ', code: 'ky' },
    { title: 'Хмельницький', code: 'hm' }
])
```

Рис. 3.13 – Приклад роботи бази даних MongoDB

3.2 Тестування та результат розробки

На етапі тестування мною було отримано результати співпраці класів представлені в вигляди рисунків 3.14-3.17. Моєю головною цілю під час тестування було отримати робочий проект для подальшого його представлення під час захисту роботи.

Отже після успішньої реєстрації нас буде чекати меню вибору міста представлене на рисунку 3.14, на цьому рисунку представлено приклад тестового макету вебсторінки з підгруженими даними з бази даних MongoDB. Також тут ми можемо спостерігати кнопки функції, які було раніше мною описані, а саме додавання нового міста, та показ компаній вже створених для певного міста.

Рис. 3.14 – Приклад тестової роботи сторінки вибір міста

Після вибору міста в якому користувач захоче отримати певні послуги йому буде представлена сторінка з вибором доступних салонів для цього міста. На якій буде представлено список салонів відсортований по рейтингу та найближчими по геолокації до користувача, відображено буде назву салону, його адресу, та його рейтинг серед користувачів додатку.

На рисунку 3.14 можна також побачити що для подальшої взаємодії користувача з салоном є кнопка Show Company, яка буде висвітлювати доступні послугу для цього салону та його ціни на певні послугу.

Рис. 3.15 – Приклад тестової роботи сторінки вибору салону

Наступним етапом в бронювані місця є вибір послуги, яку захоче отримати клієнт. Доступні послуги для кожного салону предаставлені по прикладу який зображений на рисунку 3.16.

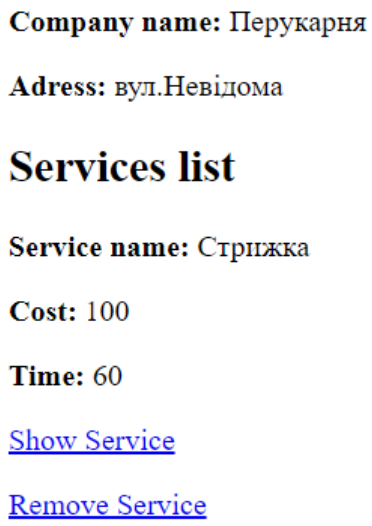

Рис. 3.16 – Приклад списку доступних послуг від салону

Оскільки ми переглядаємо приклади від ролі власника салону, то видно, що в нього є доступ до зміни даних послуг, додавання нових та видалення неактуальних. Після вибору послуги нам буде представлено список майтрів, які можуть надати цю послугу. Також на наступному етапі буде представлено список загруженості майстрів вже створеними бронюваннями.

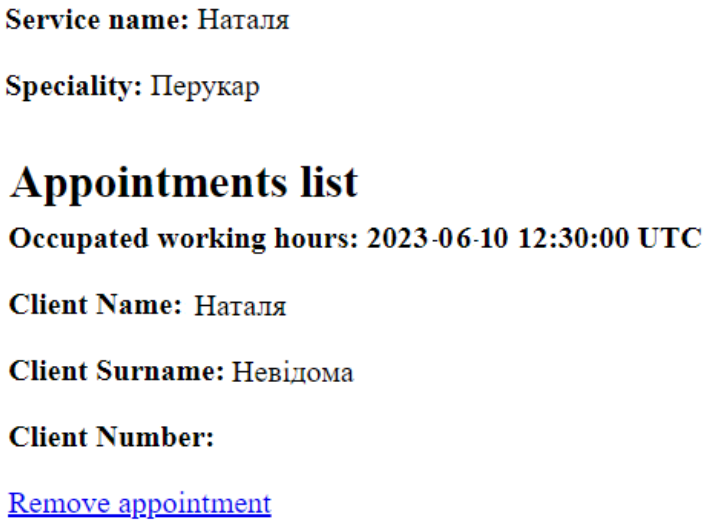

Рис. 3.17 – Приклад вигляду сторінки з бронюваннями майстрів

Як можна побачити на рисунку 3.17 представлено тестовий варіант вигляду сторінки бронювання для кожного майстра. На ній представлено дати та час зайнятості майстра, також на ній є дані про клієнта який зробив бронювання, а саме ім’я, фамілія та номер телефону за яким можна з ним зв’язатись.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при кровотечах.

Якщо поруч з вами знаходиться людина з кровотечою, а фахівці ще не наспіли, необхідно надати потерпілому першу допомогу. Для цього важливо визначити характер кровотечі, тому що при різних травмах заходи долікарської допомоги будуть відрізнятися.

Кровотеча[13] — це витікання крові із кровоносних судин при порушенні їхньої цілісності. Відповідно є різні види кровотеч, при кожній з яких надаються різні типи долікарської допомоги. Розрізняють капілярну, венозну і артеріальну кровотечі.

Капілярна кровотеча виникає навіть при незначному пораненні. Оскільки кров по капілярах тече повільно і під невеликим тиском, то капілярні кровотечі не призводять до значної втрати крові і легко зупиняються. Пошкодженні капіляри швидко закриваються тромбом, що утворюється при зсіданні крові. Першою допомогою при капілярних кровотечах є знезараження перекисом водню або йодною настойкою місце поранення і накладання на нього чистої пов'язки.

Венозна кровотеча виникає при ушкодженні поверхневих ран. У цьому разі, особливо коли ушкоджені великі вени, зсідання крові не здатне швидко зупинити кровотечу. За короткий час можуть бути значні крововтрати. При венозних кровотечах кров витікає рівномірно і має темний колір. Щоб зупинити венозну кровотечу, досить накласти компресійну пов'язку. Перед цим рану знезаражують, а для зменшення кровотечі тимчасово притискають пошкоджену судину.

Артеріальна кровотеча особливо небезпечна для життя. Вона буває у вигляді пульсуючого струменя подібно до фонтану. Кров має яскраво-червоний колір. У цьому разі треба діяти негайно. Перш за все вище місця поранення треба швидко притиснути пальцями ушкоджену судину в точках, де прощупується пульс і накласти джут.

Щоб не пошкодити нерви і шкіру, його накладають поверх одягу, хустки, або іншої м'якої тканини. Коли джгута немає, можна скористатися ременем або зробити закрутку з будь-якого шнура, тканини.

Для цього між тканиною і тілом вставляють міцну палицю і закручують тканину до зупинки кровотечі. Потім палицю прибинтовують до тіла. Джгут залишають на кінцівках не більше як на 1,5-2 години, а в холодну пору року на 1 годину, інакше настане омертвіння тканини. Щоб цього не сталося до джгуту приколюють записку з точним зазначенням часу його накладання. Якщо потерпілого за цей час не можливо доставити до лікарні 1,5-2 год. послаблюють джгут на 1-2 хв., якщо кровотеча продовжується, джгут затягують.

Також існує зовнішня і внутрішня кровотечі. При зовнішній кровотечі кров витікає через рану в шкірі та у видимих слизових оболонках або з порожнин.

При внутрішній кровотечі кров виливається в тканини й органи тіла, це називається крововиливом. Швидка значна втрата крові є дуже небезпечною, оскільки супроводжується зниженням кров'яного тиску, порушенням кровопостачання мозку, серця і всіх інших органів. Тому вона буває причиною загибелі людей, яких ще можна було б врятувати, надавши своєчасно першу допомогу.

Внутрішні кровотечі в черевну порожнечу, порожнину грудей, черепа є надзвичайно небезпечні. Встановити наявність внутрішньої кровотечі можна тільки за зовнішнім виглядом людини. Вона стає блідою, виступає холодний піт, пульс частішає і слабне. У такому разі треба негайно викликати швидку допомогу. До її прибуття потерпілого кладуть або напівсидять і не рухають з місця. До ймовірного місця кровотечі (живота, грудей, голови) прикладають холодний компрес (мішечок із льодом або снігом, грілку або пляшку з холодною водою).[14]

Отже знаючи які є типи кровотеч і як з ними боротися стає можливим надати необхідну долікарську допомогу при різних типах поранень тіла. Ця інформація є важливою в наш час, адже це може трапитися з кожним, тому важливо знати як надавати правильну долікарську допомогу вчасно.

4.2 Заходи, що покращують умови праці оператора ПК.

Діяльність більшості працівників сучасних професій у виробничій сфері пов’язана з використанням комп’ютерної техніки. Комп’ютер для сучасної людини є такою ж технічною необхідністю, як телевізор або холодильник. [15]

Побутові прилади ми використовуємо не задумуючись про їх шкідливість або нешкідливість, усвідомлюючи лише переваги їх наявності. Щодо комп’ютерів, то існує багато інформації як про їх безпечність, так і шкідливість.

Осіб, які працюють з комп’ютерами, поділяють на групи:

- розробники програм (інженери-програмісти) мають справу переважно з відео терміналами, їх робота характеризується інтенсивною розумовою творчою працею з підвищеним напруженням зору, концентрацією уваги на фоні нервово-емоційного напруження, вимушеною робочою позою, загальною гіподинамією, періодичним навантаженням на кисті рук;
- оператори електронно-обчислювальних машин виконують роботу, пов’язану з обліком інформації, одержаної з візуального дисплейного терміналу за попереднім запитом, або тієї, що самостійно надходить з нього, яка супроводжується перервами різної тривалості, пов’язана з виконанням іншої роботи і характеризується як робота з напруженням зору, невеликими фізичними зусиллями, нервовим напруженням середнього ступеня та виконується у довільному темпі;
- оператори комп’ютерного набору займаються одноманітною за характером роботою з документацією та клавіатурою, під час якої нечасто та ненадовго переключають погляд на екран дисплея, вводять дані з високою швидкістю. Робота характеризується як фізична праця з підвищеним навантаженням на кисті рук на фоні загальної гіподинамії з напруженням зору (фіксація зору переважно на документи), нервово-емоційним напруженням.

Для забезпечення комфортної роботи оператора ПК необхідно звернути увагу на такі основні вимоги:

- стан виробничих приміщень;
- характеристики та стан техніки (ПК, монітори, принтери, наявність заземлення тощо);
- організація робочого місця;
- дотримання режимів праці та відпочинку;
- моніторинг стану здоров'я працівника (стан органів зору, показники захворювань з тимчасовою втратою працездатності, ін.);
- профілактичні заходи.

Важливим елементом комфортної роботи тиша. Нормативними значеннями еквівалентного рівня звуку є:

- 50 дБА для програмістів;
- 65 дБА для операторів у залах оброблення інформації;
- 75 дБА для операторів у приміщеннях, де розташовані гучні агрегати.

За результатами проведених гігієнічних досліджень, еквівалентні рівні шуму на робочих місцях працівників офісних приміщень знаходяться у межах 48-59 дБА еквівалентно. Як правило, перевищення рівня шуму на робочих місцях офісних працівників є наслідком телефонних розмов співробітників (у приміщеннях з великою кількістю робочих місць).

Також для комфортної роботи мікроклімат у приміщенні повинен відповідати стандартам, адже саме температура та вологість повітря впливають на загальне самопочуття, стан слизових оболонок очей, верхніх дихальних шляхів та шкіри персоналу офісів. Стандартами мікроклімату для комфортної роботи є:

- У теплий період року температура повітря має бути у межах 22-25 °С, швидкість руху повітря — до 0,1 м/с, відносна вологість повітря — 40-60%.
- У холодний період року температура повітря може коливатися у межах 21-24 °С, швидкість руху повітря — до 0,1 м/с, вологість повітря — 40-60%.

Не менш важливою умовою комфортної роботи є правильний режим праці та відпочинку. Оскільки оператор персонального комп'ютера працює з монітором, сильніше за все втомлюються очі, тому напруженість його роботи прямо пропорційна з напруженістю очей, тривалістю зосередження уваги, яка може становити понад 75% часу. Саме тому очі найбільш страждають під час роботи з комп'ютером.

Велике значення при роботі за комп'ютером мають такі речі як: відстань до екрана, шрифт, розмір тексту на моніторі, наявність або відсутність мерехтіння, яскравість екрану, освітлення робочого місця, наявність перерв у роботі. Саме ігнорування таких простих на перший погляд речей у значній мірі призводить до погіршення зору та хвороб очей. [16]

Для збереження здоров'я працівників, запобігання професійним захворюванням і підтримки працездатності слід дотримуватись регламентованих перерв для відпочинку:

- для розробників програм — 15 хв. через кожну годину роботи за комп'ютером;
- для операторів ЕОМ — 15 хв. через кожні 2 год.;
- для операторів комп'ютерного набору — 10 хв. після кожної години роботи.

Для зниження нервово-емоційного напруження і втоми очей, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії доцільно деякі перерви використовувати для виконання комплексу вправ, наведених у додатку 7 до ДСанПіН 3.3.2.007-98.

Оскільки на напруженість очей впливає також неправильне положення при роботі працівникам слід дотримуватись правильної робочої пози, неправильна робоча поза шкідливо впливає на скелетно-м'язову систему. Для забезпечення правильної пози тіла при роботі у працівника має бути правильно організоване робоче місце.

Для правильної організації робочого місця можна звернутись до нормативного документа ДСТУ 8604:2015 в якому описано, що робоче місце працівника має відповідати таким умовам:

- робоче місце має бути організоване так, щоб світло падало зліва (для «правшів» і навпаки — для «лівшів»);
- при розміщенні декількох робочих місць в одному приміщенні мінімальна площа для одного робочого місця складає $6м^2$;
- екран монітора має бути розміщений на оптимальній відстані від очей користувача (60-70 см), але не ближче 50 см;
- екран має бути розташований для забезпечення комфортного зорового спостереження у вертикальній площині під кутом + 30° до нормальної лінії погляду працівника;
- поверхня клавіатури має бути з антистатичними властивостями;
- під час роботи необхідно робити перерви для розвантаження очей.

Отже дотримуючись цих правил можна уникнути небажаних професійних хвороб працівників, та покращити умові праці оператора персонального комп’ютера і уникнути небажаних недуг.

ВИСНОВКИ

В результаті виконаної роботи було спроектовано вебдодаток, в якому було представлено зручний, та зрозумілий в користувані інтерфейс. Цей додаток значно спрощує ведення бізнесу для власників салонів, та робить зручним стеження за роботою та успішністю салону.

Використання технологій RubyOnRails дозволило створити адаптивний інтерфейс з можливістю швидкого створення та маніпулювання данними додатку, що дозволило спростити більшість процесів.

Розробку додатку досить відчутно спростив Visual Studio Code. Дякуючи зрозумілому інтерфейсу, та внутрішнім додаткам розробку було спрощено. До прикладу можна привести вбудований інтерфейс взаємодії з Github на якому знаходився додаток. З допомогою цього додатку я мав змогу спостерігати за етапами роботи над проектом та за необхідності повертатись на старіші версії додатку.

Велику роль у створені додатку зіграла база даних MongoDB, адже з її допомогою можна було автоматично заповнювати дані в базі даних без тотального контролю.

Одною з основний задач проекту було створення системи, якою можна буде керувати без використання значних людських ресурсів навіть при постійному рості користувачів. При аналізі виконаної роботи можна сказати, що мені вдалось це реалізувати.

Мною було подано варіанти використання системи для різних ролей. Було створено діаграму класів для більшого розуміння елементів додатку. Також розроблено систему авторизації користувачів використовуючи технології фреймворку RubyOnRails.

В розділі «Безпека життєдіяльності, основи хорони праці» висвітлено, як надавати долікарську допомогу при кровотечах, та способи покращення умов праці оператора персонального комп’ютера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Технічна документація по RubyOnRails [Електронний ресурс] – Режим доступу до ресурсу: [https://rubyonrails.org/].
2. HTML and CSS: Design and Build Websites by Jon Duckett. 2011. 490с.
3. Visual Studio Code by Bruce Johnson. 2019. 192с.
4. Github [Електронний ресурс] – Режим доступу до ресурсу: [https://github.com/].
5. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage by Eoin Brazil & Shannon Bradshaw. 2019. 511с.
6. Проект сайту Chick-Chick [Електронний ресурс] – Режим доступу до ресурсу: [https://github.com/BTROVI4/Chick_Ckick].
7. Етапи створення сайту [Електронний ресурс] – Режим доступу до ресурсу: [https://recommerce.com.ua/etapi-stvorennya-saitu]
8. Programming Ruby by Noel Rappin, with Dave Thomas. 2004. 824с.
9. Agile Web Development with Rails 4-7 by Sam Ruby. 2023. 476с.
10. Action Mailer [Електронний ресурс] – Режим доступу до ресурсу: [https://guides.rubyonrails.org/action_mailer_basics.html]
11. The basic stages for building a webpage [Електронний ресурс] – Режим доступу до ресурсу: [https://www.designcontest.com/blog/the-basic-stages-forbuilding-a-webpage/]
12. How to Code Your Own Website [Електронний ресурс] – Режим доступу до ресурсу: [https://www.quackit.com/make-your-own-website/]
13. "Безпека життєдіяльності" Желібо Є.П., Зацарний В.В., 2018. 344 с.
14. Книга Безпека життєдіяльності. Навчальний посібник Павло Атаманчук, 2017. 276 с.
15. Грибан В.Г. Охорона праці: навч. посібник / В.Г. Грибан, О.В. Негодченко. – Київ: Центр учбової літератури, 2018. 280 с.

16. Жидецький В.Ц. Охорона праці користувачів комп’ютерів: навчальний посібник / В.Ц. Жидецький. – Львів: Афіша, 2016. 176 с.
17. Примак Т.О. Маркетингові комунікації в системі управління підприємством. — К.: Експерт, 2020. — 384 с.
18. Все про веб-сайти [Електронний ресурс] – Режим доступу до ресурсу: [http://repair.lviv.ua/stvorennya-sajtiv/osnovni-etapi-stvorennya-sajtu/].
19. Довідник по HTML [Електронний ресурс] – Режим доступу до ресурсу: [https://htmlbook.online/page/html]
20. Довідник по CSS [Електронний ресурс] – Режим доступу до ресурсу: [https://htmlbook.online/page/css]
21. Babel Documentation [Електронний ресурс] – Режим доступу до ресурсу: [https://babeljs.io/]
22. Rake Documentation[Електронний ресурс] – Режим доступу до ресурсу: [https://www.rubyguides.com/2019/02/ruby-rake/]
23. Петрик М.Р. Проєктування програмного забезпечення на основі аналізу вимог та інструментальних засобів розробки IBM Rational Software Architect (від Вимог до коду) Науково-методичний посібник. Тернопіль: Вид-во ТНТУ ім. Івана Пулюя.-2022.- 560с.
24. Методичні вказівки до написання звіту В.Земцква; Ю.Поліщук, канд. фіз.-мат. Наук; Р. Санченко, канд. техн. наук; Л.Шрамко; А.Ямчук (науковий керівник) [Електронний ресурс] – Режим доступу до ресурсу: [https://science.kname.edu.ua/images/dok/derzhstandart_3008_2015.pdf].

ДОДАТКИ

Додаток А

Лістинги контролерів класів

Лістинг 1 - контролеру Cities

```
class CitiesController < ApplicationController
  before_action :authenticate_user!

  layout 'application'

  def index
    @cities = City.all
  end

  def show
    @city = City.find(params[:id])
  end

  def new
    @city = City.new
    authorize @city
  end

  def create
    @city = City.new(city_params)
    authorize @city

    if @city.save
      redirect_to @city
    else
      render 'new'
    end
  end

  def update
    @city = City.find(params[:id])
    authorize @city

    if @city.update(city_params)
      redirect_to @city
    else
      render 'edit'
    end
  end

  def destroy
    @city = City.find(params[:id])
```

```
    authorize @city

    @city.destroy

    redirect_to cities_path
  end

  private
    def city_params
      params.require(:city).permit(:title, :code)
    end

end
```

Лістинг 2 - контролеру Companies

```
class CompaniesController < ApplicationController
  before_action :authenticate_user!

  layout 'application'

  def index
    @city = City.find(params[:city_id])
    @company = @city.companies.all
  end

  def show
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:id])
  end

  def edit
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:id])
    authorize @company
  end

  def create
    @city = City.find(params[:city_id])
    @company = @city.companies.new(company_params)
    authorize @company
    @company.save
    redirect_to city_path(@city)
  end

  def update
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:id])
    authorize @company

    if @company.update(company_params)
      redirect_to city_path(@city)
    else
```

```
      render 'edit'
    end
  end

  def destroy
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:id])
    authorize @company
    @company.destroy
    redirect_to city_path(@city)
  end

  private
    def company_params
      params.require(:company).permit(:title, :adress, :user_id)
    end
end
```

Лістинг 3 - контролеру Services

```
class ServicesController < ApplicationController
  before_action :authenticate_user!

  layout 'application'

  def index
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.all
  end

  def show
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:id])

  end

  def edit
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:id])
    authorize @service
  end

  def create
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.new(service_params)
    authorize @service
    @service.save
    redirect_to city_company_path(@city, @company)
  end
```

```
  def update
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:id])
    authorize @service

    if @service.update(service_params)
      redirect_to city_company_path(@city, @company)
    else
      render 'edit'
    end
  end

  def destroy
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:id])
    authorize @service
    @service.destroy
    redirect_to city_company_path(@city, @company)
  end

  private
    def service_params
      params.require(:service).permit(:title, :cost, :wminutes)
    end
end
```

Лістинг 4 - контролеру Specialists

```
class SpecialistsController < ApplicationController
  before_action :authenticate_user!

  layout 'application'

  def index
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.all
  end

  def show
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:id])
  end

  def edit
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
```

```
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:id])
    authorize @specialist
  end

  def create
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.new(specialist_params)
    authorize @specialist
    @specialist.save
    redirect_to city_company_service_path(@city, @company, @service)
  end

  def update
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:id])
    authorize @specialist

    if @specialist.update(specialist_params)
      redirect_to city_company_service_path(@city, @company,
@service)
    else
      render 'edit'
    end
  end

  def destroy
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:id])
    authorize @specialist
    @specialist.destroy
    redirect_to city_company_service_path(@city, @company, @service)
  end

  private
    def specialist_params
      params.require(:specialist).permit(:title, :speciality, :info)
    end
end
```

Лістинг 5 - контролеру Appointments

```
class AppointmentsController < ApplicationController
  before_action :set_city_company_service_specialist_appointment,
only: [:show, :update, :destroy]

  protect_from_forgery except: :create
```

```
  def index
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:specialist_id])
    start_date = DateTime.parse(params[:start_date])
    @appointments = @specialist.appointments.where(start_date:
start_date.all_day)
    render json: @appointments
  end

  def show
    render json: @appointment
  end

  def create
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:specialist_id])
    @appointment = @specialist.appointments.new(appointment_params)
    if @appointment.save
      render json: @appointment
    else
      render json: @appointment.errors, status:
:unprocessable_entity
    end
  end

  def destroy
    @city = City.find(params[:city_id])
    @company = @city.companies.find(params[:company_id])
    @service = @company.services.find(params[:service_id])
    @specialist = @service.specialists.find(params[:specialist_id])
    @appointment = @specialist.appointments.find(params[:id])
    @appointment.destroy
  end

  private
    def appointment_params
      params.require(:appointment).permit(:start_date, :clientName,
:clientSurName, :clientNumber, :end_date)
    end
    def set_city_company_service_specialist_appoiment
      @city = City.find(params[:city_id])
      @company = @city.companies.find(params[:company_id])
      @service = @company.services.find(params[:service_id])
      @specialist =
@service.specialists.find(params[:specialist_id])
      @appointment = @specialist.appointments.find(params[:id])
    end
end
```