Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікацій та електронних систем
(назва відділення)

Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної роботи

бакалавра
(освітній ступінь)

на тему: Розробка вебсайту центру "Space sink"

Виконав: студент VI курсу, групи КІб-602

Спеціальності **123 Комп'ютерна інженерія**
(шифр і назва, спеціальності)

Віталій ПЙОНТКОВСЬКИЙ
(ім'я та прізвище)

Керівник Володимир ШТОКАЛО
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2024

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерної інженерії
Освітній ступінь бакалавр
Освітньо-професійна програма: Комп'ютерна інженерія
Спеціальність: 123 Комп'ютерна інженерія
Галузь знань: 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"08" травня 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Пйонтковському Віталію Богдановичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Розробка вебсайту центру "Space sink"

керівник роботи Штокало Володимир
(прізвище, ім'я, по батькові)

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університетут імені Івана Пулюя» від 07.05.2024 р №4/9-224.

2. Строк подання студентом роботи: 21 червня 2024 року.

3. Вихідні дані до роботи: мова програмування JavaScript, технічне завдання на розробку вебсайту, стандарти IEEE 29148-2018, IEEE 29119

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Охорона праці, техніка безпеки та екологічні вимоги.

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

- схема структурна головної сторінки сайту;
- схема структурна компонент фронтенду сайту;
- схема структурна взаємодії ;
- блок схема ;
- текст програми;
- таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Оксана РЕДЬКВА заст. директора з НВР		
Охорона праці, техніка безпеки та екологічні вимоги	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	08.05	
2	Збір і узагальнення інформації	20.05	
3	Написання першого розділу	24.05	
4	Розробка технічного та робочого проекту	28.05	
5	Написання спеціального розділу	3.06	
6	Розрахунок економічної частини	5.06	
7	Написання розділу охорони праці	7.06	
8	Виконання графічної частини	10.06	
9	Оформлення проекту	14.06	
10	Погодження нормоконтролю	17.06	
11	Попередній захист роботи	21.06	
12	Захист кваліфікаційної роботи	26.06	

7. Дата видачі завдання: 08 травня 2024 року

Студент _____ Віталій ПЙОНТКОВСЬКИЙ
(підпис) (ім'я та прізвище)

Керівник роботи _____ Володимир ШТОКАЛО
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

Пйонтковський В.Б. Розробка вебсайту центру “Space sink” м. Тернопіль: кваліфікаційна робота на здобуття освітнього ступеня бакалавр, за спеціальністю 123 Комп’ютерна інженерія. Тернопіль: ВСП «ТФК ТНТУ», 2024. 91с.

Розроблений вебсайт “Space sink”, для детейлінг студії Space sink, на основі передових Web-технологій з метою оптимізації діяльності та ефективного представлення фірми в мережі Інтернет . Для розробки клієнтської частини веб-сайту було використано JavaScript бібліотеку React, яка дозволяє створювати динамічні та інтерактивні інтерфейси. Навігація по сайту реалізована за допомогою компонента Navbar. Серверна частина програми побудована на основі платформи Node.js з використанням фреймворку Express, що є важливим кроком у створенні потужних та масштабованих веб-додатків.

Ключові слова: клієнт, сервер, контролер, база даних, вебсайт, моделі даних, клас, авторизація, токен, користувач, безпека, запит.

ANNOTATION

Piontkovskyi V.B. Development of the website of the "Space sink" center in Ternopil: qualifying work for obtaining a bachelor's degree, specialty 123 Computer engineering. Ternopil: VSP "TFC TNTU", 2024. 73p.

The "Space sink" website was developed for the detailing of the Space sink studio, based on advanced Web technologies in order to optimize activities and effectively present the company on the Internet. To develop the client part of the website, the React JavaScript library was used, which allows you to create dynamic and interactive interfaces. Site navigation is implemented using the Navbar component. The server part of the application is built on the basis of the Node.js platform using the Express framework, which is an important step in creating powerful and scalable web applications.

Keywords: client, server, controller, database, website, data models, class, authorization, token, user, security, request.

ЗМІСТ

ВСТУП

У сучасну цифрову епоху, де розвиток інформаційних технологій набуває стрімкого характеру, важлива роль належить Інтернету. Використання комп'ютерів для зберігання та обміну різноманітної інформації стає не лише зручним, але й стратегічно важливим аспектом управління даними. Сучасні компанії не можуть ігнорувати переваги присутності в мережі Інтернет, оскільки це стає ключовим елементом підвищення продуктивності, обміну інформацією та засобом представлення своїх послуг та продуктів перед глобальною аудиторією.

Власний вебсайт дає компанії безмежні можливості для розміщення контенту та спілкування зі своїми клієнтами. Через доступність сайту 24/7, власники бізнесу можуть стежити за потребами та запитами своїх клієнтів у будь-який час, надаючи швидку відповідь і забезпечуючи високий рівень обслуговування. Крім того, відкритий доступ до інформації про компанію та її продукти дозволяє залучати нових клієнтів та розширювати сферу впливу.

Метою цієї кваліфікаційної роботи є не лише аналіз сучасних технологій, але й їх практичне застосування у розробці вебсайту для центру "Space Sink". Відповідно, дослідження включає в себе аналіз поточних тенденцій у сфері веб-розробки, вибір оптимальних інструментів та технологій, а також створення функціонального та естетичного вебсайту, який задовольнить потреби як клієнтів, так і власника бізнесу.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Веб-сайт, або інтернет-сайт, представляє собою важливий засіб комунікації та розповсюдження інформації в сучасному світі. Він є складовою частиною, який включає в себе різноманітні елементи, такі як текст, графіка, мультимедіа, програмний код і мережеві протоколи.

Одним із ключових компонентів веб-сайту є його дизайн. Дизайн визначає візуальний стиль та організацію елементів на сторінці. Він включає в себе вибір кольорів, шрифтів, композицію, розміщення кнопок та інших елементів інтерфейсу. Добре продуманий дизайн допомагає зробити сайт привабливим та зручним для користувачів.

Ще однією важливою складовою веб-сайту є його контент. Контент включає в себе текстову інформацію, зображення, відео, аудіо та інші медіа-ресурси. Якість контенту визначається його коректністю, актуальністю та цікавістю для аудиторії. Ефективний контент привертає увагу користувачів і стимулює їх до взаємодії з веб-сайтом.

Не менш важливою складовою веб-сайту є програмний код. Код відповідає за функціональність сайту, забезпечуючи його роботу та взаємодію з користувачем. Він може бути написаний мовами програмування, такими як HTML, CSS, JavaScript на frontend і PHP, Python, Ruby на backend. Через програмний код веб-сайт стає динамічним і інтерактивним, забезпечуючи користувачеві зручну та ефективну взаємодію з ресурсом.

Підсумовуючи, веб-сайт є складним інструментом, який виконує різноманітні завдання та забезпечує взаємодію з користувачем. Його успішна реалізація вимагає уваги до всіх його складових, від дизайну до програмного коду, з метою створення якісного та зручного для користувача веб-простору.

Популярність веб-сайтів у сучасному світі є надзвичайно високою через їхню універсальність і доступність. Вони використовуються для різних цілей, починаючи від особистих блогів та інтернет-магазинів, і закінчуючи корпоративними порталами та системами управління. Крім того, веб-сайти виступають як ефективний інструмент для забезпечення комунікації між бізнесом та його клієнтами, а також для залучення нових аудиторій та розширення географії обслуговування.

Ще однією важливою функцією веб-сайтів є їхній вплив на бренд і репутацію компанії. Сучасні споживачі активно використовують Інтернет для пошуку інформації про продукти та послуги перед придбанням, і веб-сайт стає першим джерелом інформації про компанію для більшості з них. Таким чином, якісний інформаційний веб-сайт забезпечує позитивне перше враження від компанії і може сприяти покращенню її репутації та збільшенню лояльності клієнтів.

Крім того, веб-сайт може стати ефективним інструментом для залучення нових клієнтів та збільшення обсягів продажів. Він надає можливість представити продукт чи послугу широкому колу аудиторії, здійснити ефективну рекламу та маркетингові кампанії. Багато компаній використовують веб-сайти для проведення акцій, розсилки новин та спеціальних пропозицій, що дозволяє привертати увагу та зацікавленість клієнтів і стимулювати їх до покупок.

Отже, веб-сайт в сучасному світі виконує широкий спектр функцій, від представлення інформації до залучення клієнтів і підтримки бренду. Його ефективне використання вимагає не лише якісного дизайну і змісту, але й розуміння потреб і очікувань цільової аудиторії та використання сучасних технологій та методів маркетингу.

Хостинг є ключовою складовою будь-якого веб-сайту, яка забезпечує його доступність та функціональність в мережі Інтернет. Це спеціально виділене місце на сервері, призначене для зберігання та обробки даних веб-

сайту. Щоб розмістити свій сайт в мережі, необхідно використовувати хостинг, де дані сайту будуть зберігатися, а запити від користувачів оброблятися і записуватися в базу даних. Існують різні способи створення хостингу: ви можете розмістити його на власному комп'ютері або орендувати його у провайдера.

Перш ніж розпочати розробку веб-сайту, важливо провести аналіз існуючих аналогів у вашій галузі. Наприклад, в рамках нашої дослідницької роботи ми проаналізуємо веб-сторінки компаній Imperial garage, Smart та Garage detailinglab.

Imperial garage - це перша студія автодетейлінгу у Львові та одна з перших у всій Україні. Вони розпочали свою діяльність у 2012 році. Студія спеціалізується на авторському детейлінгу, який відрізняється величезним досвідом та використанням професійного обладнання і косметики.

На головній сторінці веб-сайту Imperial garage міститься коротка інформація про студію та її діяльність. Також доступна можливість розрахувати вартість послуг і скористатися формою зворотнього зв'язку для зв'язку зі студією. На рисунку 1.1 представлена головна сторінка студії.

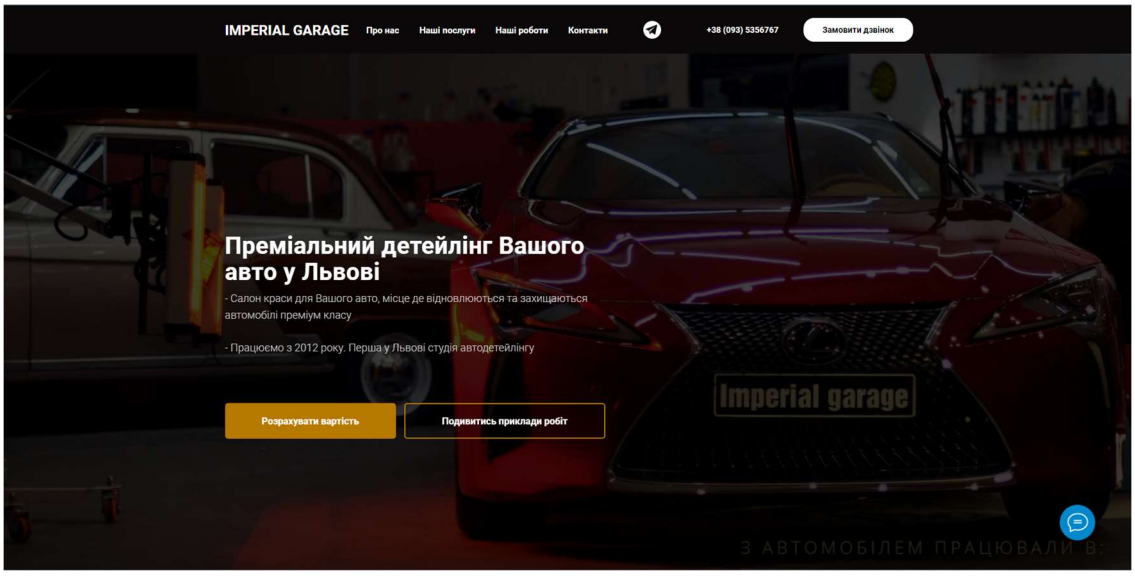


Рисунок 1.1 - Головна сторінка Imperial garage

На рисунку 1.2 представлено зображення розділу "Наші послуги", де можна побачити ілюстрації або фотографії, що візуалізують різноманітні види послуг.

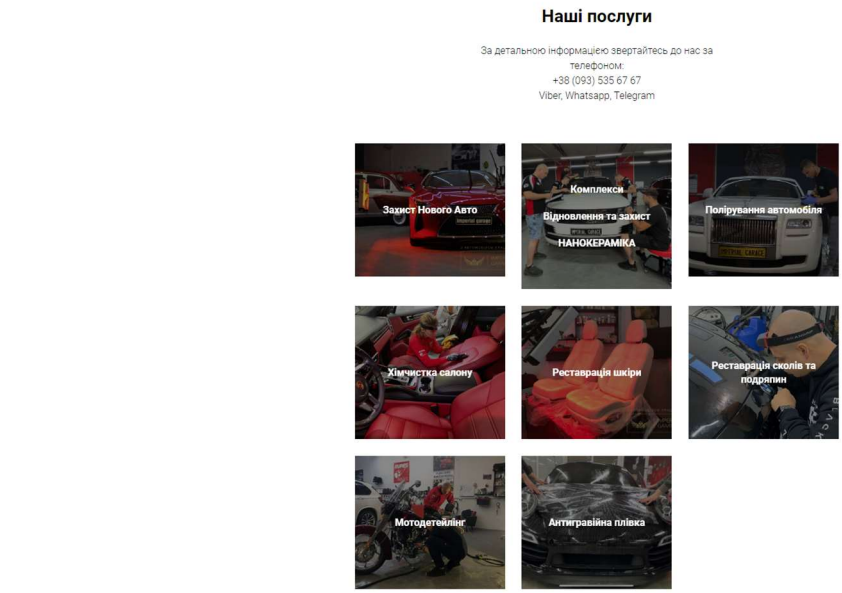


Рисунок 1.2 – Розділ послуг

На рисунку 1.3 представлено розділ "Контакти", де можна побачити інформацію про адресу студії, номер телефону для зв'язку, поштову адресу.

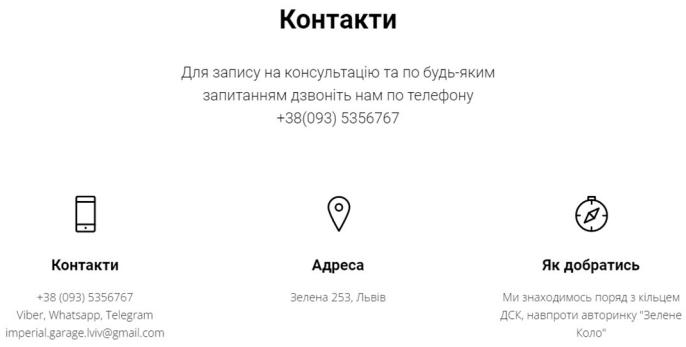


Рисунок 1.3 – Розділ контакти

Аналізуючи надану веб-сторінку, можна зробити висновки щодо її функціональності, доступності та навігації. У даному випадку, сторінка не

містить можливості авторизації користувачів та замовлення послуг, що може обмежувати її функціональні можливості.

Автокомплекс "Смарт" є одним з найсучасніших автокомплексів у столиці України. Він відзначається зручним розташуванням та надає колосальну економію часу для своїх клієнтів. Протягом п'яти років команда комплексу приділяє максимальну увагу кожному клієнту. На відміну від багатьох інших автомийок подібного типу, "Смарт" регулярно оновлює своє обладнання та використовує лише оригінальні комплектуючі. На рисунку 1.4 зображено логотип компанії.

Рисунок 1.4 – Логотип компанії

Головна сторінка веб-сайту автокомплексу складається з блоку, що демонструє діяльність комплексу, а також надає інформацію про його послуги.

На рисунку 1.5 зображено головну сторінку автокомплексу

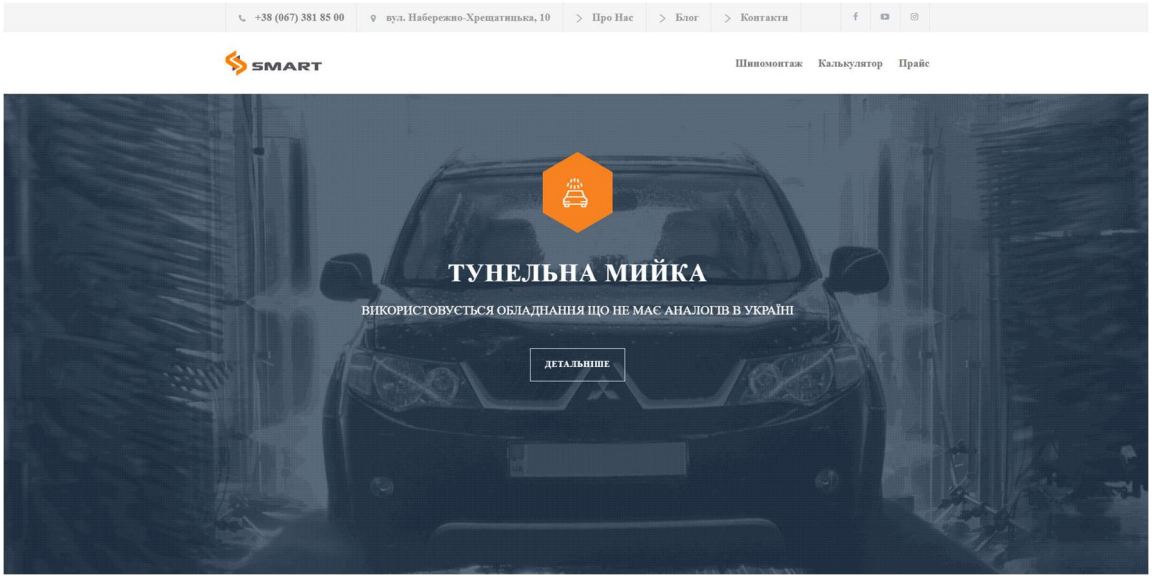


Рисунок 1.5 - Головна сторінка автокомплексу

На рисунку 1.6 зображено розділ "Послуги", де можна побачити ілюстрації або фотографії, що ілюструють різноманітні послуги комплексу.

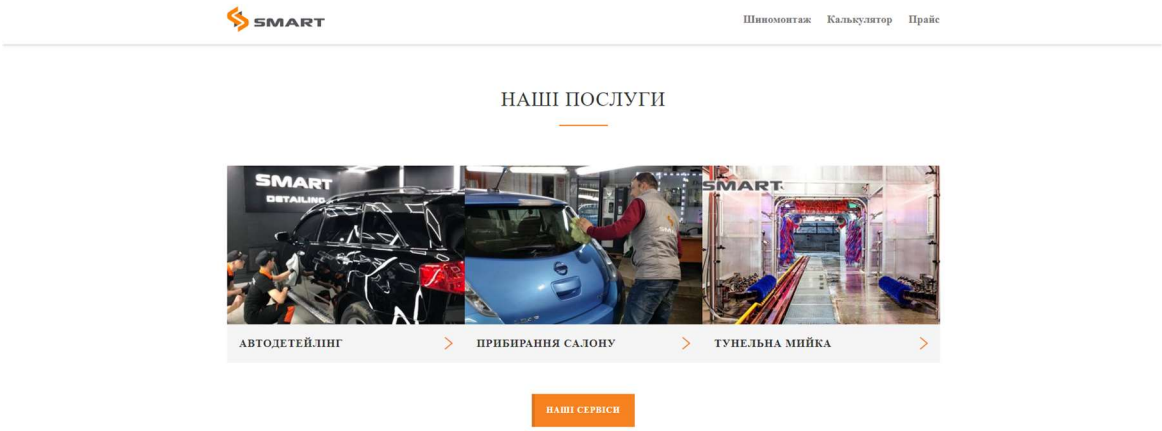


Рисунок 1.6 – Розділ послуг

На рисунку 1.7 зображено розділ "Контакти", де можна побачити контактну інформацію, таку як номер телефону, електронну адресу або адресу комплексу. Крім того, можливо, в цьому компоненті розміщені останні новини або оголошення про події, які цікавлять клієнтів.

Рисунок 1.7 – Розділ контакти

GARAGE DetailingLab - це детейлінг-студія преміум класу, яка розпочала свою діяльність у 2014 році. Протягом 8 років студія приділяла максимум зусиль для покращення якості та продуктивності виконання робіт, щоб задовольнити потреби найвибагливіших клієнтів.

На рисунку 1.8 зображено логотип студії, який відображає її стиль та ідентичність. Ця студія відома своєю високою якістю послуг та індивідуальним підходом до кожного клієнта.

Рисунок 1.8 – Логотип компанії

На головній сторінці веб-сайту GARAGE DetailingLab представлена інформація про детейлінг та його різновиди. Крім того, нижче наведена класифікація автомобілів, яка допомагає клієнтам краще зрозуміти, який вид детейлінгу підходить для їхнього автомобіля.

На рисунку 1.9 зображено головну сторінку GARAGE DetailingLab, де можна побачити ілюстрації або фотографії, що демонструють процес детейлінгу та його результати. Крім того, можливо, наявні розділи або блоки з інформацією про різновиди детейлінгу, його переваги та умови надання послуг.

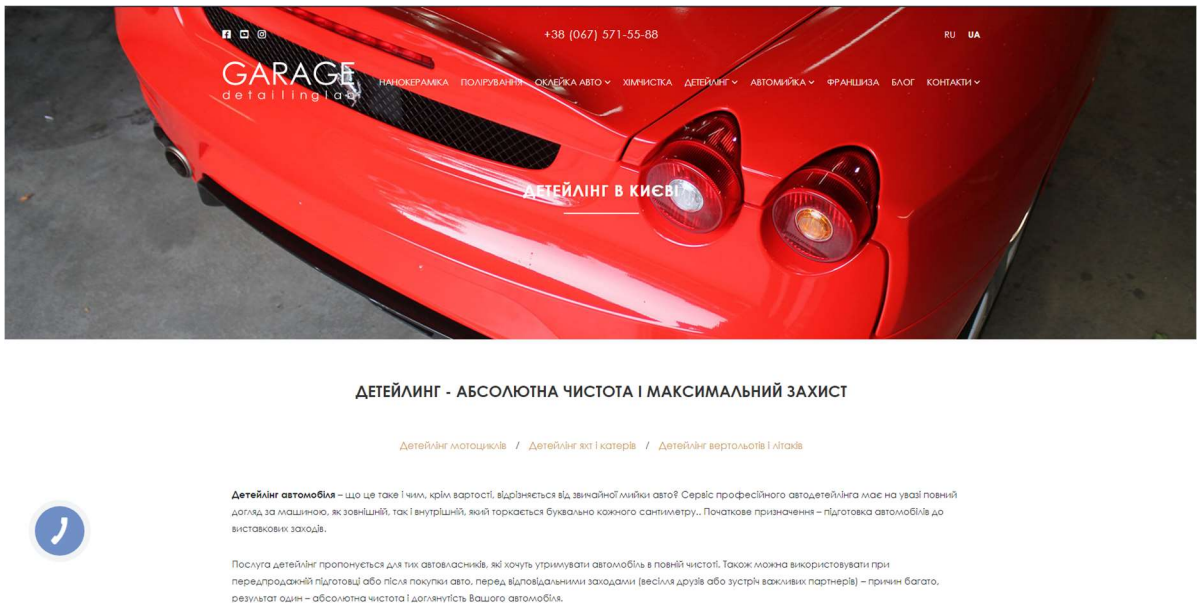


Рисунок 1.9 - Головна сторінка GARAGE DetailingLab

У розділі послуги можна переглянути послуги які надає студія автодетейлінгу клієнтам. На рисунку 1.10 зображено розділ послуг.

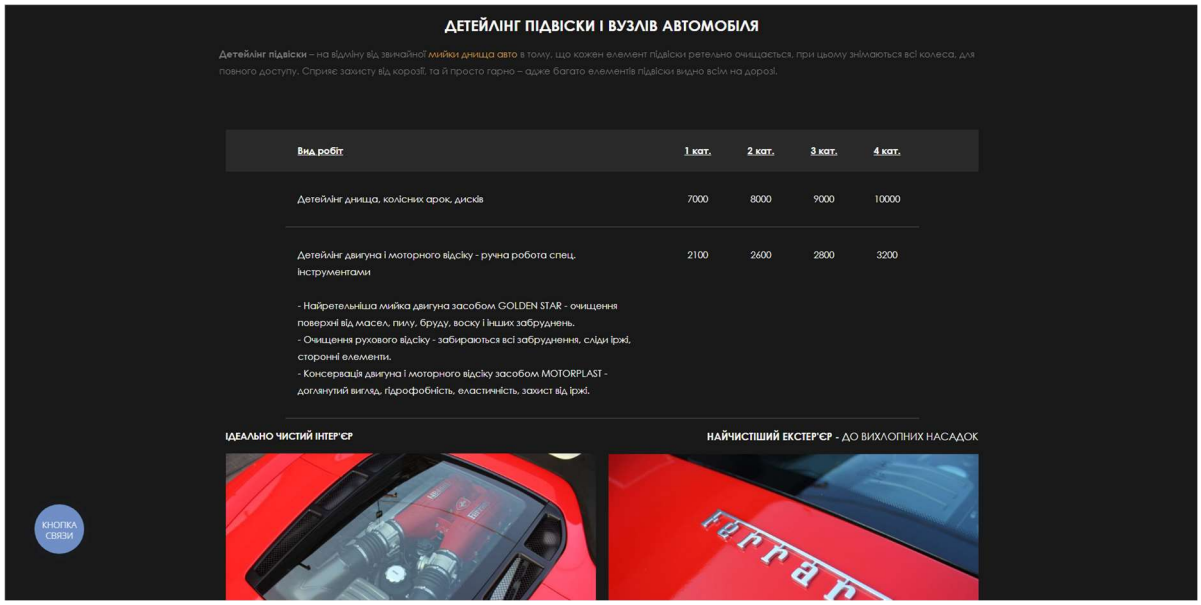


Рисунок 1.10 – Розділ послуг

Наприкінці сторінки розташований компонент, який містить контакти та місце розташування студії. На рисунку 1.11 зображено розділ контакти.

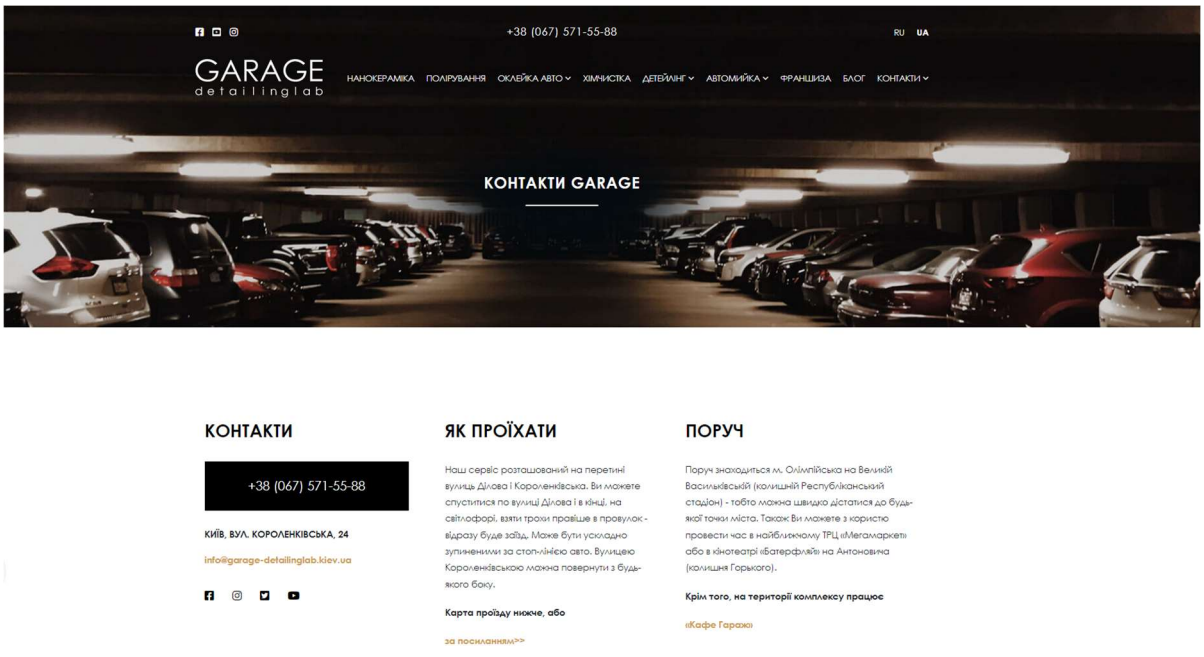


Рисунок 1.11 – Розділ контакти

Аналізуючи подані вище веб-сторінки, можна зробити висновки щодо кількох ключових аспектів, які впливають на користувацький досвід та ефективність сайту.

Перш за все, важливо звернути увагу на адаптацію ресурсів під різні пристрої та розміри екранів. Зручна навігація та швидкість завантаження є критичними для забезпечення позитивного враження користувачів.

Єдиний стиль та колірна гамма на всіх сторінках дозволяють створити єдиний образ бренду та підсилюють впізнаваність.

Що стосується контенту, важливо, щоб він був інформативним, дійсним та цікавим для майбутніх клієнтів. Наповнення ресурсу змістовним контентом допоможе привернути увагу аудиторії та створити позитивне враження про компанію.

Управління інформацією на сайті відіграє важливу роль, тому система вкладок та логічна організація контенту допоможуть користувачам легко знайти потрібну інформацію.

Нарешті, дизайн сайту має велике значення для формування позитивного враження про компанію. Підбір відповідного дизайну та врахування сучасних тенденцій дозволять створити привабливий та професійний веб-ресурс, який приверне увагу цільової аудиторії.

Крім того, зручна навігація та інтуїтивно зрозумілий інтерфейс сприятимуть довготривалому перебуванню користувачів на сайті. Ретельно продумана структура меню, чіткі посилання та логічне розміщення важливих розділів допоможуть уникнути плутанини та забезпечать приємний користувацький досвід. Це, в свою чергу, сприятиме підвищенню рівня задоволеності відвідувачів та зростанню їхньої лояльності до компанії.

Не можна забувати і про контент. Якісний, унікальний та регулярно оновлюваний контент є ключем до успішного сайту. Статті, блоги, новини та інші матеріали повинні бути цікавими, корисними та актуальними для цільової аудиторії.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Метою даної кваліфікаційної роботи є створення веб для центру "Space sink" на основі передових Web-технологій з метою оптимізації його діяльності та ефективного представлення в мережі Інтернет. Сучасний веб-сайт стає невід'ємною складовою успішного функціонування компаній, сприяючи полегшенню обміну інформацією та залученню нових клієнтів.

Представлення компанії в інтернеті має стратегічне значення для поліпшення робочих процесів та взаємодії з ринковим середовищем. Володіючи веб-сторінкою, компанія отримує можливість розмістити різноманітну інформацію та здійснювати свою діяльність в мережі Інтернет, доступну для користувачів у будь-який час.

Специфіка застосування розробленого веб-ресурсу стосується сфери діяльності детейлінгової студії. Цей продукт має на меті надати клієнтам можливість ознайомитися з різноманітними послугами, які пропонує центр "Space sink", що стосуються ретельного догляду за автомобілями.

1.2.2 Призначення розробки

Мета цієї розробки полягає в розв'язанні завдань, пов'язаних з представленням діяльності детейлінг-студії в глобальній мережі Інтернет та сприянням її подальшому розвитку. Веб-сайт спрямований на задоволення потреб клієнтів, надаючи їм можливість ознайомитись з послугами компанії та здійснити замовлення необхідних послуг онлайн. Це дозволяє забезпечити зручний та швидкий доступ до інформації про компанію, а також сприяти активізації взаємодії між студією та потенційними клієнтами через віртуальний простір Інтернету.

1.2.3 Вимоги до програмного забезпечення

Для правильної роботи веб-сайту необхідно мати наступне програмне забезпечення:

- Операційна система Windows версій XP, 7, 8, 10, 11;

- Браузер Google Chrome, Safari, Edge, Mozilla Firefox, оновлені до останньої версії.

Додатково, рекомендується мати актуальні версії веб-браузерів, оскільки це забезпечить оптимальну сумісність із веб-сайтом та забезпечить кращий досвід користувача. Також, важливо мати стабільне Інтернет-з'єднання для швидкого завантаження контенту сторінок та безперебійного використання функціоналу веб-сайту.

1.2.4 Техніко-економічні показники

Для техніко-економічного обґрунтування розробки та впровадження програмного продукту "Space sink" були визначені наступні показники:

- Операційна система: Для коректної роботи веб-сайту "Space sink" рекомендується використання операційних систем Windows (версій XP, 7, 8, 10, 11), які забезпечують стабільну роботу програми.

- Об’єм веб-сайту: Обсяг веб-сайту "Space sink" визначається розміром і кількістю веб-сторінок, графічних елементів, текстового та мультимедійного контенту, що використовується на сайті.

- Використані технології розробки: Для розробки веб-сайту "Space sink" були використані сучасні технології, такі як HTML, CSS, JavaScript, Node.js, React, Axios, Mobx, Express, CORS, bcrypt, та JWT. Кожна з цих технологій відповідає за певні аспекти функціональності та безпеки веб-сайту.

- Економічні показники: Для економічного аналізу розробки та впровадження програмного продукту "Space sink" необхідно враховувати витрати на розробку, тестування, маркетинг, обслуговування, а також очікувані прибутки та вигоди від використання веб-сайту. Також важливо провести порівняльний аналіз з подібними програмними продуктами на ринку та визначити конкурентні переваги "Space sink".

Ці показники допомагають зробити обґрунтоване рішення щодо розробки та впровадження програмного продукту, а також оцінити його ефективність та прибутковість.

1.2.5 Стадії та етапи розробки

Для успішної розробки веб-сайту дотримання наступних кроків є критично важливим:

- Визначення тематики та мети проекту: Це допомагає зрозуміти, які функції та функціональність повинен мати веб-сайт, а також які очікування має замовник.

- Розробка технічного завдання: Цей документ визначає всі вимоги до функціональності, дизайну та інших аспектів веб-сайту.

- Створення прототипу, макету та дизайну: Це етап, коли розробляються концепція та вигляд веб-сайту, включаючи його інтерфейс і навігацію.

- Написання текстів програми: Розробка вмісту важлива для відображення інформації та привернення уваги відвідувачів.

- Наповнення контентом: Це включає завантаження текстів, зображень, відео та іншого контенту на веб-сайт.

- Тестування веб-сайту: Цей етап дозволяє перевірити функціональність та виявити будь-які помилки або недоліки перед запуском веб-сайту.

- Здача готового проекту: Після успішного тестування веб-сайт готовий для запуску та використання.

Успішна розробка веб-сайту означає виконання всіх цих кроків у відведений час та бюджет, а також створення якісного продукту, який задовольнить потреби замовника.

На рисунку 1.12 представлені статистичні дані проектів в області веб-розробки, які були зібрані групою The Standish Group International, Inc. Ці дані відносяться до проектів в США і можуть служити додатковою інформацією для аналізу та планування веб-розробки.

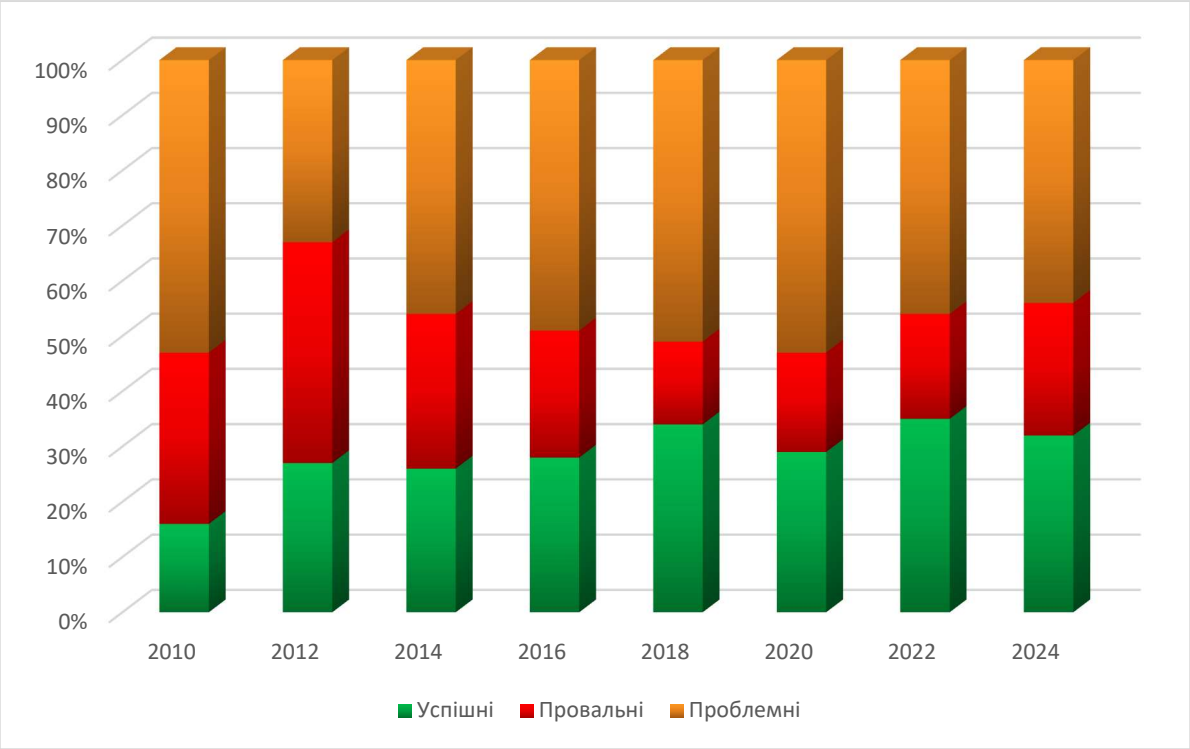


Рисунок 1.12 - Статистичні дані проектів в США

Відповідно до термінології The Standish Group, успішними вважаються проекти, які виконані без порушення часових та бюджетних обмежень, із функціональністю, що співпадає з планованою у специфікації продукту. Проблемні проекти, з свого боку, є тими, які завершені і функціонують коректно, але виникли перевищення часового графіка або бюджету, і в них реалізовані не всі заявлені в специфікації продуту функції. Нарешті, провальні проекти - це ті, які зупинені ще до завершення або ніколи не будуть завершені.

Аналізуючи діаграму, можна зробити висновок, що за 8 років існує прогрес у створенні успішніших проектів. Проте, частина проблемних проектів залишається великою, що може свідчити про необхідність подальшого удосконалення методів управління проектами та розробки програмного забезпечення.

1.2.6 Порядок контролю та прийому

Контроль процесу виконання кваліфікаційної роботи здійснюється керівником кваліфікаційної роботи, який відповідає за якість виконаної роботи. Цей етап охоплює перевірку всіх аспектів, таких як зручність навігації, працездатність та наявність всіх необхідних посилань, орфографія та пунктуація.

Після передачі виконавцем одного з функціональних модулів програми замовникові, останній має право тестувати модуль. Після тестування замовник зобов'язаний прийняти роботу по цьому етапу або в письмовому виді представити причину відмови. У випадку обґрунтованої відмови виконавець повинен доробити модуль.

Крім того, керівник кваліфікаційної роботи забезпечує відповідність виконаної роботи вимогам та очікуванням замовника. Він також відповідає за вчасну корекцію помилок та врахування зауважень замовника, щоб досягти оптимального результату. Цей етап контролю є важливим для забезпечення якості та успішного завершення кваліфікаційної роботи бакалавра.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЕКТУ

2.1 Постановка задачі на розробку програмного забезпечення

Розробка веб-сайту починається з чіткого визначення завдань і мети проекту. Після цього важливим етапом є формування користувацьких історій, які допомагають краще зрозуміти потреби та очікування майбутніх користувачів. За допомогою цих історій обирається оптимальна стратегія розробки, що відповідає вимогам та цілям проекту. Після узгодження стратегії переходимо до розробки дизайну веб-сайту, який має бути привабливим та зручним для користувачів, забезпечуючи їм комфортне взаємодію з сайтом.

Наступним кроком є реалізація програмної частини веб-сайту. На серверній стороні для забезпечення продуктивності та якості використовується платформа Node.js з фреймворком Express.js, що дозволяє ефективно взаємодіяти з базою даних та обробляти запити користувачів. Управління об'єктно-реляційними базами даних здійснюється за допомогою PostgreSQL. На клієнтській стороні веб-сайту використовується JavaScript бібліотека React, яка забезпечує швидку та зручну роботу з інтерфейсом користувача, а також може бути легко інтегрована в різноманітні додатки.

Загалом, успішна розробка веб-сайту передбачає комплексний підхід, починаючи від чіткого визначення завдань і закінчуючи ефективною реалізацією програмного забезпечення, що задовольняє потреби користувачів.

Після створення прототипу та дизайну веб-сайту настає час наповнення контентом. Це включає створення та редагування текстів, вибір та оптимізацію мультимедійного вмісту, такого як зображення та відео. Важливо, щоб контент відповідав меті та специфіці проекту, а також був цікавим та інформативним для цільової аудиторії.

Тестування веб-сайту є також критичним етапом в розробці. Під час тестування перевіряється коректність роботи всіх функцій, виявляються та

виправляються помилки та недоліки, а також перевіряється сумісність із різними браузерами та пристроями. Це дозволяє забезпечити оптимальну якість роботи веб-сайту перед його запуском.

Node.js став ключовим інструментом у веб-розробці, забезпечуючи розробникам можливість створювати ефективні та масштабовані серверні додатки з використанням JavaScript. Його мультиплатформність дозволяє розробникам працювати на різних операційних системах, забезпечуючи гнучкість у роботі.

Великі компанії, такі як eBay, Netflix і Uber, вже оцінили переваги Node.js і використовують його у своїх проектах. Відгуки технічних директорів, як у випадку з PayPal, підтверджують його ефективність у швидкому втіленні проектів та підвищенні продуктивності розробничих команд.

Одним з ключових факторів популярності Node.js є його легкість освоєння для розробників Frontend. Завдяки спадковості від JavaScript, вони можуть швидко адаптуватися до його особливостей та ефективно використовувати його для розробки серверної частини додатків.

Node.js забезпечує можливість використання готових модулів або розробки власних, що сприяє зручності та ефективності у розробці. Це робить його одним із найпопулярніших інструментів у сфері веб-розробки та дозволяє розробникам створювати продуктивні та масштабовані додатки.

Express.js є ключовим інструментом у веб-розробці на основі Node.js, надаючи розробникам можливість створювати веб-сайти швидко і ефективно. Він володіє мінімальною конфігурацією, що робить його легким у використанні, а також гнучким, дозволяючи розробникам налаштовувати його під свої потреби.

Express.js забезпечує широкий набір функцій для створення різноманітних веб-додатків - від односторінкових до багатосторінкових і гібридних. Використовуючи його, розробники можуть структурувати

програму для обробки різних HTTP-запитів за певними URL-адресами, що робить процес розробки більш зручним та ефективним.

Однією з важливих переваг Express.js є наявність великої кількості компонентів, доступних у менеджері пакетів. Ці компоненти дозволяють автоматизувати і спростити різноманітні завдання веб-розробки, такі як обробка запитів, робота з базами даних, аутентифікація користувачів та інші.

2.2 Опис та обгрунтування вибору структури та методу організації вхідних та вихідних даних

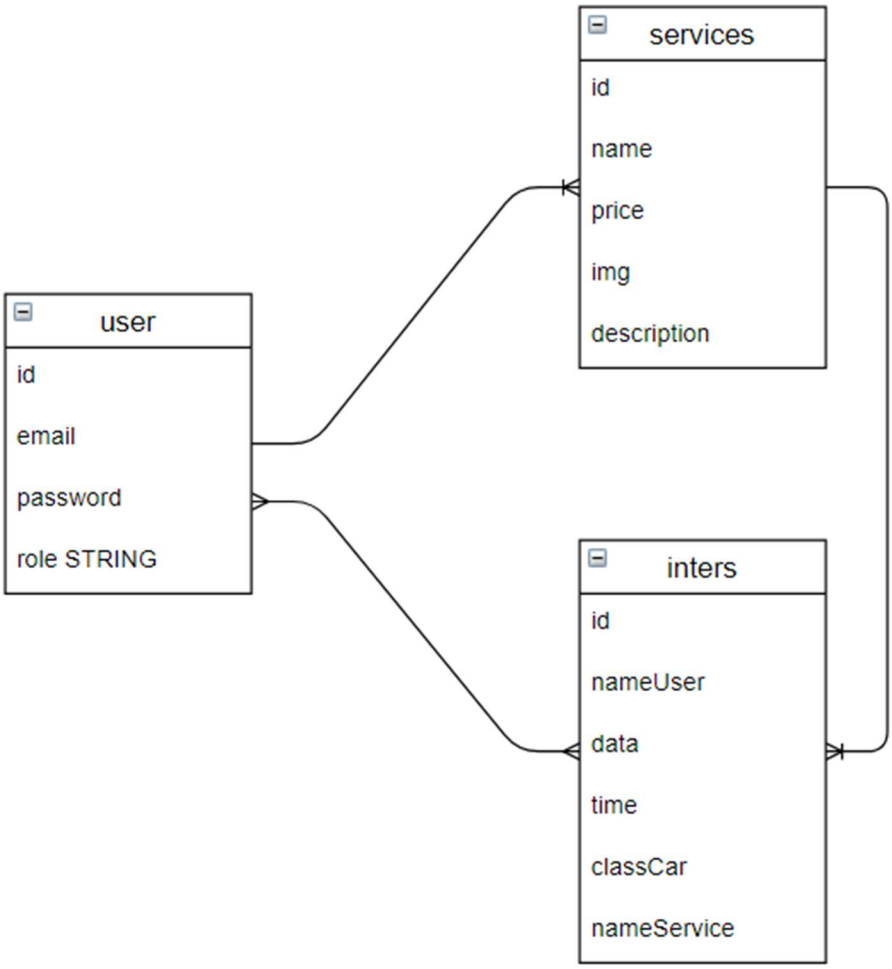
Підготовка до розробки програмного забезпечення передбачає не лише визначення завдань, але й способу організації та обробки інформації на веб-сайті. Це включає в себе створення бази даних та виведення необхідної інформації, такої як перелік послуг, їх опис, інформація про замовлення та дані користувачів.

Для забезпечення повноцінної та ефективної роботи веб-сайту необхідно спроектувати та підключити базу даних. В рамках проектування бази даних для веб-сайту "services" було розроблено такі таблиці: "users" для зберігання даних про користувачів, "inters" для зберігання інформації про замовлення та їх взаємозв'язки, і "services" для опису доступних послуг.

На рисунку 2.1 зображена структура бази даних "services" та зв'язки між її таблицями. Це графічне представлення допомагає краще зрозуміти взаємозв'язки між даними і визначити, як кожна таблиця пов'язана з іншими для ефективного зберігання та отримання інформації.

Правильне проектування бази даних є ключовим етапом у розробці веб-сайту, оскільки від цього залежить ефективність та продуктивність роботи всього застосунку. Врахування потреб користувачів, правильне моделювання взаємозв'язків між даними та оптимальний вибір структури допомагають забезпечити швидкий доступ до інформації та зручну роботу з нею. Такий

підхід дозволяє створити веб-сайт, який відповідає всім вимогам функціональності, безпеки та ергономіки.



Рисунку 2.1 – Схема бази даних «services»

Контролери виступають ключовими елементами у взаємодії між серверною та клієнтською частинами програмного продукту. Вони відповідають за обробку запитів від користувачів, виконання відповідних дій з базою даних та повернення результатів у вигляді відповідей. Розробка контролерів включає в себе створення асинхронних методів, які забезпечують коректну обробку даних та їх взаємодію з базою даних.

У контролері "userController" реалізація методів "registration", "login" та "check" дозволить забезпечити користувачам можливість реєстрації, входу в систему та перевірки авторизації. Контролер "serviceController" відповідає за

створення, отримання всіх послуг та отримання конкретної послуги. Нарешті, контролер "interController" забезпечує можливість створення, отримання всіх інтерв'ю та отримання конкретного інтерв'ю.

Ці контролери виконують ключові функції у взаємодії з клієнтами та базою даних, допомагаючи забезпечити коректну та ефективну роботу програмного продукту. Розробка методів кожного контролера відповідає вимогам функціональності веб-сайту та дозволяє забезпечити зручний та надійний інтерфейс для користувачів.

2.3 Структура та огляд сайту

Структурна модель веб-сайту є одним із ключових етапів в розробці, оскільки вона визначає логічну організацію та взаємозв'язки між різними компонентами сайту. Навігація та зв'язки між сторінками веб-сайту повинні бути розроблені таким чином, щоб забезпечити зручний та легкий доступ до інформації для користувачів.

Зовнішнє проектування веб-сайту, зокрема дизайн, відіграє важливу роль у привернені уваги користувачів і створенні першого враження про компанію. Лаконічний, приємний та зручний у використанні інтерфейс веб-сайту допомагає залучити і утримати аудиторію.

Для розробки клієнтської частини веб-сайту було використано JavaScript бібліотеку React, яка дозволяє створювати динамічні та інтерактивні інтерфейси. Навігація по сайту реалізована за допомогою компонента Navbar з бібліотеки React, що спрощує доступ до різних розділів та сторінок сайту для користувачів.

Ці кроки дозволяють створити зручний та ефективний веб-сайт, який відповідає потребам користувачів та допомагає досягти поставлених цілей компанії.

React Router є потужним інструментом для маршрутизації в React додатках, який дозволяє легко управляти переходами між сторінками. Його основні пакети, такі як react-router і react-router-dom, надають набір компонентів і функцій для створення маршрутів та навігації в додатку.

Пакет react-router-dom є основним для веб-додатків і містить компоненти для маршрутизації в браузері. Ці компоненти, такі як Route, Switch, Link і Redirect, дозволяють визначати маршрути і переходи між різними сторінками. Вони дозволяють створювати динамічні маршрути, які відображаються залежно від стану додатку та дій користувача.

Завдяки React Router DOM можна ефективно організувати маршрутизацію на основі компонентів, що спрощує розробку і підтримку веб-додатків. Це дає можливість створювати динамічні та інтерактивні додатки, які забезпечують зручну навігацію користувачам.

Використання бібліотеки ReactDOM в програмі React дозволяє ефективно працювати з Document Object Model (DOM), використовуючи декларативний підхід. Замість того, щоб напряму маніпулювати DOM елементами, React використовує віртуальне представлення інтерфейсу користувача, яке зберігається в пам'яті. Після цього віртуальне представлення синхронізується з реальним DOM за допомогою ReactDOM, що називається узгодженням.

Цей декларативний підхід дозволяє вказувати React, у якому стані повинен знаходитись інтерфейс, і він автоматично керує оновленням DOM, щоб відповідати цьому стану. Такий підхід дозволяє уникнути прямих маніпуляцій з атрибутами, обробки подій та оновлення DOM вручну, що зроблює розробку додатків більш простою та зрозумілою.

На головній сторінці веб-сайту "Space sink" використано компонент "Carousel" з бібліотеки React-Bootstrap. Цей компонент дозволяє створювати привабливий та інтерактивний слайдер.

Рисунок 2.2 - Зовнішній вигляд компоненту «Carousel»

Нижче наведено інформацію про детейлінг автомобіля та коротку характеристику діяльності студії "Space sink". На рисунку 2.3 представлена головна сторінка "Space sink".

Рисунок 2.3 - Головна сторінка «Space sink»

На сторінці послуг, з лівого боку, розташований компонент з інформацією про діяльність центру. Основна частина веб-сторінки заповнена блоками послуг (див. рис. 2.4).

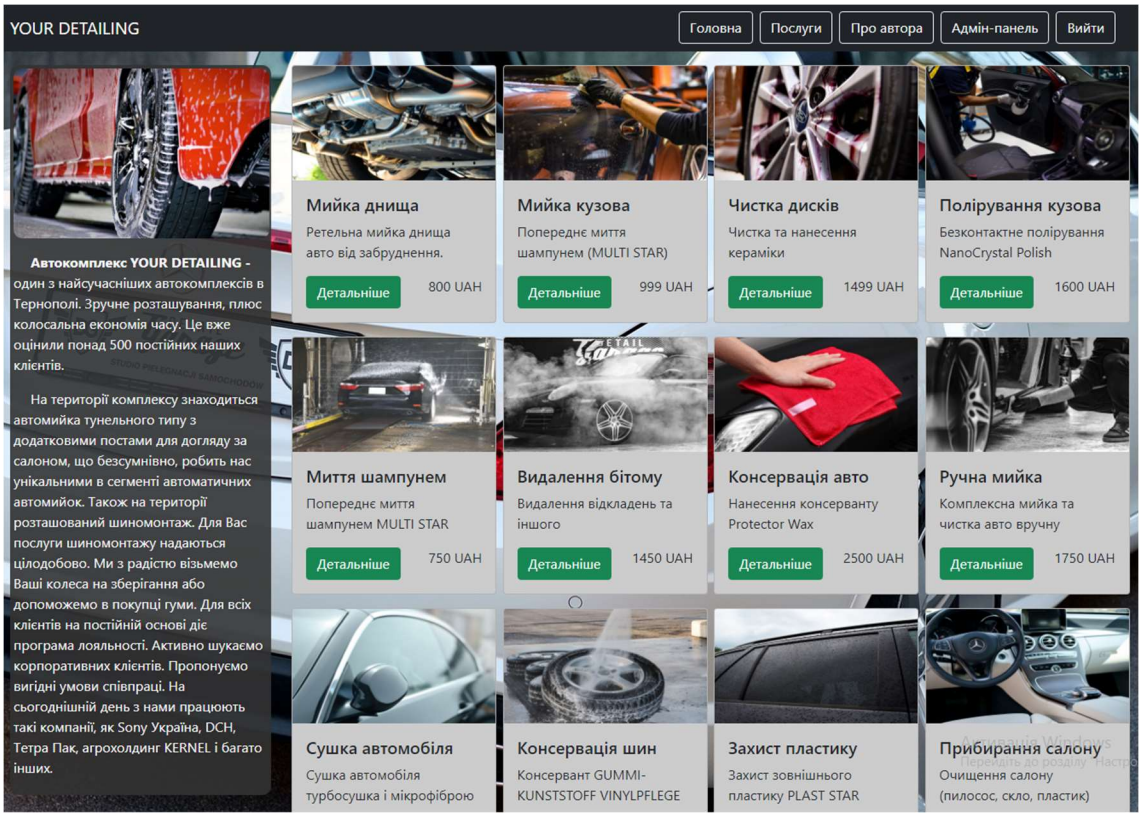


Рисунок 2.4 – Сторінка послуги

При натисканні на кнопку детальніше, відкривається сторінка детального огляду поточної послуги (див. рис. 2.5). Нижче розташована інформація про класифікацію автомобілів.

На сторінці детального огляду послуги користувач може знайти докладну інформацію про включені процедури та послуги, що надаються. Це дозволяє клієнту отримати повний обсяг інформації перед прийняттям рішення про вибір конкретної послуги. Класифікація автомобілів, розташована нижче, надає додаткову корисну інформацію про різні типи транспортних засобів, на які розповсюджуються надані послуги.

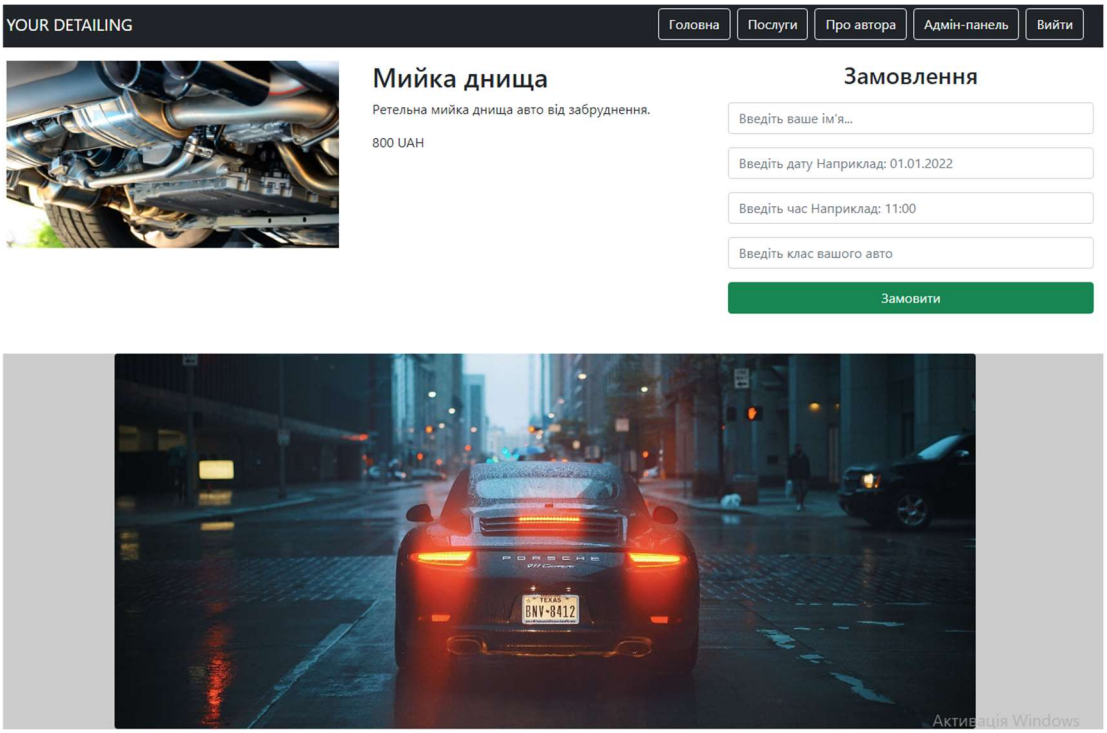


Рисунок 2.5 – Сторінка детальніше

На сторінці адміністратора центру "Space sink" доступні функції для додавання нових послуг та редагування замовлень. Використано компонент "Modal" з бібліотеки React-Bootstrap для зручного взаємодії з цими функціями. Зображена на рисунку 2.6 сторінка адміністратора забезпечує зручний та ефективний інтерфейс для виконання адміністративних завдань.

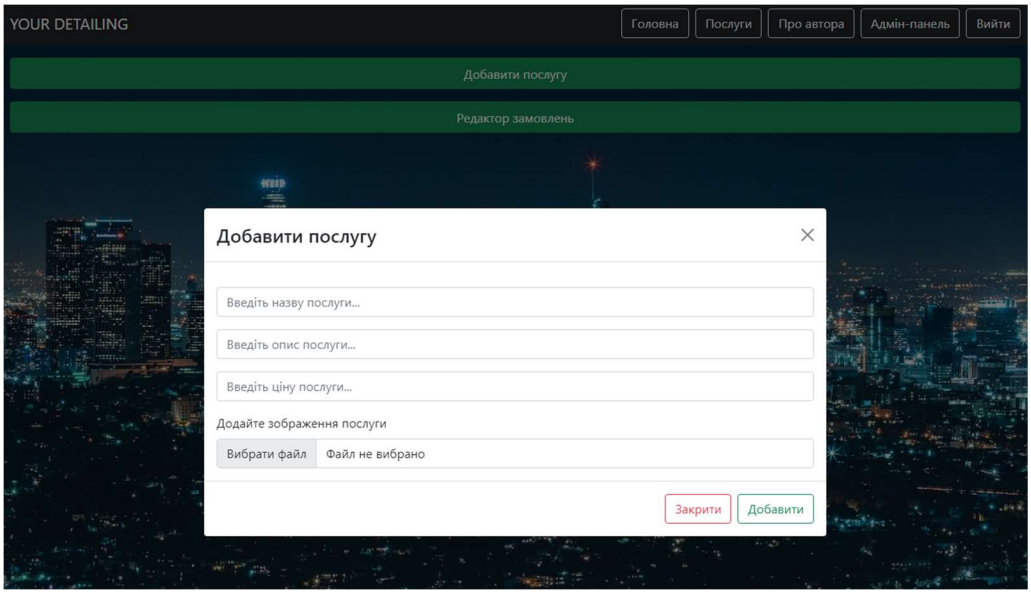


Рисунок 2.6 – Сторінка адміністратора

Для створення сторінки авторизації та реєстрації використано компонент "Card" з бібліотеки React-Bootstrap. Цей компонент дозволяє створювати стилізовані блоки з контентом, що допомагає створити привабливий та функціональний інтерфейс для користувача. На рисунку 2.7 показано вигляд компонента "Card", який використовується для представлення форм авторизації та реєстрації.

Компонент "Card" забезпечує можливість організації контенту у вигляді карток з заголовками, текстом та іншими елементами, які можуть використовуватися для різноманітних цілей. Наприклад, на сторінці авторизації та реєстрації "Card" може містити поля введення для ім'я користувача, пароля, кнопки "Увійти" та "Зареєструватися", а також інші додаткові елементи, такі як посилання на відновлення пароля чи інформаційні повідомлення.

Користуючись можливостями бібліотеки React-Bootstrap, розробники можуть ефективно створювати інтерфейси користувача з використанням готових компонентів, що спрощує процес розробки та забезпечує однорідний стиль у всьому додатку. Компонент "Card" допомагає підвищити зручність та привабливість інтерфейсу, що сприяє поліпшенню користувацького досвіду та зростанню ефективності роботи з програмою.

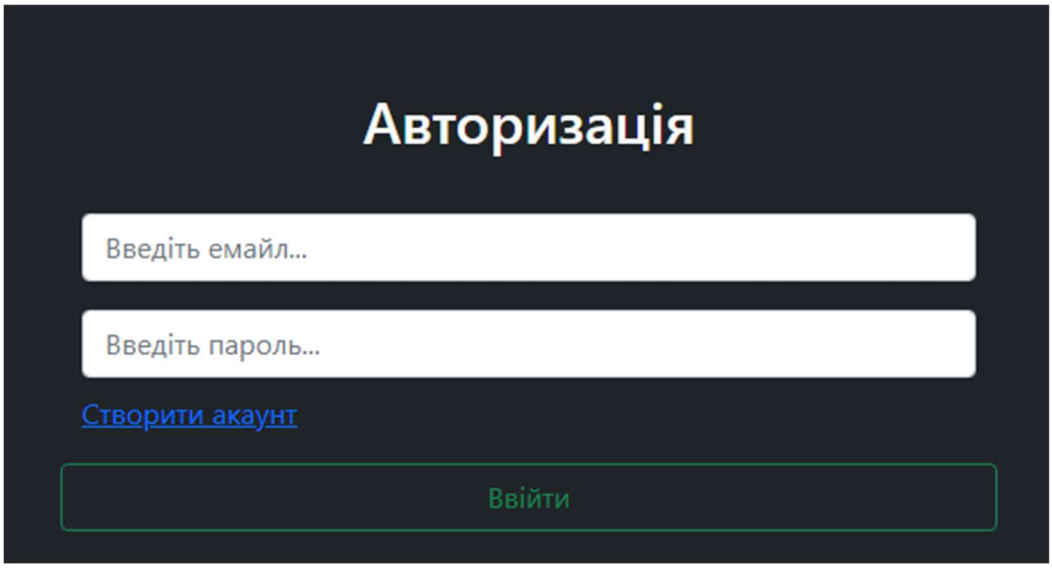


Рисунок 2.7 - Компонент «Card»

2.4 Написання текстів програми

На рисунку 2.8 зображено файлову структуру веб-сайту “Space sink”.

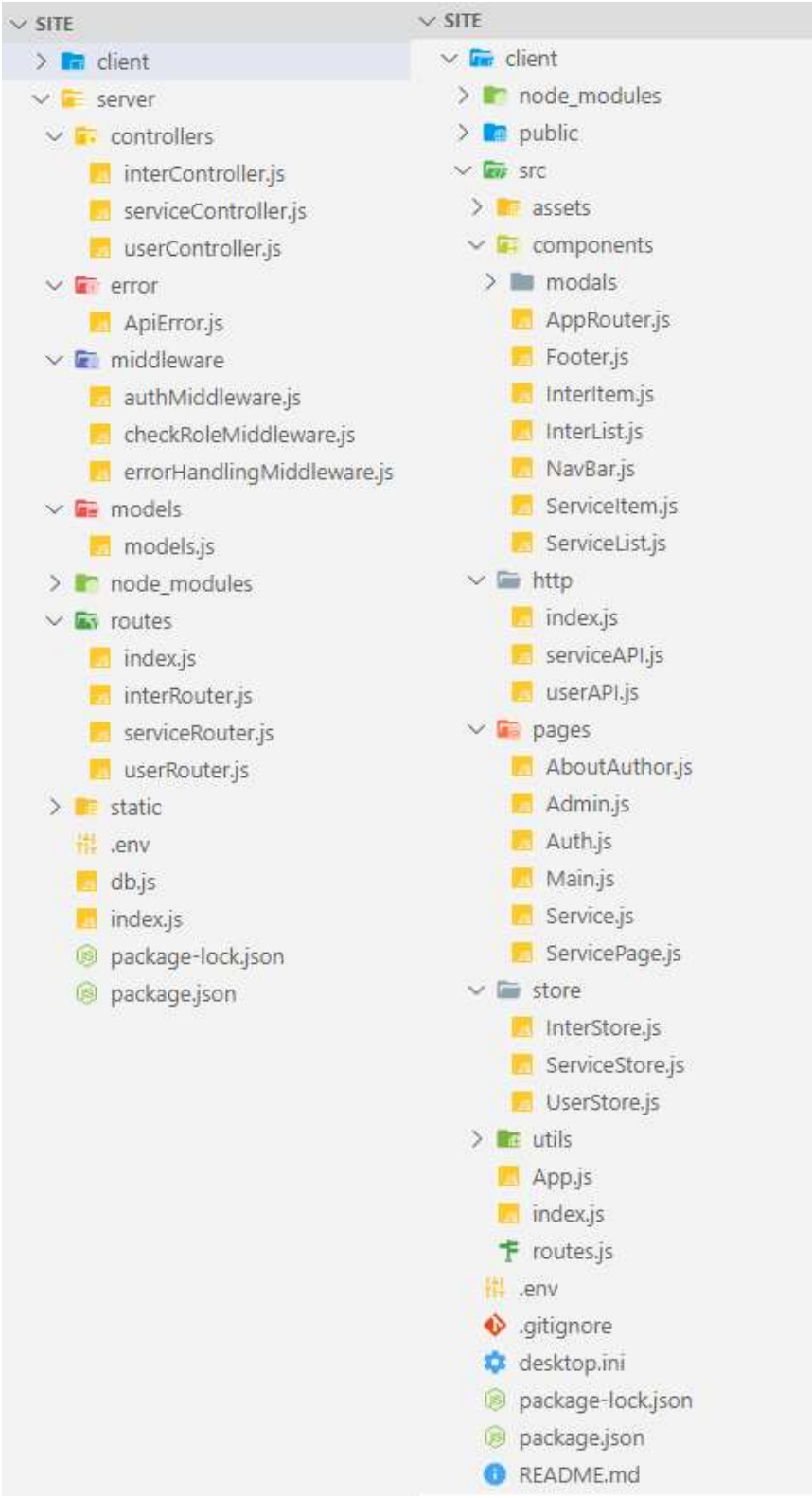


Рисунок 2.8 – Файлова структура проекту

Використання інструменту Nodemon дозволяє значно полегшити процес розробки, забезпечуючи автоматичний перезапуск сервера при внесенні змін у коді. Це забезпечує швидкий цикл змін та тестування, оскільки розробники можуть бачити результати своєї роботи миттєво, без необхідності вручного перезапуску сервера.

Серверна частина програми побудована на основі платформи Node.js з використанням фреймворку Express, що є важливим кроком у створенні потужних та масштабованих веб-додатків. Файл index.js відображає головний файл серверної частини програми, де визначені основні маршрути, middleware та налаштування сервера. Використання Express спрощує обробку запитів, маршрутизацію та інші аспекти роботи з сервером, забезпечуючи зручний та ефективний спосіб створення веб-додатків.

```
require('dotenv').config()

const express = require('express');
const sequelize = require('./db')
const models = require('./models/models')
const PORT = process.env.PORT || 5000;
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const errorHandler = require('./middleware/errorHandlingMiddleware')
const path = require('path')
const app = express();
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use('/api', router)

app.use(errorHandler) //Обробка помилок, останній Middleware

const start = async () => {
    try{
        await sequelize.authenticate() // Функція для підключення бази даних до сервера
        await sequelize.sync()      //функція звіряє стан бази данних зі схемою данних
        app.listen(PORT, () => console.log(`Server started on port ${PORT}`)) // Функція для прослуховування порта
    } catch (e) {
        console.log(e)
    }
}

start()
```

Рисунок 2.9 – Файл index.js

На рисунку 2.10 зображено конфігураційний файл db.js підключення бази даних PostgreSQL.

```
const {Sequelize} = require('sequelize')
module.exports = new Sequelize(
    process.env.DB_NAME, //Передача назви бази данних
    process.env.DB_USER, //Передача назви користувача
    process.env.DB_PASSWORD, //Передача пароля користувача
    {
        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
    }
)
```

Рисунок 2.10 – Файл db.js

Для комфортної роботи та взаємозв'язку бази даних з серверною частиною програмного продукту було використано ORM інструмент Sequelize. Моделі даних та зв'язки між ними подано на рисунку 2.11.

```
const sequelize = require('../db')
const {DataTypes} = require('sequelize')

const User = sequelize.define( 'user', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
    email: {type:DataTypes.STRING, unique: true},
    password: {type:DataTypes.STRING},
    role:{type:DataTypes.STRING, defaultValue: "USER"}
})
const Inter = sequelize.define( 'inter', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
    nameUser: {type:DataTypes.STRING},
    data: {type:DataTypes.STRING},
    time:{type:DataTypes.STRING},
    classCar: {type:DataTypes.STRING},
    nameService: {type:DataTypes.STRING},
    userId: {type:DataTypes.INTEGER}
})
const Basket = sequelize.define( 'basket', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
})
const BasketService = sequelize.define( 'basket_service', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
})
const Service = sequelize.define( 'service', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
    name: {type: DataTypes.STRING, allowNull: true, unique: true},
    price: {type: DataTypes.INTEGER, allowNull: true},
    img: {type: DataTypes.STRING},
    description: {type: DataTypes.STRING}
})

User.hasMany(Inter)
Inter.belongsTo(User)

Basket.hasMany(BasketService)
BasketService.belongsTo(Basket)

BasketService.hasOne(Service)
Service.belongsTo(BasketService)

module.exports = {
    User, Basket, BasketService, Service, Inter
}
```

Рисунок 2.11 – Моделі даних

Для обробки помилок у програмі створено клас ApiError, який забезпечує зручний спосіб роботи з повідомленнями про помилки та статус-кодами (див. рис. 2.12). Конструктор цього класу приймає два параметри: status та message. Спочатку викликається батьківський конструктор за допомогою методу super(), після чого параметри status та message зберігаються в об'єкті.

Також у класі ApiError реалізовані статичні функції, які дозволяють повертати відповідні статус-коди та повідомлення про помилки. Це забезпечує стандартизований підхід до обробки помилок у всій програмі.

```
class ApiError extends Error {
    constructor(status, message) {
        super();
        this.status = status;
        this.message = message;
    }
    static badRequest(message) {
        return new ApiError(404, message);
    }
    static internal(message) {
        return new ApiError(500, message);
    }
    static forbidden(message) {
        return new ApiError(403, message);
    }
}
module.exports = ApiError;
```

Рисунок 2.12 – Клас ApiError

Авторизація користувачів у системі реалізована за допомогою JSON Web Token (JWT). JWT є стандартом для створення токенів доступу на основі JSON і зазвичай використовується для передачі даних та аутентифікації в клієнт-серверних програмах. Процес роботи з JWT полягає в наступному: сервер створює токен, підписує його секретним ключем, а потім передає клієнту. Клієнт використовує цей токен для підтвердження своєї особи при подальших запитах до сервера.

JWT токен складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису (signature), що наочно показано на рисунку 2.13. Заголовок містить інформацію про тип токена та алгоритм шифрування, корисне навантаження зберігає заявлені дані, а підпис використовується для верифікації цілісності токена та підтвердження його автентичності.

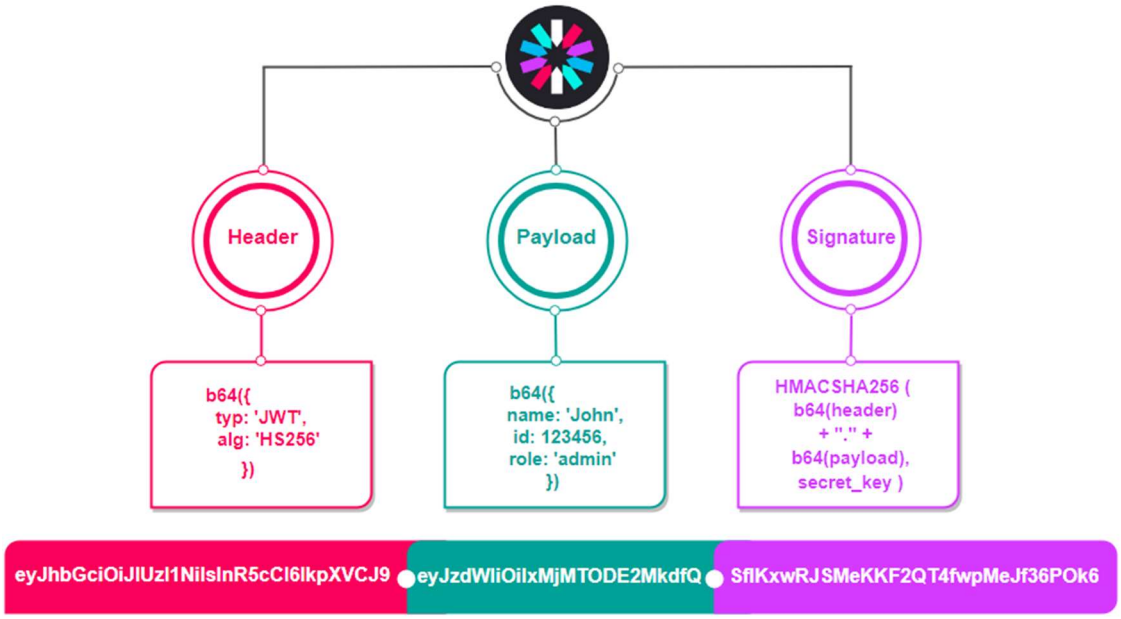


Рисунок 2.13 – Склад JWT токена

Для авторизації користувачів розроблено клас `UserController`, який забезпечує асинхронну роботу з користувачами та містить три основні методи: реєстрації, авторизації та перевірки стану користувача (див. рис. 2.14).

- Метод реєстрації відповідає за створення нового користувача в системі. Він приймає необхідні дані від клієнта, перевіряє їх коректність, створює нового користувача та зберігає його у базі даних.

- Метод авторизації обробляє вхід користувача. Він перевіряє введені дані, а саме електронну пошту та пароль, порівнює їх з даними у базі, і якщо вони співпадають, створює та повертає JWT токен для подальшої автентифікації.

- Метод перевірки стану користувача дозволяє перевірити, чи авторизований користувач, і отримати інформацію про його поточний стан. Це може включати перевірку валідності JWT токена та повернення відповідної інформації.

Ці методи забезпечують необхідний функціонал для управління користувачами та їх авторизації у системі, дозволяючи ефективно працювати з клієнтськими запитами та підтримувати безпеку даних користувачів.

```
class UserController {
    async registration(req, res, next){
        const {email, password, role} = req.body
        if(!email || !password) {
            return next(ApiError.badRequest('Некоректно введено емайл або пароль'))
        }
        const candidate = await User.findOne({where: {email}})
        if(candidate) {
            return next(ApiError.badRequest('Користувач з таким email уже існує'))
        }
        const hashPassword = await bcrypt.hash(password, 5) // хешування пароля
        const user = await User.create({email, role, password: hashPassword}) // передача данних в обробник події, створення акаунта
        const basket = await Basket.create({userId: user.id}) // створення моделі корзини
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }

    async login(req, res, next) {
        const {email, password} = req.body
        const user = await User.findOne({where: {email}})
        if (!user) {
            return next(ApiError.internal('Користувача з таким email не існує'))
        }
        let comparePassword = bcrypt.compareSync(password, user.password)
        if (!comparePassword) {
            return next(ApiError.internal('Невірно вказаний пароль'))
        }
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }

    async check(req, res, next){
        const token = generateJwt(req.user.id, req.user.email, req.user.role)
        return res.json({token})
    }
}
```

Рисунок 2.14 – Клас UserController

Для взаємодії з REST API було використано JavaScript бібліотеку Axios. Axios – це HTTP клієнт, який використовує проміси (Promises) для обробки асинхронних запитів, що дозволяє ефективно використовувати функції JavaScript, такі як async та await. Ця бібліотека забезпечує простий і зрозумілий спосіб роботи з HTTP запитами, що полегшує обмін даними між клієнтом та сервером.

На рисунку 2.15 представлений файл index.js з директорії http, в якому налаштовано використання бібліотеки Axios. У цьому файлі налаштовано базову конфігурацію для роботи з REST API, зокрема встановлено базову URL адресу, додано заголовки за замовчуванням, а також створено функції для відправки GET, POST, PUT та DELETE запитів.

Основні моменти налаштування:

- Базова URL адреса: визначається шлях до API сервера, який буде використовуватися у всіх запитах.

- Заголовки: включаються стандартні заголовки, такі як Content-Type, Authorization тощо.

- Функції для запитів: створюються функції для виконання різних типів запитів (GET, POST, PUT, DELETE), які використовують async та await для асинхронної роботи та обробки відповідей від сервера.

Це дозволяє спростити процес написання коду для взаємодії з API, підвищує читабельність коду та забезпечує більш структурований підхід до обробки запитів і відповідей.

Окрім базової конфігурації, у файлі index.js реалізовано обробку помилок, що дозволяє легко ідентифікувати та реагувати на різні види помилок, які можуть виникнути під час виконання HTTP запитів. Це забезпечує додаткову надійність та стабільність роботи додатка, полегшуючи підтримку та відлагодження коду.

```
import axios from "axios";
const $host = axios.create({ baseURL: process.env.REACT_APP_API_URL})
const $authHost = axios.create({ baseURL: process.env.REACT_APP_API_URL })
const authInterceptor = config => { config.headers.authorization = `Bearer ${localStorage.getItem('token')}`
    return config
}
$authHost.interceptors.request.use(authInterceptor)
export { $host, $authHost}
```

Рисунок 2.15 – Файл директорії http, index.js.

На рисунку 2.16 подано приклади запитів для створення та отримання послуг і замовлень. Ці запити демонструють, як за допомогою бібліотеки Axios можна легко взаємодіяти з REST API для виконання таких операцій.

```
import {$authHost, $host } from "./index";
export const createService = async (service) => {
    const {data} = await $authHost.post('api/service', service)
    return data
}
export const fetchServices = async () => {
    const {data} = await $host.get('api/service')
    return data
}
export const fetchOneService = async (id) => {
    const {data} = await $host.get('api/service/' + id)
    return data
}
export const createInter = async (inter) => {
    const {data} = await $authHost.post('api/inter', inter)
    return data
}
export const fetchInter = async () => {
    const {data} = await $host.get('api/inter')
    return data
}
export const fetchOneInter = async () => {
    const {data} = await $host.delete('api/inter')
    return data
}
```

Рисунок 2.16 – Файл serviceAPI.js

Для авторизації подано наступні запити: реєстрація, авторизація та валідація токена(див. рис. 2.17).

```
import {$authHost, $host } from "./index";
import jwt_decode from "jwt-decode";
export const registration = async (email, password) => {
    const {data} = await $host.post('api/user/registration', {email, password, role: 'USER'})
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token)
}
export const login = async (email, password) => {
    const {data} = await $host.post('api/user/login', {email, password})
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token)
}
export const check = async () => {
    const {data} = await $authHost.get('api/user/auth',)
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token)
}
```

Рисунок 2.17 – Файл userAPI.js

На рисунку 2.18 відображений файл клієнського застосунку index.js. За допомогою селектора getElementById додаток React було додано в index.html.

```
import React, {createContext} from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import UserStore from './store/UserStore';
import ServiceStore from './store/ServiceStore';
import InterStore from './store/InterStore';

export const Context = createContext(null)

ReactDOM.render(
    <Context.Provider value={{
        user: new UserStore(),
        service: new ServiceStore(),
        inter: new InterStore()
    }}>
        <App />
    </Context.Provider>,
   document.getElementById('root')
);
```

Рисунок 1.18 – Файл index.js

Основний компонент React – app.js, який поданий на рисунку 2.19.

```
import React, { useContext, useEffect, useState } from 'react';
import {BrowserRouter} from 'react-router-dom';
import AppRouter from './components/AppRouter';
import Footer from './components/Footer';
import NavBar from './components/NavBar';
import { observer } from 'mobx-react-lite';
import { Context } from './index';
import { check } from './http/userAPI';
import { Spinner } from 'react-bootstrap';
const App = observer( () => {
  const {user} = useContext(Context)
  const [loading, setLoading] = useState(true)
  useEffect( () => {
    check().then(data => {
      user.setUser(true)
      user.setIsAuth(true)
      user.setIsAdmin(true)

    }).finally(() => setLoading(false))
  })
  if (loading) {
    return <Spinner animation={"grow"}/>
  }
  return (
    <BrowserRouter>
      <NavBar/>
      <AppRouter />
      <Footer />
    </BrowserRouter>
  );
})
export default App;
```

Рисунок 2.19 – Файл app.js

2.5 Тестування та налагодження програм

При тестуванні веб-сайту “Space sink” реалізовано наступний ряд тестів:

- функціональне тестування;
- тест на зручність у користуванні веб-сайтом;
- тестування сумісності або конфігураційне тестування;
- тест на продуктивність;
- перевірка безпеки.

Функціональне тестування сайту – це один з найбільш важливих видів тестування, з якого все розпочинається. Основна мета – переконатися в справності роботи основних функцій веб-сайту. Чек-лист цього етапу виглядає наступним чином:

- перевірка коректності посилань;
- перевірка форм – забезпечує взаємодію з користувачами та отримання від них даних;
- тест файлів cookie – важливий для підтримки та коректної роботи сеансів авторизації;
- валідація HTML/CSS – перевірка на відсутність значних синтаксичних помилок та забезпечення доступу ресурсу для різних пошукових систем;
- перевірка бази даних – важливо відстежити коректність виконання запитів, а також вилучення та оновлення даних.

Перевірка зручності у використанні дозволяє оцінити сприйняття дизайну сайту та його функціоналу користувачем і те, наскільки довго користувач готовий залишитися на ньому. Сайт повинен бути послідовним і чітко структурованим. На даному етапі критерії перевірки включають:

- навігаційне тестування – перевірка доступу до меню, сторінок, блоків, полів і кнопок. Важливо, щоб структура була логічною та зрозумілою;
- перевірка контенту – перевірка граматики і орфографії, розміщення заголовків, а також коректні розміри зображень;

- комфорт користування – перевірка структури сайту та наявності зайвих елементів на сторінці.

На етапі тестування сайту на сумісність слід перевірити адаптивність сайту. Це так зване нефункціональне тестування, яке відображає, наскільки коректно відображається ресурс на різних екранах, браузерах і пристроях.

Розрізняють наступні типи:

- кросплатформений тест – перевірка роботи сайту на різних операційних системах;

- перевірка на кросбраузерність – сайт повинен однаково коректно функціонувати в усіх відомих браузерах (Google Chrome, Firefox, Internet Explorer). Важливу роль у цьому відіграє якісна та сучасна верстка.

Тест на продуктивність – важливий етап перевірки, який дозволяє оцінити стабільність роботи ресурсу в критичних ситуаціях. Включає наступні методи:

- тестування навантаженням – перевіряється здатність одночасної обробки великої кількості користувацьких запитів. Проводиться для визначення пропускної здатності ресурсу;

- стрес-тестування – перевірка стійкості ресурсу в екстремальних умовах. Допомагає уникнути помилок під час сильних навантажень та визначити їх допустимий рівень;

- перевірка на швидкість з’єднання – перевіряється час відгуку ресурсу.

Перевірка безпеки – на сьогодні безпека є важливим аспектом в роботі веб-сайтів. Основне завдання тестування полягає в тому, щоб виявити слабкі місця, через які можна втратити дані або порушити роботу системи. Цей етап включає аналіз можливих загроз, перевірку захисту від зловмисних атак та забезпечення конфіденційності даних користувачів.

При функціональному тестуванні важливо також перевірити логіку бізнес-процесів, реалізованих на сайті. Це включає в себе тестування всіх можливих сценаріїв використання функцій, таких як реєстрація, вхід у

систему, розміщення замовлень, обробка платежів та інші ключові операції. Крім того, слід перевіряти інтеграцію з зовнішніми сервісами, такими як платіжні системи, API інших сервісів, щоб упевнитися, що всі взаємодії відбуваються коректно і без помилок.

На етапі тестування зручності використання також важливо залучити реальних користувачів або провести опитування, щоб отримати зворотний зв’язок щодо дизайну і функціональності сайту. Ці дані допоможуть виявити недоліки, які могли залишитися непоміченими під час внутрішнього тестування. Аналіз відгуків користувачів може дати цінну інформацію для покращення навігації, розташування елементів інтерфейсу та загальної взаємодії з сайтом.

Під час тестування на сумісність варто не забувати про мобільні пристрої. Адаптивний дизайн повинен забезпечувати однаковий комфорт користування сайтом як на великих екранах комп’ютерів, так і на менших екранах смартфонів і планшетів. Важливо перевірити, як елементи інтерфейсу змінюються і пристосовуються до різних розмірів екрану, а також переконатися, що всі функції залишаються доступними і зручними у використанні незалежно від пристрою.

Проведення регулярних перевірок безпеки має стати невід’ємною частиною процесу розробки та підтримки сайту. Це включає виявлення та виправлення вразливостей, оновлення систем безпеки, а також постійний моніторинг для запобігання новим загрозам. Впровадження багатофакторної аутентифікації, шифрування даних і інших сучасних методів захисту допоможе забезпечити високу надійність та безпеку роботи веб-сайту.

Крім того, важливо проводити регулярні аудити та оновлення системи для підтримання її в актуальному стані. Це включає оновлення бібліотек і фреймворків до останніх версій, виправлення відомих вразливостей та оптимізацію коду для підвищення продуктивності.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції та налаштування сервера

Щоб встановити Node.js, необхідно перейти на офіційний сайт Node.js та завантажити інсталятор, який відповідає вашій операційній системі. Після завантаження інсталятора необхідно виконати його запуск і слідувати інструкціям майстра встановлення, який крок за кроком проведе вас через процес встановлення. На рисунку 3.1 можна побачити приклад майстра завантаження, який надає інструкції щодо встановлення Node.js. Після успішного завершення установки ви зможете використовувати Node.js для розробки та виконання JavaScript-програм на вашому комп'ютері.

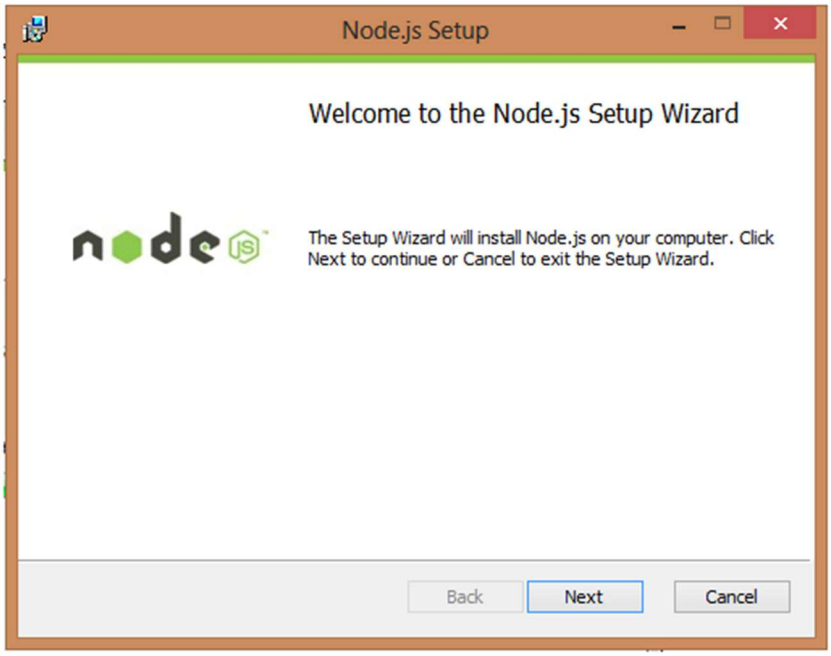


Рисунок 3.1 – Майстер встановлення Node.js

Після встановлення Node.js, необхідно налаштувати сервер. На рисунку 3.2 показаний файл index.js, який містить початкові налаштування сервера.

```
var http = require('http');

http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(3000);

console.log('Server running at http://localhost:3000/');
```

Рисунок 3.2- Файл index.js

Для запуску сервера необхідно відкрити командний рядок Node.js, перейти до директорії проекту і запустити сервер командою "node index.js" (див. рис. 3.3).

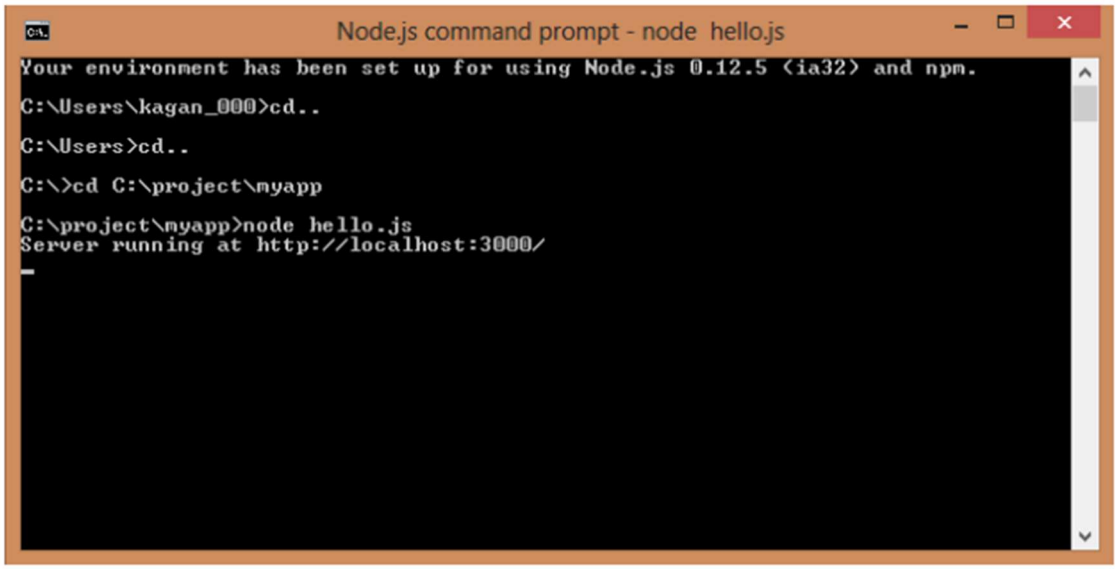


Рисунок 3.3 – Командна стрічка Node.js

Після цього, сервер Node.js буде запущений на вказаному порті з конфігурацією, і ви зможете переглянути його у браузері, відкривши URL: "http://localhost:3000". Після запуску сервера Node.js за вказаною адресою та портом буде доступна веб-сторінка вашого проекту. Це дозволить вам переглядати результати вашої роботи та взаємодіяти з ними у браузері за допомогою локального сервера.

3.2 Інструкція з встановлення програмного забезпечення

Для ефективного обслуговування цього веб-сайту рекомендується встановити наступне програмне забезпечення:

- Visual Studio Code для зручної роботи з кодом;
- pgAdmin4 для керування базами даних PostgreSQL;
- Postman для тестування API;
- Google Chrome для перегляду та відладки веб-сайту.

Visual Studio Code можна завантажити з офіційного веб-сайту за посиланням: "https://code.visualstudio.com" (див. рис. 3.4).

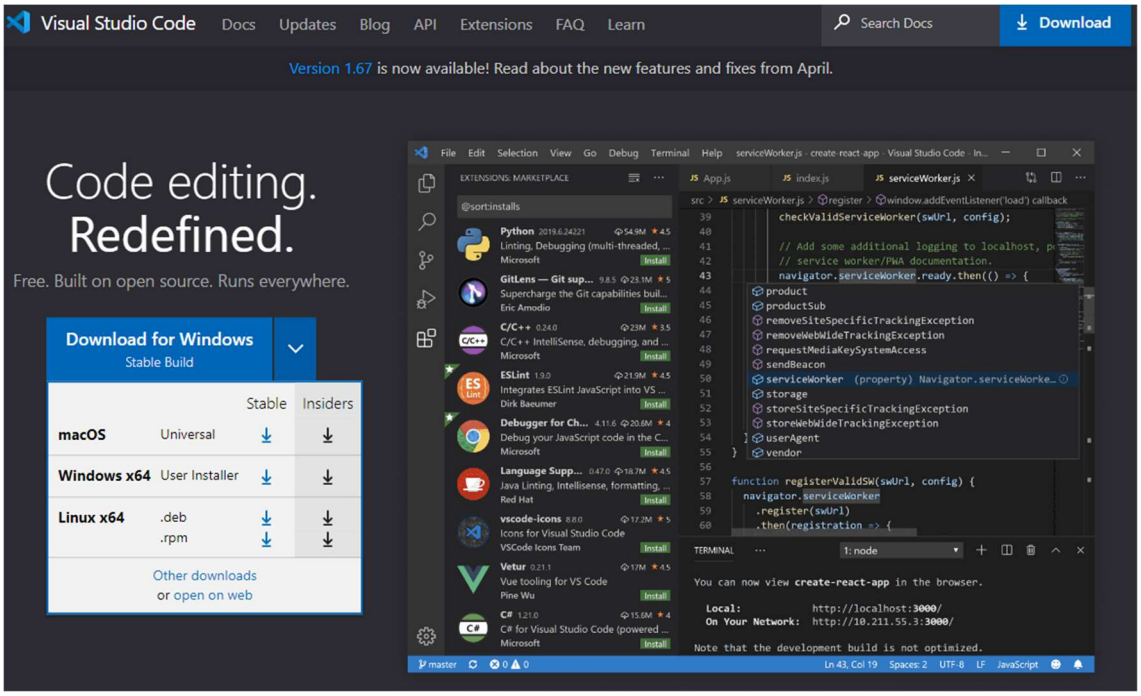


Рисунок 3.4 – Веб-сайт Visual Studio Code

Для встановлення pgAdmin4, перейдіть на офіційний веб-сайт за посиланням: "https://www.pgadmin.org/download/" (див. рис. 3.5) і завантажте інсталятор, який відповідає вашій операційній системі.

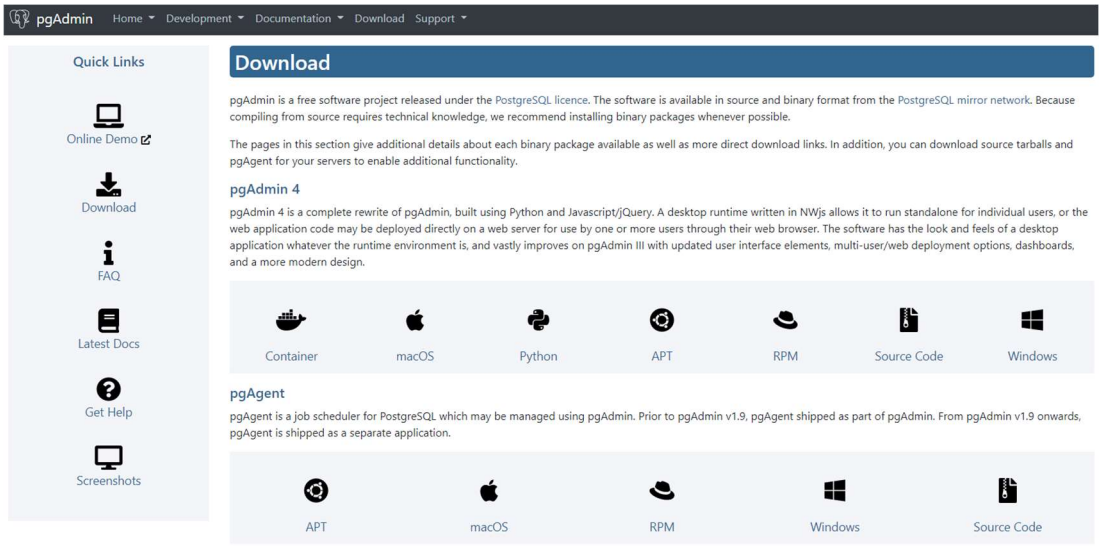


Рисунок 3.5 – Веб-сайт pgAdmin4

Postman необхідний для тестування веб-сайтів. Для встановлення цього інструменту, відвідайте офіційний веб-сайт і завантажте відповідний інсталятор.

На рисунку 3.6 показано офіційний веб-сайт Google, з якого можна завантажити інсталятор браузера Google Chrome.

Після завершення завантаження інсталятора Postman та програми Google Chrome, виконайте інструкції з встановлення програм на вашому пристрої. Після успішної установки ви зможете використовувати ці інструменти для тестування та відлагодження вашого веб-сайту.

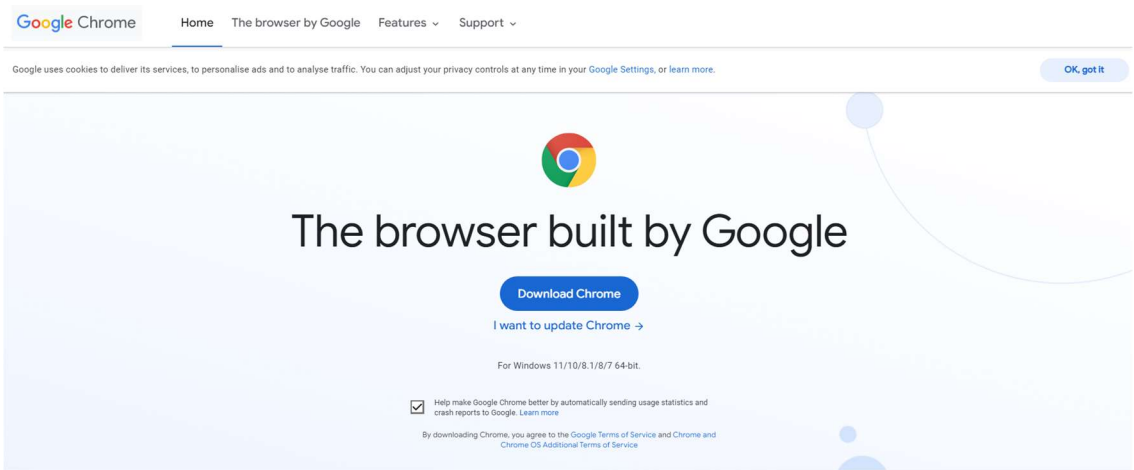


Рисунок 3.6 – Веб-сайт Google Chrome

3.3 Інструкція з встановлення сайту на хостинг

Щоб встановити веб-сайт на хостинг, спочатку необхідно створити обліковий запис на обраному хостингу. У цьому випадку буде використано безкоштовний хостинг Heroku. Після створення облікового запису слід створити проект, в який буде встановлюватися ваш застосунок (див. рис. 3.7).

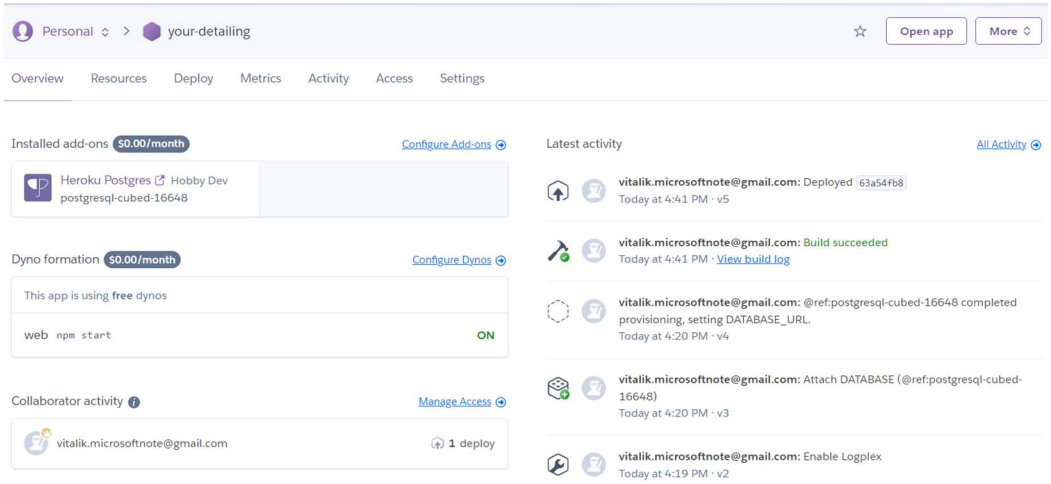


Рисунок 3.7 – Створення проекту на Heroku

Далі необхідно встановити Heroku CLI. Після цього в командній стрічці Visual Studio Code переходимо в директорію проекту та виконуємо наступні команди:

- $ heroku login;
- $ heroku git:clone -a space-sink;
- $ cd space-sink;
- $ git add;
- $ git commit -am "make it better";
- $ git push heroku master;

Далі слід підключити базу даних до веб-сайту. Для цього потрібно створити базу даних на Heroku та отримати дані для підключення до неї.

На рисунку 3.8 зображено дані для підключення до бази даних Heroku.

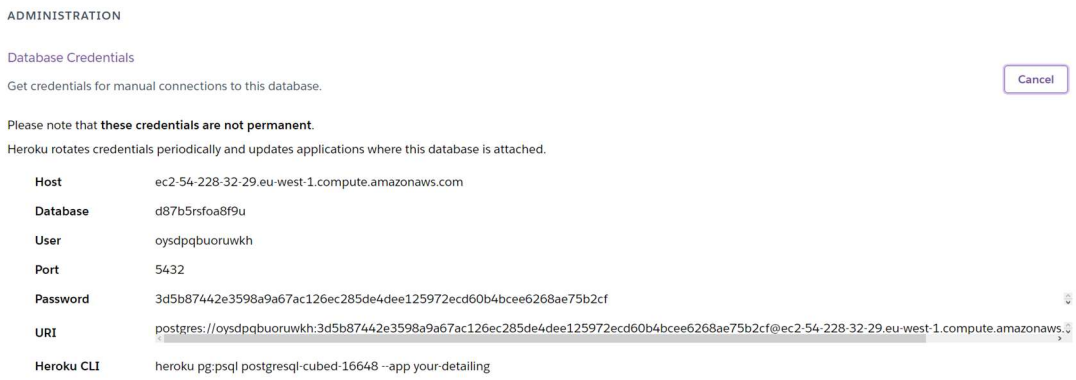


Рисунок 3.8 – Дані бази даних Heroku

Щоб завантажити базу даних на сервер Heroku, спочатку потрібно здійснити підключення до нього через pgAdmin4, використовуючи отримані під час налаштування сервера дані. Після успішного підключення можна створити дамп бази даних, який представлятиме собою резервну копію всієї інформації з бази. Цей дамп можна використати для відновлення даних на сервері Heroku в разі необхідності.

Після створення дампу бази даних важливо виконати процедуру відновлення на сервері Heroku. Це може здійснюватися через інтерфейс адміністрування баз даних на платформі Heroku або через командний рядок, використовуючи Heroku CLI. Після відновлення бази даних на сервері Heroku ви можете бути впевнені, що ваш сайт коректно працюватиме з актуальною версією даних.

Після встановлення свого доменного імені на хостингу Heroku можна забезпечити більш зручний доступ до вашого веб-сайту для користувачів. Вони зможуть легко знаходити та використовувати ваш сайт, використовуючи звичайні та знайомі їм доменні імена. Додавання власного домену також може підвищити професійний вигляд вашого веб-проекту та підвищити його впізнаваність серед користувачів.

3.4 Інструкція з експлуатації веб-сайту

Для авторизації на веб-сайті "Space sink" спочатку потрібно зареєструватися. Для цього виберіть кнопку "Авторизація" і введіть адресу електронної пошти та пароль (див. рис. 3.9).

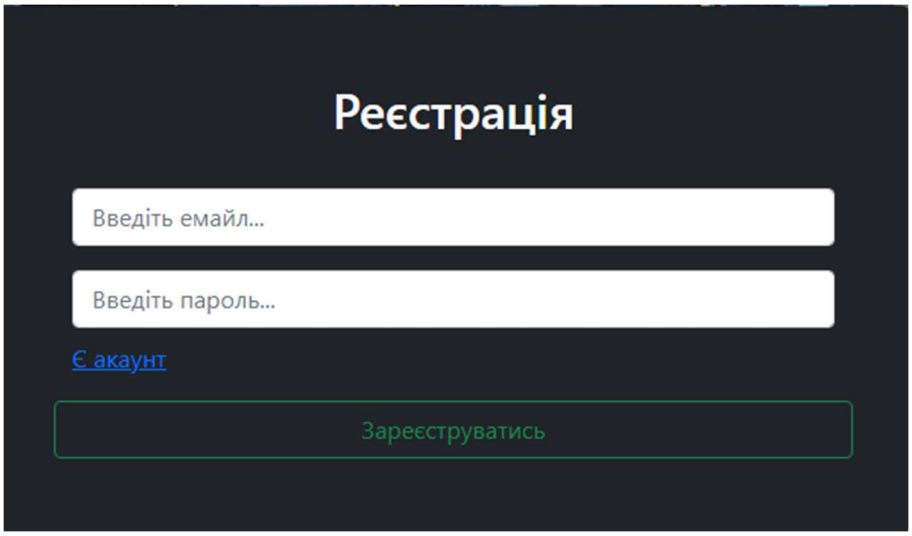


Рисунок 3.9 – Форма реєстрації

У випадку, якщо ваш акаунт уже створено, вам необхідно перейти на сторінку авторизації та ввести адресу електронної пошти та пароль, які ви вказали при реєстрації (див. рис. 3.10).

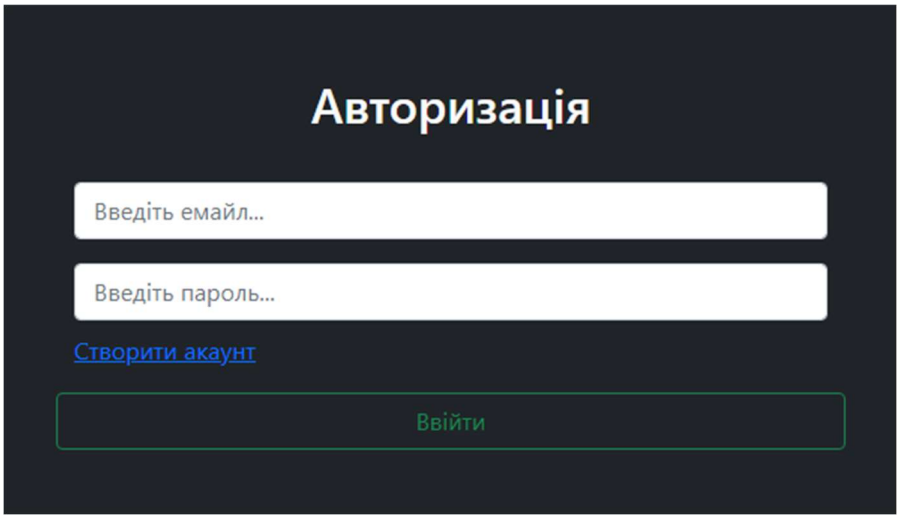


Рисунок 3.10 – Форма авторизації

На веб-сайті реалізовані наступні ролі користувачів:

- Неавторизований користувач: Цей тип користувача має обмежений доступ і може переглядати лише публічний контент веб-сайту. Він не може виконувати дії, які доступні авторизованим користувачам та адміністраторам.

- Авторизований користувач: Після авторизації користувач має доступ до розширених можливостей, таких як замовлення послуг та перегляд історії замовлень. Він може виконувати дії, які не доступні неавторизованим користувачам, але не має повного доступу до адміністративних функцій.

- Адміністратор: Як адміністратор, користувач має повний доступ до адмін-панелі, де може керувати послугами та замовленнями. Він може створювати нові послуги, переглядати та редагувати інформацію про замовлення. Адміністратор також може мати доступ до додаткових функцій, які не доступні іншим користувачам, таких як управління користувачами або зміна налаштувань веб-сайту.

Зображення на рисунках 3.11, 3.12 та 3.13 показують адмін-панель, де адміністратор може керувати послугами та замовленнями.

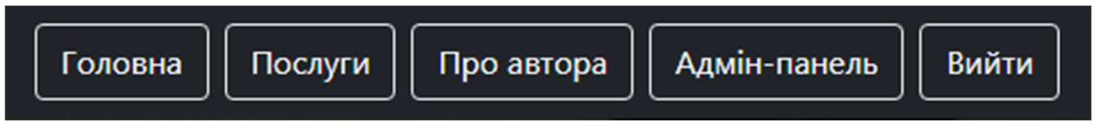


Рисунок 3.11 – Хедер веб-сайту адміністратора

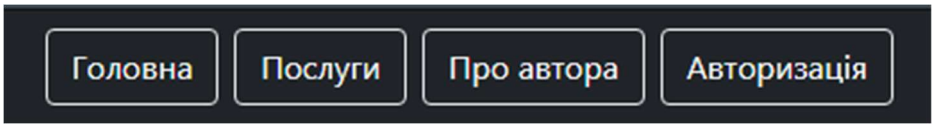


Рисунок 3.12 – Хедер веб-сайту не авторизованого користувача

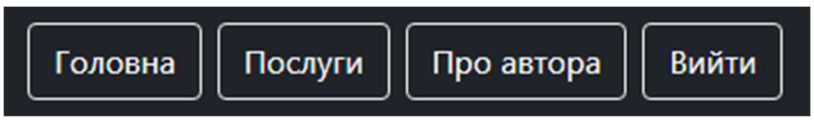


Рисунок 3.13 – Хедер веб-сайту авторизованого користувача

Для створення нової послуги необхідно авторизуватись з роллю адміністратора, перейти на сторінку адмін-панелі та натиснути кнопку "Додати послугу". Після цього відкриється модальне вікно, що містить форму, в яку потрібно ввести дані та зображення для нової послуги. Після введення необхідних даних слід натиснути кнопку "Додати", після чого вікно буде закрито, а нова послуга буде додана до списку.

У випадку, якщо неавторизований користувач спробує отримати доступ до сторінки адміністратора, йому буде виведено повідомлення "Немає доступу". Це забезпечить захист від несанкціонованого доступу до адміністративних функцій.На рисунку 3.14 зображено сторінку “додати послугу”.

Рисунок 3.14 – Форма добавлення послуг

На сторінці "адмін-панелі" також присутня функція для перегляду замовлень (див. рис. 3.15). Ця функція надає адміністратору можливість переглядати усі замовлення, що надійшли на сайт. Вона дозволяє адміністратору з легкістю перевіряти стан замовлень, їх деталі, а також виконувати інші дії, пов'язані з обробкою та управлінням замовленнями. Ця функція є важливою для забезпечення ефективного управління бізнесом.

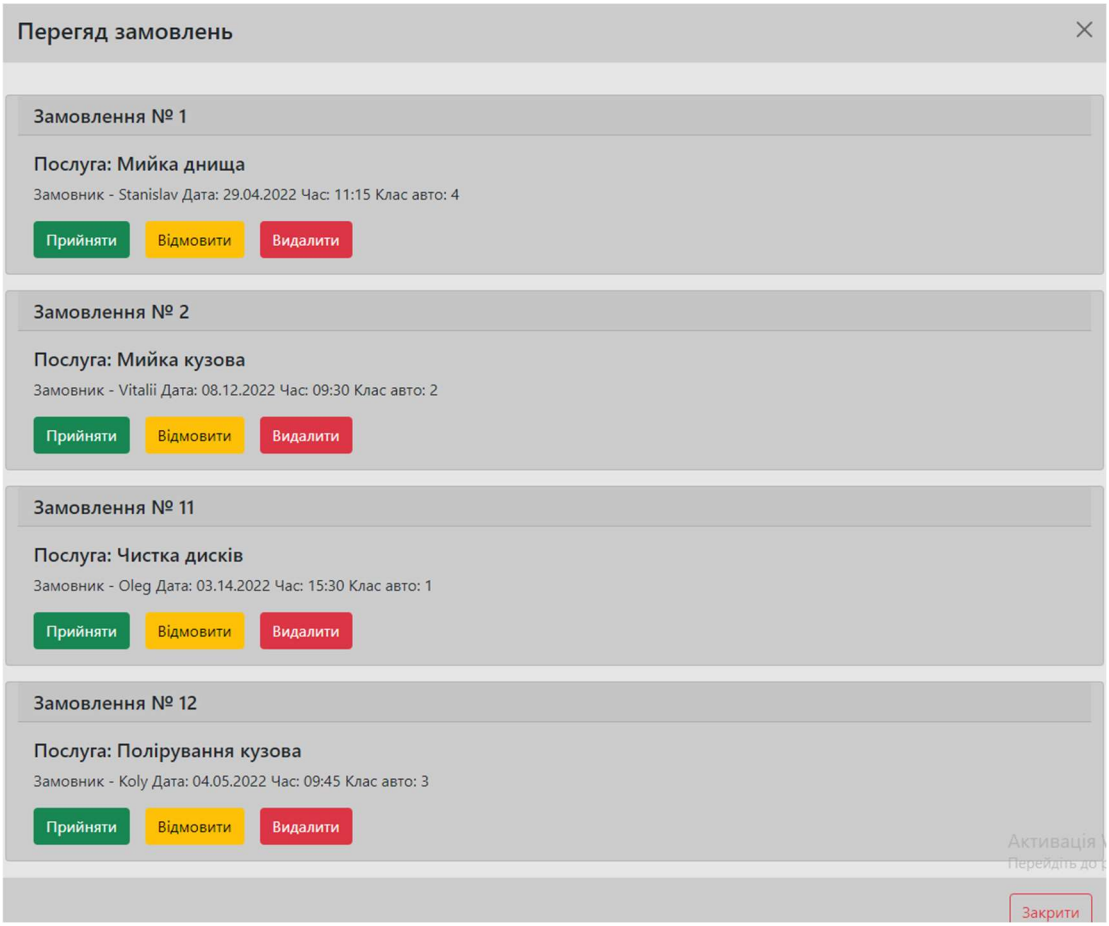


Рисунок 3.15 – Модальне вікно “Редактор замовлень”

Щоб замовити послугу, перейдіть на сторінку "Послуги" (див. рис. 3.16) і оберіть потрібну послугу. Після цього натисніть на кнопку "Детальніше", щоб перейти на сторінку з детальною інформацією про обрану послугу (див. рис. 3.17). На цій сторінці ви знайдете повну інформацію про послугу, а також форму замовлення, яку потрібно заповнити. Після заповнення форми натисніть кнопку "Замовити", і вам буде виведено повідомлення про успішне замовлення. Тепер ваше замовлення буде оброблено, і ви можете очікувати на отримання відповіді з подальшими інструкціями.

Цей процес забезпечує зручний та простий спосіб замовлення послуги для користувачів, дозволяючи їм швидко та легко отримати необхідні послуги без зайвих труднощів. Інформація, доступна на сторінці детального перегляду, допомагає користувачам зробити інформований вибір, а форма замовлення забезпечує простий спосіб оформлення замовлення прямо з сайту.

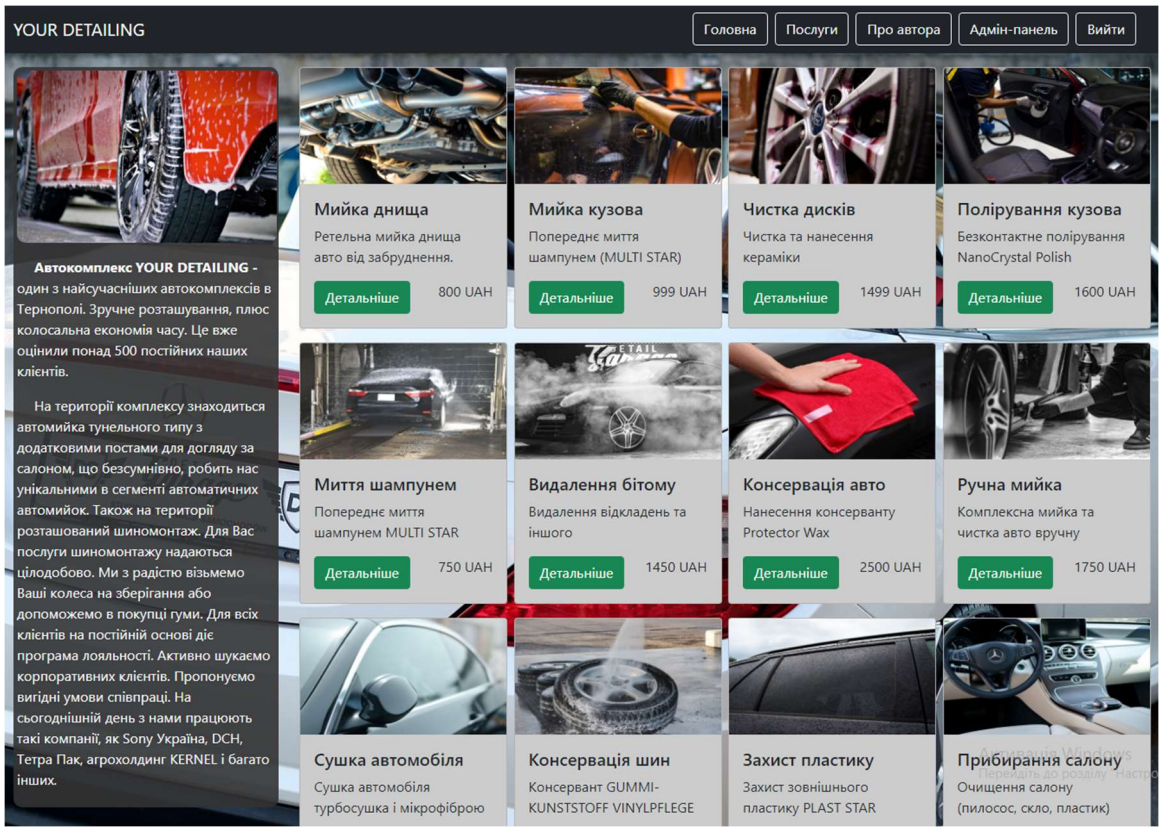


Рисунок 3.16 – Сторінка послуги

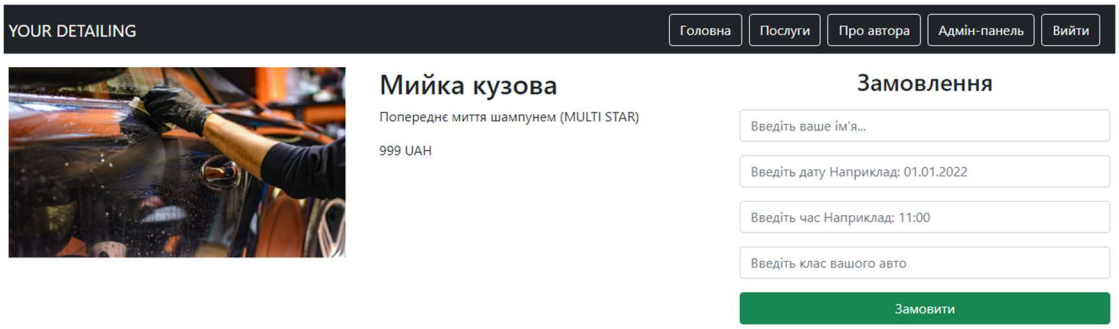


Рисунок 3.17 – Сторінка детальніше

Більш того, процес замовлення послуги через веб-сайт дозволяє клієнтам здійснювати цю операцію у будь-який зручний для них час, без необхідності особисто відвідувати або телефонувати в офіс компанії. Це особливо важливо для сучасних користувачів, які цінують свій час та хочуть мати можливість замовляти послуги швидко та без зайвих турбот.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини дипломного проекту є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності розробки веб–сайту “Space sink”.

Веб–сайт і прийняття рішення про її подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Для розрахунку вартості НДР необхідно виконати наступні етапи:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– визначити суму матеріальних затрат;

– обчислити витрати на електроенергію для науково–виробничих цілей;

– розрахувати транспортні витрати;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість НДР;

– розрахувати ціну НДР;

– визначити економічну ефективність та термін окупності продукту.

4.1 Визначення економічної ефективності і терміну окупності капітальних вкладень

Для визначення загальної тривалості проведення НДР доцільно дані витрат часу по окремих операціях технологічного процесу звести у табл. 4.1.

Таблиця 4.1 – Середній час виконання НДР та стадії (операції) технологічного процесу

№п/п	Назва стадії	Виконавець	Середній час виконання операції, год.
1	Аналіз технічного завдання	Керівник проекту	3 год.
2	Вибір програмного забезпечення	Керівник проекту	3 год.
3	Аналіз потреб та огляд існуючих рішень	Керівник проекту	7 год.
4	Розробка логічної схеми проєкту	Лаборант	12 год.
5	Налаштування програмного забезпечення	Лаборант	4 год.
6	Розробити макет сайту	Лаборант	10 год.
7	Розробка логічної схеми бази даних	Лаборант	6 год.
8	Написання front–end частини сайту	Лаборант	42 год.
9	Написання back–end частини сайту	Лаборант	22 год.
Разом			109 год.

4.2 Визначення витрат на оплату праці та відрахування на соціальні заходи

Відповідно до Закону України «Про оплату праці» заробітна плата – це «винагорода, обчислена у грошовому виразі, яку власник виплачує працівникові за виконану ним роботу».

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно–ділових якостей працівника, результатів його праці та господарської діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата нараховується на виконану роботу за тарифними ставками, відрядними розцінками чи посадовими окладами і не залежить від результатів господарської діяльності підприємства.

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов'язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, умов виробництва, кваліфікації виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

Основна заробітна плата розраховується за формулою:

$$З_{осн.}=Т_{с}* К_{г}, \quad (4.1)$$

де $Т_{с}$ – тарифна ставка, грн.;

$К_{г}$ – кількість відпрацьованих годин.

Рекомендовані тарифні ставки: керівник проекту – 80 грн./год., лаборант – 60 грн./год.

$$З_{осн.}=80*13+60*96=6800 \text{ грн.}$$

Додаткова заробітна плата становить 10–15 % від суми основної заробітної плати.

$$З_{дод.}= З_{осн.}* К_{допл.}, \quad (4.2)$$

де $К_{допл.}$ – коефіцієнт додаткових виплат працівникам.

$$З_{дод.}= 6800*0,15=1020 \text{ грн.}$$

Звідси загальні витрати на оплату праці (Во.п.) визначаються за формулою:

$$В_{о.п.}= З_{осн.}+ З_{дод.}, \quad (4.3)$$

$$В_{о.п.}=6800+1020=7820 \text{ грн.}$$

Крім того, слід визначити відрахування на заробітну плату: єдиний соціальний внесок – 22 %.

Отже, сума відрахувань на соціальні заходи буде становити:

$$B_{з.п.}=ФОП * 0,22, \quad (4.4)$$

де ФОП – фонд оплати праці, грн.

$$B_{с.з.}= 7820*0,22=1720,4 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці зведемо у таблицю 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

No п/п	Категорія прац.	Основна заробітна плата, грн			Додаткова зароб. плата, грн.	Нарахув. на ФОП, грн.	Всього витрат на оплату праці, грн.
		Тариф. ставка, грн.	К–сть від–пр. год.	Факт. нарах. з/пл., грн.			
1	Керівник проекту	80	13	1040	156	–	–
2	Лаборант	60	96	5760	864	–	–
Разом				6800	1020	1720,4	9540,4

Отже, загальні витрати на оплату праці становлять 9540,4 грн.

4.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{Bi} * q_i * p_i, \quad (4.5)$$

де q_i – кількість витраченого матеріалу і–го виду;

p_i – ціна матеріалу і–го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.}=\sum М_{Вi}, \quad (4.6)$$

$$Зм.в.= 220 \text{ грн.}$$

Проведені розрахунки занесемо у табл. 4.3.

Таблиця 4.3 – Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна 1–ці, грн	Загальна сума витрат, грн.
1	Накопичувач SanDisk 64GB Ultra Black	шт.	1 шт.	240	240
Разом			1 шт.	240	240

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1–ці обладнання визначаються за формулою:

$$З_e * W * T * S, \quad (4.7)$$

де W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат–години електроенергії.

Для розробки проекту веб–сайту “Space sink” використовується один ПК, потужність якого W = 0,5 кВт і який працює 100 години. Вартість 1 кВт електроенергії становить 4,32 грн.

$$З_e=0,50*100*4,32=216 \text{ грн.}$$

4.5 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$Ц = \frac{Б_в \cdot Н_А}{100\%} \quad (4.8)$$

де $А$ – амортизаційні відрахування за звітний період, грн.;

$БВ$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$НА$ – норма амортизації, %.

Для проектування даного веб–сайту використовується один комп'ютер (вартість якого становить 25000 грн.), який працює 100 години.

Тоді:

$$А = 25000*0,04*109/150 = 726,6 \text{ грн.}$$

4.6 Обчислення накладних витрат

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно–правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$Н_в = В_{о.п.} * 0,2 \ldots 0,6 , \quad (4.9)$$

де $НВ$ – накладні витрати.

$$Н_В = 7820*0,4 = 3128 \text{ грн.}$$

4.7 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведемо у табл. 4.4.

Таблиця 4.4 – Кошторис витрат на НДР

Зміст витрат	Сума, грн.	В % до загальної суми
Витрати на оплату праці	7820	56,37
Відрахування на соціальні виплати	1720,4	12,4
Матеріальні витрати	240	1,73
Витрати на електроенергію	235,44	1,69
Амортизаційні відрахування	726,6	5,23
Накладні витрати	3128	22,55
Собівартість	13771,06	100

Собівартість ($С_в$) НДР розрахуємо за формулою:

$$С_в = В_{о.п.} + В_{с.з.} + З_{м.в.} + З_е + А + Н_в, \quad (4.10)$$

Св=7820+1720,4+240+235,4+726,6+3128=13771,06 грн.

4.8 Розрахунок ціни НДР

Ціну *НДР* можна визначити за формулою:

$$Ц = \frac{С_в(1+Р_{рвн})\cdot К + В_{н.і}}{К} \cdot (1+ПДВ), \quad (4.11)$$

де $Р_{рен.}$ – рівень рентабельності;

$К$ – кількість замовлень, од.;

$В_{н.і.}$ – вартість носія інформації, грн.;

ПДВ – ставка податку на додану вартість, (20 %).

Ц=13771,06*1,3*1,2=21482,85 грн.

4.9 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва - категорія, яка характеризує результативність виробництва. Вона свідчить не лише про приріст обсягів виробництва, а й про те, якими витратами ресурсів досягається цей приріст, тобто свідчить про якість економічного зростання.

Прибуток розраховується за формулою:

$$П = Ц - С_в \quad (4.13)$$

$$П = 21482,85 – 13771,06 = 7711,79 \text{ грн.}$$

Економічна ефективність ($Е_р$) полягає у відношенні результату виробництва до затрачених ресурсів і розраховується за формулою 4.14.

$$Е_р = П / С_в, \quad (4.14)$$

де П – прибуток;

$С_в$ – собівартість.

$$Е_р = 7711,79 / 13771,06 = 0,55$$

Поряд із економічною ефективністю розраховують (формула 4.15) термін окупності капітальних вкладень ($Т_р$):

$$Т_р = 1 / Е_р \quad (4.15)$$

Допустимим вважається термін окупності до 5 років. В даному випадку

$T_р=1/0,55=1,8$

Таблиця 4.5 - Економічні показники НДР

№	Показник	Значення
1.	Собівартість, грн.	13771,06
2.	Плановий прибуток, грн.	7711,79
3.	Ціна, грн.	21482,85
4.	Термін окупності, рік	1,8

Враховуючи основі економічні показники, зведені у таблицю 4.5, можна зробити висновок, що при терміні окупності – 1,8 року проводити роботи по модернізації веб-сайту є доцільним та економічно вигідним.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Органи державного нагляду за станом охорони праці

Охорона праці – це система правових, соціальноекономічних, організаційно-технологічних, санітарногігієнічних і лікувально-профілактичних заходів і способів, спрямованих на збереження життя, здоров'я, і працездатності людини у процесі праці. (ст.1 Закону України «Про охорону праці»)

Закон України «Про охорону праці»: визначає основні положення щодо реалізації права на охорону життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні.

Основні принципи державної політики в галузі охорони праці, визначені Законом України «Про охорону праці»:

- пріоритет життя й здоров'я працівників стосовно результатів виробничої діяльності підприємства;

- повна відповідальність власника за створення безпечних і нешкідливих умов праці;

- соціальний захист працівників, які постраждали від нещасних випадків на виробництві й професійних захворювань;

- встановлення єдиних нормативів з охорони праці для всіх форм власності й видів їх діяльності і т.п.;

Державний нагляд за додержанням законодавчих та інших нормативно-правових актів про охорону праці здійснюють:

- Комітет по нагляду за охороною праці України (Держнаглядохоронпраці);

- Державний комітет України з ядерної та радіаційної безпеки;

- Органи та заклади санітарно-епідеміологічної служби Міністерства охорони здоров'я України

Посадові особи органів державного нагляду за охороною праці (державні інспектори) мають право:

- безперешкодно в будь-який час відвідувати підконтрольні підприємства для перевірки дотримання законодавства про охорону праці, одержувати від власника необхідні пояснення, матеріали та інформацію з даних питань;

- надсилати керівникам підприємств, а також їх посадовим особам, керівникам структурних підрозділів Ради Міністрів Республіки Крим, місцевих Рад народних депутатів, міністерств та інших центральних органів державної виконавчої влади, обов'язкові для виконання розпорядження (приписи) про усунення порушень і недоліків в галузі охорони праці;

- зупиняти експлуатацію підприємств, окремих виробництв, цехів, дільниць, робочих місць і обладнання до усунення порушень вимог щодо охорони праці, які створюють загрозу життю або здоров'ю працюючих;

- притягати до адміністративної відповідальності працівників, винних у порушенні законодавчих та інших нормативних актів про охорону праці;

- надсилати власникам, керівникам підприємств подання про невідповідність окремих посадових осіб займаній посаді, передавати в необхідних випадках матеріали органам прокуратури для притягнення їх до кримінальної відповідальності.

Громадський контроль за додержанням законодавства про охорону праці здійснюють

- професійні спілки, їх об'єднання в особі своїх виборних органів і представників,

- уповноважена найманими працівниками особа за відсутності професійної спілки на підприємстві.

5.2 Вимоги безпеки під час експлуатації, обслуговування, ремонту й налагодження ПЕОМ

До санітарно-гігієнічних вимоги до приміщення в якому облаштувані робочі місця, обладнанні відеотерміналами, повинно забезпечувати:

-належні умови освітлення приміщення і робочого місця, відсутність відблисків;

- оптимальні параметри мікроклімату;

- належні ергономічні характеристики основних елементів робочого місця, а також враховувати небезпечні і шкідливі фактори.

Вимоги до приміщень наступні:

- площа на одне робоче місце, обладнане відеотерміналом не менше $6м^2$;

- об'єм на одне робоче місце – не менше 20,0 м3;

- обовязково система - опалення;

- система кондиціонування повітря або припливно-витяжна вентиляція;

- щоденне вологе прибирання;

- розміщення робочих місць із ПЕОМ у підвалах і цокольних приміщеннях не допускається;

Санітарно-гігієнічні вимоги до штучного освітлення Приміщення з ПК повинні мати природне і штучне освітлення, яке відповідало б вимогам:

- ДБН В.2.5-28-2006 «Природне і штучне освітлення»,

- ДСанПІН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електроннообчислювальних машин».

Вимоги до вмісту шкідливих речовин у повітрі робочої зони наступні:

- концентрація озону – не більше 0,1 мг/м3;

- вміст оксидів азоту – не більше 5 мг/м3;

- вміст пилу – не більше 4 мг/м3.

Вимоги до організації робочого місця користувача ПЕОМ повино відповідати наступним нормативним документам:

- ГОСТ 12.2.032-78 Система стандартів безпеки праці. Робоче місце при виконанні робіт сидячи. Загальні ергономічні вимоги.

- ДСАНПІН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Розміщення монітора на робочому столі повино бути наступним:

- Монітор на робочому місці встановлюється так, щоб верхній край екрана знаходився на рівні очей.

- Розташування монітора ПК має забезпечувати безпечність роботи в цілому і зручність та ефективність зорової роботи з екраном в вертикальній площині під кутом ± 300 від лінії зору, площина екрана при цьому має бути перпендикулярною нормальній лінії зору користувача.

Вимоги безпеки під час експлуатації, обслуговування, ремонту й налагодження ПЕОМ наступні:

- Щодня перед початком роботи необхідно проводити очищення екрана відеотермінала від пилу й інших забруднень.

- Після закінчення роботи ПЕОМ і периферійні пристрої повинні бути відключені від електричної мережі, а при виникненні аварійної ситуації відключення необхідно виконати негайно.

- Монтаж, підключення й відключення кабелів, ремонт ПЕОМ варто виконувати тільки при повністю відключеному живленні.

- При необхідності виконання робіт при включеному живленні роботи повинні виконуватися не менш чим двома працівниками, використовувати інструмент із ізоляційними ручками й стояти на діелектричному килимку.

- При виконанні ремонтних робіт варто користуватися електроінструментом з номінальною напругою не більше 36 В.

- Забороняється виконання ремонтних робіт з ручними годинниками з металевим браслетом

Відповідно вимоги до профілактичних оглядів та обов’язкові медичні огляди:

- попередні – під час оформлення на роботу;

- періодичні – протягом трудової раз у два роки комісією в складі терапевта, невропатолога й офтальмолога;

- Жінки, що працюють з ВДТ ПЕОМ обов’язково оглядаються лікарем акушером-гінекологом один раз на два роки;

- Жінки з часу встановлення вагітності та в період годування дитини грудьми до виконання всіх робіт, пов’язаних з використання ВДТ ПЕОМ не допускаються.

Дотримання санітарно-гігієнічних вимог до приміщень, де розташовані робочі місця з відеотерміналами, є критично важливим для забезпечення здоров'я та продуктивності працівників. Належне освітлення без відблисків запобігає перенапруженню очей та підвищує концентрацію, що безпосередньо впливає на якість виконання робочих завдань.

Оптимальні параметри мікроклімату, такі як температура, вологість і циркуляція повітря, створюють комфортні умови для працівників, знижуючи ризик розвитку захворювань, пов'язаних з роботою в несприятливих умовах. Система опалення та вентиляції забезпечує стабільний повітрообмін і комфортну температуру протягом усього робочого дня.

Ергономічні характеристики робочого місця є ключовими для запобігання професійним захворюванням, пов'язаним з тривалим сидінням і неправильним положенням тіла. Правильне розташування робочих місць, достатня площа та об'єм приміщення, а також регулярне вологе прибирання сприяють створенню безпечного та здорового робочого середовища. Заборона розміщення робочих місць у підвалах та цокольних приміщеннях підкреслює важливість забезпечення адекватного природного освітлення та вентиляції.

Таким чином, виконання цих норм є необхідним для створення сприятливих умов праці, що зменшує ризики для здоров'я працівників і підвищує їхню ефективність.

5.3 Розрахунок штучного освітлення виробничих приміщень

Розрахунок системи загального рівномірного освітлення з люмінісцентними лампами (світильники типу ЛПО 01 з двома лампами ЛБ–40) для виробничого приміщення, в якому виконуються зорові роботи високої точності (розряд IIIв, Е=300 лк). Розміри приміщення: довжина а=4 м; ширина b=4 м; висота Н=2.7 м. Приміщення має світлу побілку: коефіцієнти відбиття стелі Рстелі=30%, стін Рстін=10%. Коефіцієнт запасу, що враховує зниження освітленості в результаті забруднення та старіння ламп Кз=1.5; коефіцієнт нерівномірності освітлення Z=1.12.

Дане приміщення повинне мати природне і штучне освітлення відповідно до СНиП II-4-79 «Природнє та штучне освітлення». Природне світло повинно проходити через бічні світлопрорізи, зорієнтовані, як правило, на північ чи північний схід, і забезпечувати коефіцієнт природної освітленості (КПО) не нижче 1.5%. Розрахунки КПО проводяться так само відповідно до СНиП 11-4-79.

1. Загальна площа приміщення:

$$S = a * b = 4 * 4 = 16\text{м}^2$$

2. Необхідний світловий потік:

$$F_{\text{необх.}} = \frac{E * S * K_З * Z}{\eta}$$

де:

E =300 лк - необхідна освітленість,

S = 16м^2 - площа приміщення,

Кз = 1.5 — коефіцієнт запасу,

Z=1.12 - коефіцієнт нерівномірності освітлення,

η - світлова віддача (ефективність) системи освітлення (для ЛПО 01 з двома лампами ЛБ-40 η≈80 лм/Вт).

3. Розрахунок світлового потоку від однієї лампи:

Кожна лампа ЛБ-40 має потужність 40 Вт. Таким чином, світловий потік від однієї лампи:

$$F_{\text{лампи}} = 40\text{Вт} * 80\text{лм/Вт} = 3200\text{лм}$$

Оскільки у світильнику ЛПО 01 встановлено дві лампи:

$$F_{\text{світильника}} = 3200\text{лм} * 2 = 6400\text{лм}$$

4. Кількість світильників

$$N = \frac{F_{\text{необх.}}}{F_{\text{світильника.}}}$$

Підставляємо значення:

$$F_{\text{необх.}} = \frac{300\text{лк} * 16^2 * 1.5 * 1.12}{80} = \frac{8064}{80} = 100.8\text{лм}$$

5. Формула для кількості світильників:

$$N = \frac{100.8}{6400} = \frac{8064}{6400} \approx 1.26$$

Отже, для забезпечення необхідного рівня освітленості у виробничому приміщенні необхідно встановити 2 світильники типу ЛПО 01 з двома лампами ЛБ-40.

Коефіцієнт природної освітленості (КПО) не повинен бути нижче 1.5%. Природне світло має проникати через бічні світлопрорізи, зорієнтовані на північ чи північний схід. Розрахунки КПО проводяться відповідно до СНиП 11-4-79, враховуючи розташування вікон, їх площу та освітленість зовнішнього середовища.

ВИСНОВКИ

Під час виконання даної кваліфікаційної роботи було успішно розроблено веб-додаток "Space sink" для діяльності детейлінг студії "Space sink" у формі веб-сторінки з можливістю оформлення замовлень. Веб-додаток було розроблено відповідно до вимог технічного завдання та узгоджених вимог замовника.

Аналіз предметної області дозволив:

- визначити актуальність поставленої задачі та необхідність розробки веб-додатку;

- дослідити та проаналізувати аналогічні існуючі сайти на ринку;

- сформулювати мету та завдання дипломного проекту для використання розробленого продукту;

- провести детальний аналіз існуючих технологій та вибрати найбільш раціональний і сучасний підхід до реалізації веб-додатку відповідно до визначених вимог.

На основі проведеного аналізу предметної області та існуючих веб-додатків було сформовано перелік функціональних та нефункціональних вимог до розробленого додатку. Це дозволило створити зручний інструмент для ведення бізнесу як з точки зору підприємця, так і з точки зору клієнта, забезпечуючи їм ефективне взаємодію та комфортне користування.

Крім того, розроблений веб-додаток "Space sink" відповідає сучасним стандартам та вимогам до веб-технологій, що забезпечує його стабільну та надійну роботу. Важливим аспектом розробки була увага до інтерфейсу користувача, щоб забезпечити зручність та інтуїтивно зрозуміле використання додатку для всіх категорій користувачів. Таким чином, розроблений веб-додаток "Space sink" став результатом комплексного підходу до вирішення завдань і відповідає вимогам сучасного ринку та потребам клієнтів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бази даних PostgreSQL. URL: http://ukrhosting.ua/bazi_danih_ postgresql-p-263951.html Дата доступу: 01.05.2024

2. Джон Пакстон, Расс Фергюсон, Джон Резиг. JavaScript для професіоналів. Видавництво: Диалектика-Вильямс, 2018.

3. Методичні вказівки до дипломного проектування. URL: https://eguru1.tk.te.ua/pluginfile.php/76350/mod_resource/content/1/Metod_vkazi v_DP.pdf Дата доступу: 03.05.2024

4. Ознайомлення з Node.js. URL: https://armedsoft.com/ua /blog/oznayomlennya-z-nodejs Дата доступу: 04.05.2024

5. Способи і засоби пожежогасіння. URL: http://cde.nuft.edu.ua/mod/ book/view.php?id=430290&chapterid=260207 Дата доступу: 08.05.2024

6. Посібник: знайомство з React. URL: https://uk.reactjs.org/tutorial/ tutorial.html Дата доступу: 09.05.2024

7. Різниця між MySQL та PostgreSQL. URL: https://uk.living-in-belgium.com/difference-between-mysql-and-postgresql-155 Дата доступу: 10.05.2024

8. Створення сайтів і додатків на REACT.JS: швидко, професійно, під ключ. URL: https://brander.ua/technologies/reactjs Дата доступу: 12.05.2024

9. Що таке node.js?. URL: https://uk.theastrologypage.com/node-js Дата доступу: 14.05.2024

10. Що таке React? URL: https://uk.education-wiki.com/9050114-what-is-react Дата доступу: 15.05.2024

11. Cross-Origin Resource Sharing. URL: https://uk.wikipedia.org/wiki/Cross-Origin_Resource_Sharing Дата доступу: 16.05.2024

12. Express. URL: https://nodejsdev.ru/doc/express/ Дата доступу: 18.05.2024

13. Express documentation. URL: https://expressjs.com/en/5x/api.html Дата доступу: 19.05.2024

14. GARAGE detailinglab. URL: https://garage-detailinglab.kiev.ua/ language/ uk/contact-ua Дата доступу: 20.05.2024

15. IMPERIAL GARAGE. URL: https://detailing-lviv.com/#rec327595679 Дата доступу: 21.05.2024

16. PostgreSQL 9.3 матеріали для преси. URL: https://www.postgresql.org/about/ press/presskit93/ua/ Дата доступу: 22.05.2024

17. Sequelize. URL: https://metanit.com/web/nodejs/9.1.php Дата доступу: 23.05.2024

18. SMART. URL: https://smartavto.org/?lang=uk Дата доступу: 24.05.2024

19. What is PERN Stack? URL: https://www.geeksforgeeks.org/what-is-pern-stack/ Дата доступу: 26.05.2024

ДОДАТКИ

Додаток А – код клієнтської частини веб-сайту

// index.js

```
import React, {createContext} from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import UserStore from './store/UserStore';
import ServiceStore from './store/ServiceStore';
import InterStore from './store/InterStore';
export const Context = createContext(null)
ReactDOM.render(
    <Context.Provider value={{
        user: new UserStore(),
        service: new ServiceStore(),
        inter: new InterStore()
    }}>
        <App />
    </Context.Provider>,
  document.getElementById('root')
);
```

//app.js

```
import React, { useContext, useEffect, useState } from 'react';
import {BrowserRouter} from 'react-router-dom';
import AppRouter from './components/AppRouter';
import Footer from './components/Footer';
import NavBar from './components/NavBar';
import { observer } from 'mobx-react-lite';
```

```
import { Context } from './index';
import { check } from './http/userAPI';
import { Spinner } from 'react-bootstrap';
const App = observer( () => {
  const {user} = useContext(Context)
  const [loading, setLoading] = useState(true)
  useEffect( () => {
    check().then(data => {
      user.setUser(true)
      user.setIsAuth(true)
      user.setIsAdmin(true)
    }).finally(() => setLoading(false))
  })
  if (loading) {
    return <Spinner animation={"grow"}/>
  }
  return (
    <BrowserRouter>
      <NavBar/>
       <AppRouter />
       <Footer />
    </BrowserRouter>
  );
})
export default App;
//router.js
import Admin from "./pages/Admin"
import Auth from "./pages/Auth"
import Main from "./pages/Main"
```

```
import Service from "./pages/Service"
import AboutAuthor from "./pages/AboutAuthor"
import ServicePage from "./pages/ServicePage"
import{ ADMIN_ROUTE, REGISTRATION_ROUTE, SERVICE_ROUTE,
LOGIN_ROUTE,MAIN_ROUTE,ABOUTAUTHOR_ROUTE,SERVICEPAGE_
ROUTE  } from "./utils/consts"
export const authRoutes = [
  {
    path:  ADMIN_ROUTE, Component: Admin
  }
]
export const publicRoutes = [
  {
    path: SERVICE_ROUTE, Component: Service
  },
  {
    path:  REGISTRATION_ROUTE, Component: Auth
  },
  {
    path:   LOGIN_ROUTE, Component: Auth
  },
  {
    path:   ABOUTAUTHOR_ROUTE, Component: AboutAuthor
  },
  {
    path:   SERVICEPAGE_ROUTE + '/:id', Component: ServicePage
  },
  {
    path:   MAIN_ROUTE, Component: Main
  }
]
```

```
//AppRouter.js
import React, { useContext } from 'react';
import {Switch, Route, Redirect} from 'react-router-dom';
import {authRoutes, publicRoutes} from '../routes'
import { MAIN_ROUTE } from '../utils/consts';
import { Context } from '../index';
const AppRouter = () => {
   const {user} = useContext(Context)
   return  (
      <Switch>
        {user.isAuth && authRoutes.map(({path, Component}) =>
          <Route    key={path}    path={path}    component={Component}
exact/>
        )}
        {publicRoutes.map(({path, Component}) =>
          <Route    key={path}    path={path}    component={Component}
exact/>
        )}
        <Redirect to={MAIN_ROUTE}/>
      </Switch>
   )
}
export default AppRouter;
// consts.js
export const ADMIN_ROUTE = '/admin'
export const LOGIN_ROUTE = '/login'
export const REGISTRATION_ROUTE = '/registration'
export const SERVICE_ROUTE = '/service'
export const MAIN_ROUTE = '/main'
export const SERVICEPAGE_ROUTE = '/servicepage'
export const ABOUTAUTHOR_ROUTE = '/aboutauthor'
```

```
//interStore.js
import {makeAutoObservable} from "mobx"
export default class InterStore {
    constructor() {
        this._inters = []
        makeAutoObservable(this); }
    setInters(inters) {
        this._inters = inters; }
    get inters() {
        return this._inters} }
//ServiceStore.js
import {makeAutoObservable} from "mobx"
export default class ServiceStore {
    constructor() {
        this._services = []
        makeAutoObservable(this); }
    setServices(services) {
        this._services = services; }
    get services() {
        return this._services} }
//userStore.js
import {makeAutoObservable} from "mobx"
export default class UserStore {
    constructor() {
        this._isAuth = true;
        this._isAdmin = true;
        this._user = {}
        makeAutoObservable(this); }
    setIsAuth(bool) {
        this._isAuth = bool; }
    setIsAdmin(bool) {
```

```
            this._isAdmin = bool; }
        setUser(user) {
            this._user = user; }
        get isAuth() {
            return this._isAuth}
        get isAdmin() {
            return this._isAdmin}
        get user() {
            return this._user} }
    //NavBar.js
    import React, { useContext } from "react";
    import {Context} from "../index"
    import Nav from "react-bootstrap/Nav"
    import Navbar from "react-bootstrap/Navbar"
    import {Button} from "react-bootstrap"
    import Container from 'react-bootstrap/Container'
    import { observer } from "mobx-react-lite";
    import {useHistory} from "react-router-dom"
    import { ABOUTAUTHOR_ROUTE, ADMIN_ROUTE, LOGIN_ROUTE,
SERVICE_ROUTE, MAIN_ROUTE } from "../utils/consts";
    const NavBar = observer (() => {
        const {user} = useContext(Context);
        const history = useHistory()
        const logOut = () => {
            user.setUser({})
            user.setIsAuth(false)
            user.setIsAdmin(false) }
        return (
            <Navbar bg="dark" variant="dark">
            <Container>
            <Navbar.Brand>YOUR DETAILING</Navbar.Brand>
```

```
                {user.isAuth ?
                <Nav className="ms-auto" style={{color:"white"}}>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(MAIN_ROUTE)}>Головна</Button>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(SERVICE_ROUTE)}>Послуги</Button>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(ABOUTAUTHOR_ROUTE)}>Про автора</Button>
                    {user.isAdmin ? <Button variant={"outline-light"}
className="ms-2" onClick={() => history.push(ADMIN_ROUTE)}>Адмін-
панель</Button> : <div> </div> }
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => logOut()}>Вийти</Button>
                    </Nav>
                    :
                    <Nav className="ms-auto" style={{color:"white"}}>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(MAIN_ROUTE)}>Головна</Button>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(SERVICE_ROUTE)}>Послуги</Button>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(ABOUTAUTHOR_ROUTE)}>Про автора</Button>
                    <Button variant={"outline-light"} className="ms-2"
onClick={() => history.push(LOGIN_ROUTE)}>Авторизація</Button>
                </Nav>}
            </Container>
        </Navbar>)})
    export default NavBar;
    //Footer.js
    import React from "react"
    import Image from "react-bootstrap/Image"
```

```
import call from "../assets/call.png"
import map from "../assets/map.png"
const Footer = () => <footer className="page-footer font-small blue" >
  <div className="footer-copyright text-center py-2" style={{backgroundColor: "#212529", color: "#ffffff"}}>
    © 2022 Copyright: Vitalii Piontkovskiy <Image width="25px" src={call}/> +38 (096) 931 64 10 <Image width="25px" src={map}/>проспект Степана Бандери, 157, Тернопіль
  </div>
</footer>
export default Footer
```

//http/index.js

```
import axios from "axios";
const $host = axios.create({
  baseURL: process.env.REACT_APP_API_URL})
const $authHost = axios.create({
  baseURL: process.env.REACT_APP_API_URL})
const authInterceptor = config => {
  config.headers.authorization = `Bearer ${localStorage.getItem('token')}`
  return config}
$authHost.interceptors.request.use(authInterceptor)
export {$host, $authHost}
```

//serviceAPI.js

```
import {$authHost, $host } from "./index";
export const createService = async (service) => {
  const {data} = await $authHost.post('api/service', service)
  return data}
export const fetchServices = async () => {
  const {data} = await $host.get('api/service')
  return data}
```

```
export const fetchOneService = async (id) => {
    const {data} = await $host.get('api/service/' + id)
    return data}
export const createInter = async (inter) => {
    const {data} = await $authHost.post('api/inter', inter)
    return data}
export const fetchInter = async () => {
    const {data} = await $host.get('api/inter')
    return data}
export const fetchOneInter = async () => {
    const {data} = await $host.delete('api/inter')
    return data}
```

//userAPI.js

```
import {$authHost, $host } from "./index";
import jwt_decode from "jwt-decode";
export const registration = async (email, password) => {
    const {data} = await $host.post('api/user/registration', {email, password,
role: 'USER'})
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token)}
export const login = async (email, password) => {
    const {data} = await $host.post('api/user/login', {email, password})
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token)}
export const check = async () => {
    const {data} = await $authHost.get('api/user/auth',)
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token)}
```

Додаток Б – код серверної частини веб-сайту

//index.js

```
require('dotenv').config()
const express = require('express');
const sequelize = require('./db')
const models = require('./models/models')
const PORT = process.env.PORT || 5000;
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const errorHandler = require('./middleware/errorHandlingMiddleware')
const path = require('path')
const app = express();
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use('/api', router)
//Обробка помилок, останній Middleware
app.use(errorHandler)
const start = async () => {
   try{
      await sequelize.authenticate()
      await sequelize.sync()      //функція буде звіряти стан бази данних зі
схемою данних
      app.listen(PORT, () => console.log(`Server started on port ${PORT}`))
   } catch (e) {
      console.log(e) }}start()
```

```
//db.js
const {Sequelize} = require('sequelize')
module.exports = new Sequelize(
   process.env.DB_NAME, //Передача назви бази данних
   process.env.DB_USER, //Передача назви користувача
   process.env.DB_PASSWORD, //Передача пароля користувача
   { dialect: 'postgres', host: process.env.DB_HOST, port:
process.env.DB_PORT })
//.env
PORT = 5000
DB_NAME=services
DB_USER=postgres
DB_PASSWORD=1987
DB_HOST=localhost
DB_PORT=5432
SECRET_KEY=random_secret_key123
//routes/index.js
const Router = require('express')
const router = new Router()
const serviceRouter = require('./serviceRouter')
const userRouter = require('./userRouter')
const interRouter = require('./interRouter')
router.use('/user', userRouter)
router.use('/service', serviceRouter)
router.use('/inter', interRouter)
module.exports = router
//interRouter.js
const Router = require('express')
const router = new Router()
const interController = require('../controllers/interController')
```

```
router.post('/', interController.create)
router.get('/', interController.getAll)
router.delete('/', interController.getOne)
module.exports = router
```

//serviceRouter.js

```
const Router = require('express')
const router = new Router()
const serviceController = require('../controllers/serviceController')
const checkRole = require('../middleware/checkRoleMiddleware')
router.post('/',checkRole('ADMIN'), serviceController.create)
router.get('/', serviceController.getAll)
router.get('/:id', serviceController.getOne)
module.exports = router
```

//userRouter.js

```
const Router = require('express')
const router = new Router()
const userController = require('../controllers/userController')
const authMiddleware = require('../middleware/authMiddleware')
router.post('/registration', userController.registration)
router.post('/login', userController.login)
router.get('/auth', authMiddleware, userController.check)
module.exports = router
```

//models.js

```
const sequelize = require('../db')
const {DataTypes} = require('sequelize')
const User = sequelize.define( 'user', {
   id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
   email: {type:DataTypes.STRING, unique: true},
   password: {type:DataTypes.STRING},
```

```
    role:{type:DataTypes.STRING, defaultValue: "USER"}})
  const Inter = sequelize.define( 'inter', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
    nameUser: {type:DataTypes.STRING},
    data: {type:DataTypes.STRING},
    time:{type:DataTypes.STRING},
    classCar: {type:DataTypes.STRING},
    nameService: {type:DataTypes.STRING},
    userId: {type:DataTypes.INTEGER}})
  const Service = sequelize.define( 'service', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:true},
    name: {type: DataTypes.STRING, allowNull: true, unique: true},
    price: {type: DataTypes.INTEGER, allowNull: true},
    img: {type: DataTypes.STRING},
    description: {type: DataTypes.STRING}})
  User.hasMany(Inter)
  Inter.belongsTo(User)
  module.exports = {User, Service, Inter}
  // authMiddleware.js
  const jwt = require('jsonwebtoken')
  module.exports = function (req, res, next) {
    if (req.method === "OPTIONS") {next()}
    try{const token = req.headers.authorization.split(' ')[1] //Bearer token...
      if (!token) {return res.status(401).json({message: "Користувач не
авторизований"})}
      const decoded = jwt.verify(token, process.env.SECRET_KEY)
      req.user = decoded next() } catch (e) {
      res.status(401).json({message: "Користувач не авторизований"})
    }}
```

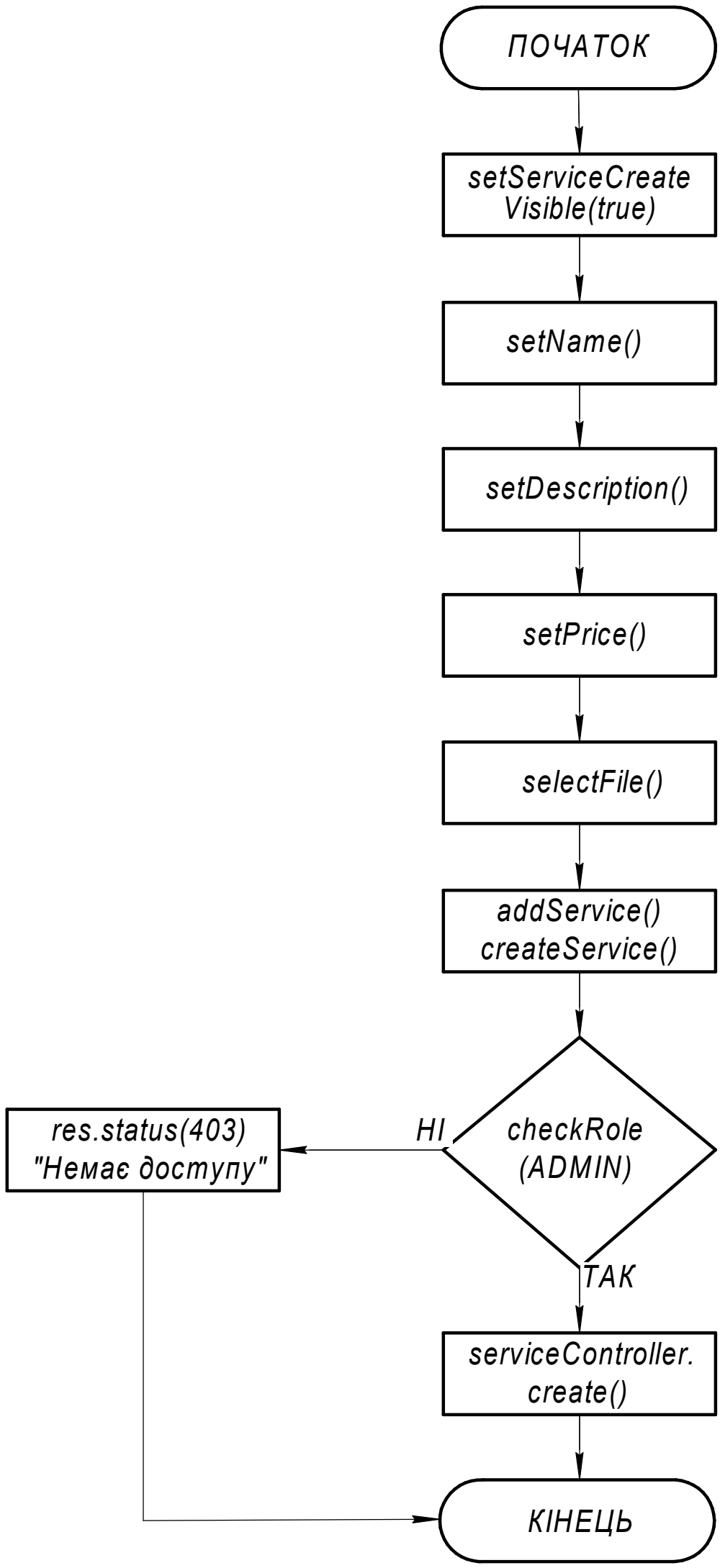
```
// checkRoleMiddleware.js
const jwt = require('jsonwebtoken')
module.exports = function (role) {
    return  function (req, res, next) {
        if (req.method === "OPTIONS") {
            next()}
        try{
            const token = req.headers.authorization.split(' ')[1] //Bearer token...
            if (!token) {
                return res.status(401).json({message: "Користувач не
авторизований"})}
            const decoded = jwt.verify(token, process.env.SECRET_KEY)
            if(decoded.role !== role) {
                return res.status(403).json({message: "Немає доступу"})}
            req.user = decoded next()
        } catch (e) {
            res.status(401).json({message: "Користувач не авторизований"})
        }}}
// errorHandlingMiddleware.js
const { json } = require('sequelize/dist')
const ApiError = require('../error/ApiError')
module.exports = function(err, req, res, next){
    if(err instanceof ApiError) {
        return res.status(err.status).json({message: err.message})}
    return res.status(500).json({message: "Непередбачування помилка"}) }
// ApiError.js
class ApiError extends Error {
    constructor(status, message) {
        super();
```

```
        this.status = status;
        this.message = message; }
    static badRequest(message) {
        return new ApiError(404, message); }
    static internal(message) {
        return new ApiError(500, message); }
    static forbidden(message) {
        return new ApiError(403, message); }}
module.exports = ApiError;
// interController.js
const uuid = require('uuid')
const path = require('path')
const {Inter} = require('../models/models')
const ApiError = require('../error/ApiError')
class interController {
    async create(req, res, next){ try {
            const {nameUser, data, time, classCar, nameService} = req.body
            const inter = await Inter.create({nameUser, data, time, classCar,
nameService}) return res.json(inter)
        } catch (e) {
            next(ApiError.badRequest(e.message)) }      }
    async getAll(req, res){
        let inter = await Inter.findAll()
        return res.json(inter) }
    async getOne(req, res){
        const {id} = req.params
        const inter = await Inter.findOne(
            {where: {id}}) return res.json(inter) } }
module.exports = new interController()
```

```
// serviceController.js
const uuid = require('uuid')
const path = require('path')
const {Service} = require('../models/models')
const ApiError = require('../error/ApiError')
class serviceController {
    async create(req, res, next){ try {
        const {name, price, description} = req.body
        const {img} = req.files
        let filename = uuid.v4() + ".jpg"
        img.mv(path.resolve(__dirname, '..', 'static', filename))
        const service = await Service.create({name, price, description,
img:filename}) return res.json(service)
    } catch (e) {
        next(ApiError.badRequest(e.message)) }}
    async getAll(req, res){
        let service = await Service.findAll()
        return res.json(service) }
    async getOne(req, res){
        const {id} = req.params
        const service = await Service.findOne({where: {id}})
            return res.json(service) } }
module.exports = new serviceController()
// userController.js
const ApiError = require('../error/ApiError');
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {User, Basket} = require('../models/models')
const generateJwt = (id, email, role) => {
    return jwt.sign(
```

```
            {id, email, role},
            process.env.SECRET_KEY,
            {expiresIn: '24h'})}
        class UserController {
            async registration(req, res, next){
                const {email, password, role} = req.body
                if(!email || !password) {
                    return next(ApiError.badRequest('Некоректно введено емайл або
пароль')) } const candidate = await User.findOne({where: {email}})
                if(candidate) {
                    return next(ApiError.badRequest('Користувач з таким email уже
існує')) }
                const hashPassword = await bcrypt.hash(password, 5) // хешування
пароля const user = await User.create({email, role, password: hashPassword}) //
передача данних в обробник події, створення акаунта
                const basket = await Basket.create({userId: user.id}) // створення
моделі корзини
                const token = generateJwt(user.id, user.email, user.role)
                return res.json({token})}
            async login(req, res, next) {
                const {email, password} = req.body
                const user = await User.findOne({where: {email}})
                if (!user) {
                    return next(ApiError.internal('Користувача з таким email не існує')) }
                let comparePassword = bcrypt.compareSync(password, user.password)
                if (!comparePassword) {
                    return next(ApiError.internal('Невірно вказаний пароль')) }
                const token = generateJwt(user.id, user.email, user.role)
                return res.json({token})}
            async check(req, res, next){
                 const token = generateJwt(req.user.id, req.user.email, req.user.role)
                 return res.json({token})}}
        module.exports = new UserController()
```

Алгоритм роботи підпрограми



					2024.КРБ.123.602.20.00.00 АР			
						Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка вебсайту центру "Space sink" Алгоритм роботи підпрограми			
Розроб.		В. Пйонтковський				Аркуш	Аркушів 1	
Перевір.		В. Штокало				ВСП "ТФК ТНТУ" гр. КІ-602 м. Тернопіль		
Консульт.								
Реценз.								
Н. контр.		В. Приймак						
Затв.								

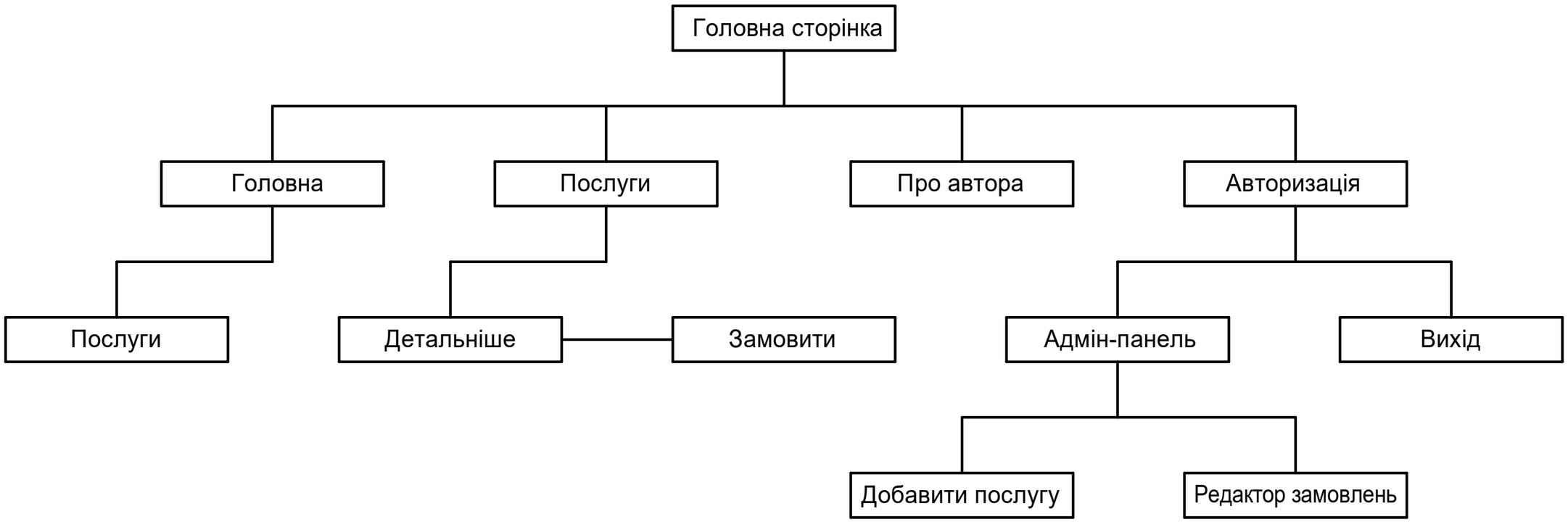
Скрипт роботи підпрограми

```
import React, {useState} from "react";
import { Button, Container } from "react-bootstrap";
import adminbg from "../assets/adminbg.jpg"
import CreateService from "../components/modals
/CreateService"
import ShowInterdiction from "../components/
modals/ShowInterdiction" const Admin = () => {
const [serviceCreatVisible, setServiceCreatVisible]
 = useState(false) const [serviceInterVisible,
setServiceInterVisible] = useState(false) return  (
<div style={{backgroundImage: "url(" + adminbg + ")",
backgroundSize: "cover", backgroundRepeat:
'no-repeat', backgroundPosition: 'center'}}>
<Container className="d-flex flex-column"
style={{height: window.innerHeight - 54}}>
<Button variant={"success"} className="mt-3
pt-2" onClick={() => setServiceCreatVisible(true)}>
Добавити послугу</Button> <Button variant=
{"success"} className="mt-3 pt-2" onClick={() =>
setServiceInterVisible(true)}>Редактор замовлень
</Button> <CreateService show={serviceCreat
Visible} onHide={() => setServiceCreatVisible(false)}/>
<ShowInterdiction show={serviceInterVisible} onHide=
{() => setServiceInterVisible(false)}/>
</Container> </div> ) } export default Admin;
import React, {useState}  from "react";
import Modal from "react-bootstrap/Modal"
import Button from "react-bootstrap/Button"
import {Form} from 'react-bootstrap'
import {createService} from '../../http/serviceAPI'
//Компонент CreateService.js
const CreateService = ({show, onHide}) => {
const [name, setName] = useState()
const [price, setPrice] = useState()
const [description, setDescription] = useState()
const [file, setFile] = useState(null)
const selectFile = e => { setFile(e.target.files[0])}
const addService = () => { const formData = new
FormData() formData.append('name', name) formData
.append('description', description) formData.append
('price', `${price}`) formData.append('img', file)
createService(formData).then(data => onHide()) }
return  ( <Modal show={show} onHide={onHide}
size="lg" aria-labelledby="contained-modal-title-
vcenter" centered > <Modal.Header closeButton>
<Modal.Title id="contained-modal-title-vcenter">
Добавити послугу</Modal.Title></Modal.Header>
<Modal.Body> <Form> <Form.Control placeholder=
{"Введіть назву послуги..."} className="mt-3"
value={name} onChange={e =>setName(e.target.value)}>
<Form.Control placeholder={"Введіть опис
послуги..."} className="mt-3" value={description}
 onChange={e =>setDescription(e.target.value)}/>
<Form.Control type="number"placeholder={"
Введіть ціну послуги..."} className="mt-3" value=
{price} onChange={e => setPrice(Number(e.target.
value))}/><Form.Label className="mt-3">Додайте
зображення послуги</Form.Label> <Form.Control
type="file" onChange={selectFile}/> </Form>
</Modal.Body> <Modal.Footer> <Button variant=
"outline-danger" onClick={onHide}>Закрити</Button
> <Button variant="outline-success" onClick={
addService}>Добавити</Button> </Modal.Footer>
</Modal> ) } export default CreateService;
// Компонент serviceAPI.js
import {$authHost, $host } from "./index";
export const createService = async (service) => {
const {data} = await $authHost.post('api/service',
service) return data }
// Компонент serviceRouter.js
const Router = require('express') const router =
new Router() const serviceController = require
('../controllers/serviceController') const
checkRole = require('../middleware/checkRole
Middleware') router.post('/',checkRole('ADMIN'),
serviceController.create) router.get('/', service
Controller.getAll) router.get('/:id', serviceController
.getOne) module.exports = router
// Компонент checkRoleMiddleware.js
const jwt = require('jsonwebtoken') module.exports
= function (role) {return  function (req, res, next) {
if (req.method === "OPTIONS") { next() } try{
const token = req.headers.authorization.split(' ')[1]
if (!token) {return res.status(401).json({message:
"Користувач не авторизований"}) }const decoded
= jwt.verify(token, process.env.SECRET_KEY)
if(decoded.role !== role) {return res.status(403).json
({message: "Немає доступу"}) } req.user = decoded
next() } catch (e) { res.status(401).json({message:
"Користувач не авторизований"}) } }
//Компонент serviceController.js
async create(req, res, next){ try {
const {name, price, description} = req.body
const {img} = req.files let filename = uuid.v4() + ".jpg"
img.mv(path.resolve(__dirname, '..', 'static', filename))
const service = await Service.create({name, price,
description, img:filename})return res.json(service) }
catch (e) { next(ApiError.badRequest(e.message)) }
```

					2024.КРБ.123.602.20.00.00 СР		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		В. Пйонтковський			Розробка вебсайту центру "Space sink" Скрипт роботи підпрограми	Літ.	Маса	Масштаб
Перевір.		В. Штокало						
Консульт.						Аркуш	Аркушів	
Реценз.								
Н. контр.		В. Приймак				ВСП "ТФК ТНТУ" гр. КІ-602 м. Тернопіль		
Затв.								

2024.КРБ.123.602.20.00.00 СС

Структурна схема веб-сайту



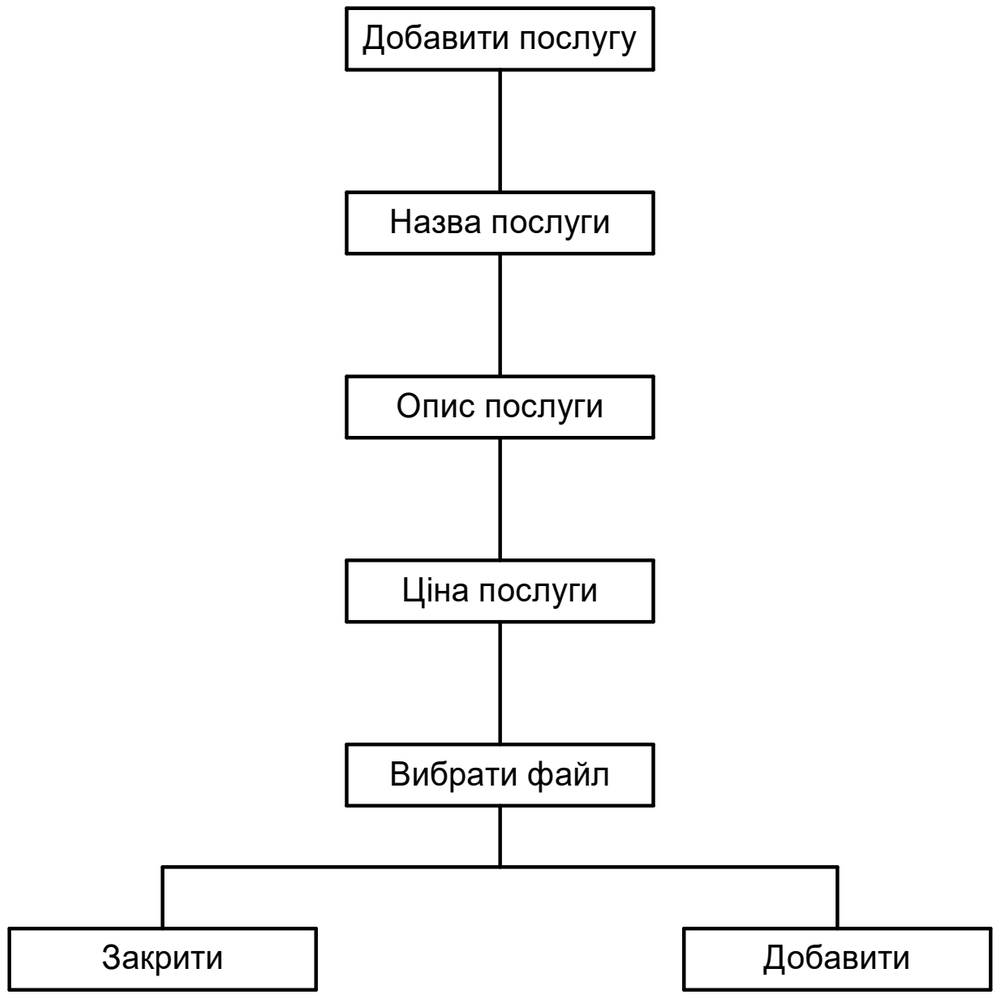
					2024.КРБ.123.602.20.00.00 СС		
						Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка вебсайту центру "Space sink" Структурна схема веб-сайту			
Розроб.		В. Пйонтковський				Аркуш	Аркушів	1
Перевір.		В. Штокало				ВСП "ТФК ТНТУ" гр. КІ-602 м. Тернопіль		
Консульт.								
Реценз.								
Н контр.		В. Приймак						
Затв.								

Таблиця техніко-економічних показників

Технічні показники			Економічні показники			
№ n/n	Показник	Значення	№ n/n	Показник	Одиниці вимірю-вання	Значення
1	Операційна система	Крос-платформенна	6	Собівартість сайту	грн.	13771,06
2	Загальний об'єм сайту	460 Мб	7	Плановий прибуток	грн.	*7711,79*
3	База даних	PostgreSQL 14.3	8	Ціна	грн.	*21482,85*
4	Базові мови програмування	JavaScript	9	Економічна ефективність	грн.	*0,55*
5	Фреймворки та платформи	Node.js, Express.js, React.js	10	Термін окупності	рік	1.8

Змн.	Арк.	№ докум.	Підпис	Дата	*2024.КРБ.123.602.20.00.00 ТБ*		
Розроб.		В. Пйонтковський			Розробка вебсайту центру "Space sink" Таблиця техніко-економічних показників	Літ.	Маса	Масштаб
Перевір.		В. Штокало						
Консульт.						Аркуш	Аркушів 1	
Реценз.						ВСП "ТФК ТНТУ" гр. КІ-602 м. Тернопіль		
Н контр.		В. ПРИЙМАК						
Затв.								

Структурна схема компонент фронтенду сайту



					2024.КРБ.123.602.20.00.00 СС	
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка веб-сторінки центру "Space sink" Структурна схема компонент форнтенду сайту	Літ. / Маса / Масштаб
Розроб.		В. Пйонтковський				
Перевір.		В. Штокало				Аркуш / Аркушів 1
Консульт.						
Реценз.						ВСП "ТФК ТНТУ" гр. КІ-602 м. Тернопіль
Н контр.		В. Приймак				
Затв.						

Текст програми

```
import React, {useState}  from "react";
import Modal from "react-bootstrap/Modal"
import Button from "react-bootstrap/Button"
import {Form} from 'react-bootstrap'
import {createService} from '../../http/serviceAPI'
const CreateService = ({show, onHide}) => {
const [name, setName] = useState()
const [price, setPrice] = useState()
const [description, setDescription] = useState()
const [file, setFile] = useState(null)
const selectFile = e => {
setFile(e.target.files[0]) }
const addService = () => {
const formData = new FormData()
formData.append('name', name)
formData.append('description', description)
formData.append('price', `${price}`)
formData.append('img', file)
createService(formData).then(data => onHide()) }
return  (<Moda lshow={show} onHide={onHide}
size="lg" aria-labelledby="contained-modal-title-
vcenter" centered > <Modal.Header closeButton>
<Modal.Title id="contained-modal-title-vcenter">
Добавити послугу </Modal.Title> </Modal.Header>
<Modal.Body> <Form> <Form.Control placeholder=
{"Введіть назву послуги..." } className="mt-3"
value={name} onChange={e => setName(e.target.
value)}/> <Form.Control placeholder={"Введіть
опис послуги..."}  className="mt-3" value=
{description} onChange= {e => setDescription
(e.target.value)}/> <Form.Control type="number"
 placeholder= {"Введіть ціну послуги..."}
className="mt-3" value= {price} onChange=
{e => setPrice(Number(e.target.value))}/> <Form.
Label className="mt-3">Додайте зображення
послуги</Form.Label> <Form.Control type="file"
onChange={selectFile}/> </Form> </Modal.Body>
<Modal.Footer> <Button variant="outline-danger"
 onClick= {onHide}>Закрити</Button> <Button
variant="outline-success" onClick={addService}
>Добавити</Button></Modal.Footer> </Modal> ) }
export default CreateService;
```

```
import {$authHost, $host } from "./index";
export const createService = async (service) => {
const {data} = await $authHost.post('api/service',
service)return data }
export const fetchServices = async () => {
const {data} = await $host.get('api/service')
return data }
export const fetchOneService = async (id) => {
const {data} = await $host.get('api/service/' + id)
return data }
export const createInter = async (inter) => {
const {data} = await $authHost.post('api/inter', inter)
return data }
export const fetchInter = async () => {
const {data} = await $host.get('api/inter')
return data }
export const fetchOneInter = async () => {
const {data} = await $host.delete('api/inter')
return data }
const uuid = require('uuid')
const path = require('path')
const {Service} = require('../models/models')
const ApiError = require('../error/ApiError')
class serviceController {
async create(req, res, next){
try { const {name, price, description} = req.body
const {img} = req.files
let filename = uuid.v4() + ".jpg"
img.mv(path.resolve(__dirname, '..', 'static', filename))
const service = await Service.create({name, price,
description, img:filename})
return res.json(service) } catch (e) {
next(ApiError.badRequest(e.message)) }
async getAll(req, res){
let service = await Service.findAll()
return res.json(service)}
async getOne(req, res){
const {id} = req.params
const service = await Service.findOne(
{where: {id}} )
return res.json(service) }
module.exports = new serviceController()
```

					2024.КРБ.123.602.20.00.00 ТП		
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка вебсайту центру "Space sink" Текст прогарми	Літ.	Маса / Масштаб
Розроб.		В. Пйонтковський					
Перевір.		В. Штокало				Аркуш	Аркушів
Консульт.						ВСП "ТФК ТНТУ" гр. КІ-602 м. Тернопіль	
Реценз.							
Н. контр.		В. Приймак					
Затв.							