МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії (ЗФН)
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Аліна САВЧЕНКО
"____" _____ 2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИЦІ ОСВІТНЬОГО СТУПЕНЯ «МАГІСТРА»
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ «ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

Тема: «Інформаційна система компанії будівництва басейнів»

Виконавиця: Мошковська Ірина Михайлівна

Керівник: к.т.н., доцент Райчев Ігор Едуардович

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії (ЗФН)

Кафедра комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Аліна САВЧЕНКО

"____" _____ 2021 р.

ЗАВДАННЯ

на виконання дипломної роботи студентки

Мошковської Ірини Михайлівни

(прізвище, ім'я, по батькові)

1. Тема дипломної роботи: «Інформаційна система компанії будівництва басейнів» затверджена наказом ректора від «12» жовтня 2021 р. за №2229/ст.

2. Термін виконання роботи: з 12 жовтня 2021 року по 31 грудня 2021 року

3. Вихідні дані до роботи: теоретичні відомості про інформаційні системи; створення вебсайтів; мова програмування JavaScript.

4. Зміст пояснювальної записки: аналіз існуючих програмних рішень та процес створення програмного забезпечення; розробка вебсайту.

5. Перелік обов'язкового графічного матеріалу: рисунки, діаграми

6. Календарний план-графік

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів	Підпис керівника
1	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт	12.10.2021 – 15.10.2021	
2	Огляд та аналіз відомих програмних технологій та засобів	16.10.2021 – 19.10.2021	
3	Написання розділу 1 дипломної роботи	20.10.2021 – 29.10.2021	
4	Визначення функціональних вимог інформаційної системи	30.10.2021 – 01.11.2021	
5	Проектування інформаційної системи, створення прототипу	02.11.2021 – 06.11.2021	
6	Розробка та програмна реалізація інформаційної системи	06.11.2021 – 19.11.2021	
7	Написання розділу 2 дипломної роботи	20.11.2021 – 25.11.2021	
8	Написання розділу 3 дипломної роботи. Завершення та створення пояснювальної записки дипломної роботи	26.11.2021 – 30.11.2021	
9	Оформлення та друк пояснювальної записки	08.12.2021 – 11.12.2021	
10	Створення презентації, доповіді, та підготовка до захисту дипломної роботи	12.12.2021 – 20.12.2021	

7. Дата видачі завдання: «12» жовтня 2021 р.

Керівник дипломної роботи: _____ Ігор РАЙЧЕВ
(підпис керівника)

Завдання прийняла до виконання: _____ Ірина МОШКОВСЬКА
(підпис випускниці)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Інформаційна система компанії будівництва басейнів» складається із вступу, трьох розділів, загальних висновків, списку використаних джерел і містить: 84 сторінки тексту, 82 рисунки, 2 діаграми, 1 таблицю та 7 джерел за посиланнями.

Об’єкт дослідження: компанія, яка займається будівництвом басейнів; її послуги.

Мета роботи: розробка інформаційної системи компанії будівництва басейнів на основі вебтехнологій, яка забезпечить можливість віддаленого доступу до системи клієнтам і співробітникам компанії в будь-який момент часу.

Методи дослідження: порівняльний аналіз, обробка літературних джерел.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, ВЕБСАЙТ, JAVASCRIPT, REACT, NEXTJS, SASS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

UI	інтерфейс користувача
API	інтерфейс прикладного програмування
HTML	мова гіпертекстової розмітки
CSS	формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки
DOM	об’єктна модель документа
BOM	об’єктна модель браузера
JSX	мова розширення JavaScript, яка дозволяє комбінувати JavaScript-код з розміткою, яка схожа на HTML
SPA	односторінковий додаток
URL	унікальна адреса веб ресурсу (сайту), яка зареєстрована в єдиній схемі адресації
MEAN	стек технологій: MongoDB, Express.js, Angular, Node.js
SEO	пошукова оптимізація
JSON	текстовий формат передачі даних

ВСТУП

На сьогодні Інтернет є засобом масової інформацій, який найбільш активно розвивається. Він все більше входить в різні сфери економіки та бізнесу. Вебсайт компанії є недорогим способом просування послуг більш широкому колу споживачів. У зв'язку з цим для будь-якого бізнесу важливо мати своє вебпредставництво, інтегроване в інформаційну систему компанії.

Створення інформаційної системи компанії на основі вебтехнологій забезпечує можливість віддаленого доступу до системи клієнтам і співробітникам компанії в будь-якій момент часу. За допомогою веборієнтованої системи на основі сайту відбувається швидка обробка запитів, оперативна видача інформації. Впровадження інформаційної системи на основі вебтехнологій забезпечує більш ефективну роботу підприємства за рахунок автоматизації основних бізнес-процесів при взаємодії з клієнтами, а також просування рекламних послуг в мережі для залучення нових клієнтів.

Архітектура веборієнтованої інформаційної системи компанії будується відповідно до вимог обліку користувальницьких характеристик: орієнтація на цілі бізнесу, надійність, гнучкість і здатність до адаптації, простота в освоєнні, прийнятна вартість. При цьому структура веборієнтованої інформаційної системи базується на онтологічній моделі бізнес-процесів, які в сукупності реалізують цільові стратегії підприємства.

У даній дипломній роботі буде розроблятись сайт для компанії, яка займається будівництвом та установкою басейнів.

Цілі створення:

- продаж послуг за допомогою сайту;
- налагодження зворотного зв'язку з клієнтами;
- підвищення лояльності аудиторії;
- впізнаваність компанії.

Для досягнення сформульованих цілей система має забезпечувати наступні можливості:

- публікація інформації про компанію та її послуги;
- публікація фотогалереї виконаних проектів;
- розрахунок приблизної вартості проекту;
- перегляд інформації та виконаних проектів;
- легкий зв’язок з компанією;
- підтримка багатомовності;
- доступ з різних пристроїв та браузерів.

Даний проект дасть змогу компанії заявити про себе, залучити нових клієнтів і, як результат, виведе компанію на новий рівень.

РОЗДІЛ 1. ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ПРОЕКТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1. Вибір архітектури

Клієнт-сервер – це популярна архітектура проектування програмного забезпечення, яка на абстрактному рівні розбиває програмне забезпечення на дві частини: сторона клієнта та сторона сервера.

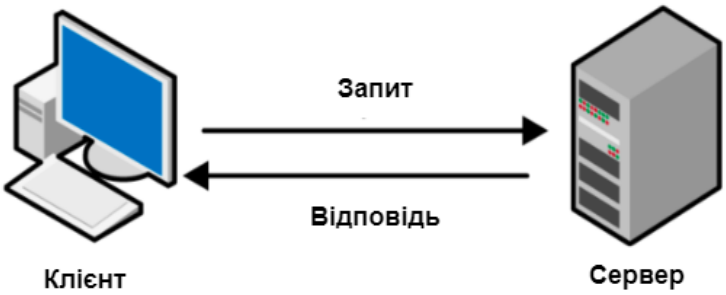


Рис. 1.1. Схема спрощеної структури клієнт-серверної архітектури

Клієнтська сторона (або просто клієнт) – це програма, яка працює на комп'ютері кінцевого користувача. Він надає UI, який відповідає за вигляд програми та її взаємодію з кінцевим користувачем. Клієнт може використовувати та споживати ресурси на машині користувача (обчислювальному пристрої), такі як тимчасове та локальне сховище тощо.

Серверна сторона (або просто сервер) – це програма, яка отримує запити від клієнтів і містить логіку для надсилання відповідних даних назад клієнту. Замість користувальницького інтерфейсу, сервер зазвичай має інтерфейс прикладного програмування. Більше того, сервер часто включає базу даних, яка буде постійно зберігати всі дані для програми.

Поки програмне забезпечення відповідає архітектурі клієнт-сервер, можна створювати будь-який UI під будь-яку платформу. Це вигідно, оскільки очікується, що сучасне програмне забезпечення буде доступне на багатьох платформах і забезпечить стабільний досвід роботи на всіх пристроях [1].

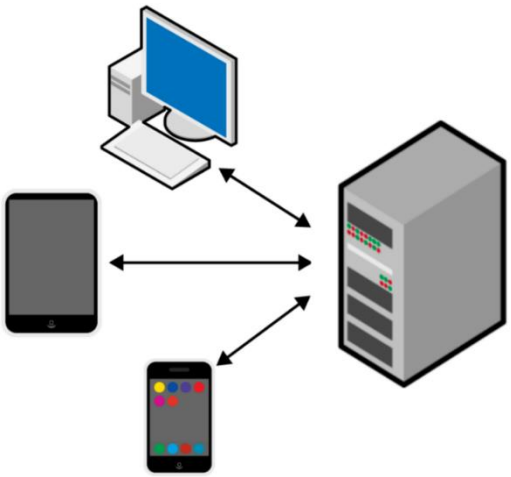


Рис. 1.2. Схема багатоплатформенної взаємодії

1.2. Вибір типу візуалізації

Візуалізація – це процес, який використовується у веброзробці та який перетворює HTML, CSS та JavaScript код у інтерактивні сторінки, які користувач бачить під час відвідування вебсайту.

Відповідно до архітектури з попереднього підрозділу існує два типи візуалізації: візуалізація на стороні клієнта (англ. Client-Side Rendering – CSR) та візуалізація на стороні сервера (англ. Server-Side Rendering – SSR). Хоча більшість користувачів Інтернету насправді не помічають великої різниці між SSR або CSR, для розробників це грає велику роль, оскільки візуалізація впливає на якість програмного забезпечення.

Розглянемо її різницю на прикладі:

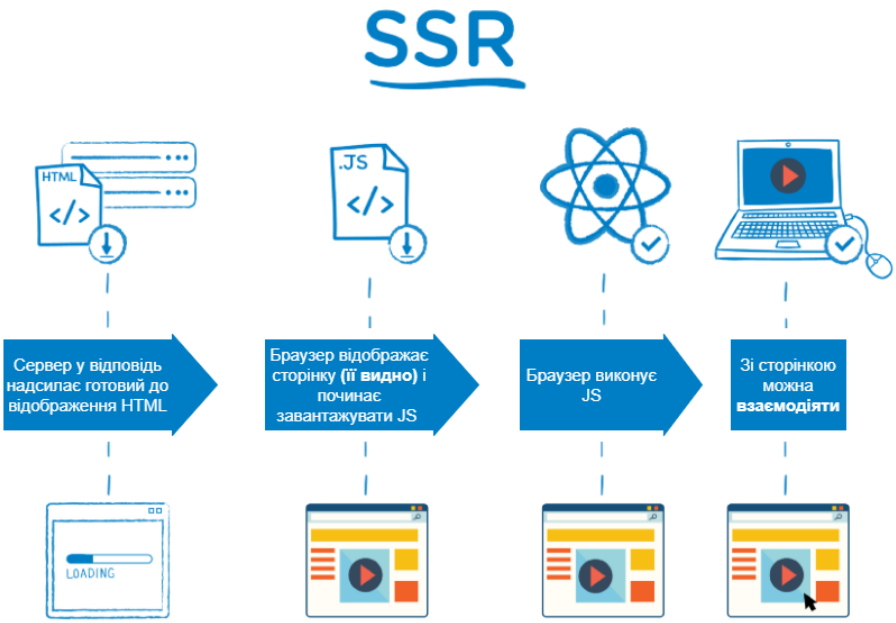


Рис. 1.3. Схема візуалізації на стороні сервера

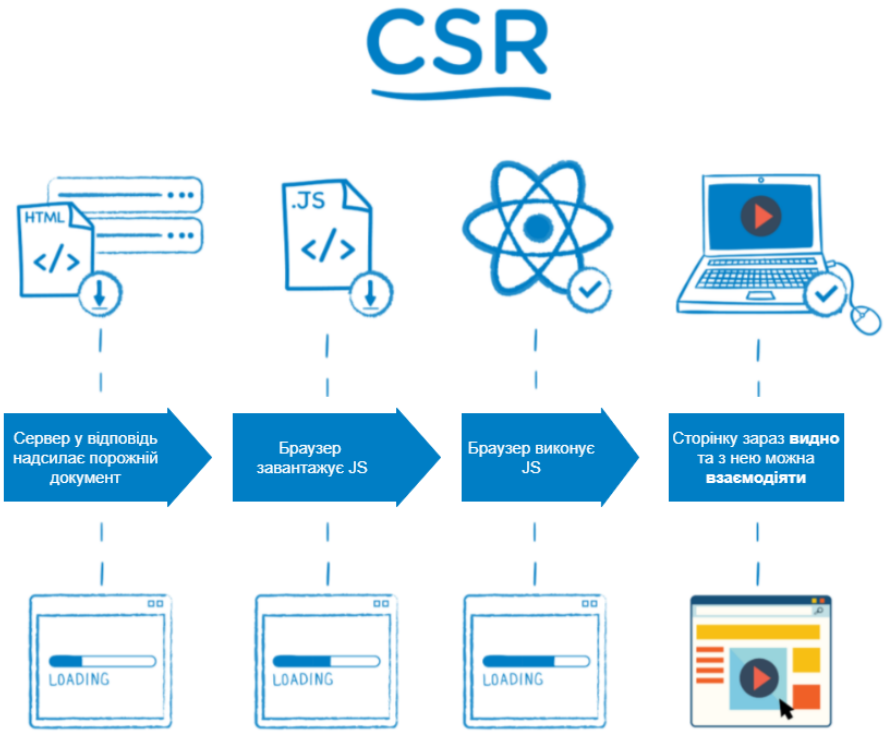


Рис. 1.4. Схема візуалізації на стороні клієнта

Основна відмінність полягає в тому, що при SSR відповіддю сервера на запит браузера є HTML сторінка, яка готова до відображення, тоді як при CSR браузер отримує досить порожній документ із посиланнями на JavaScript. Це означає, що при SSR браузер почне відтворювати HTML не чекаючи на JavaScript. Але в обох випадках JavaScript потрібно буде завантажити та

виконати: створити віртуальний DOM та приєднати до нього події. Це потрібно для того, щоб зробити сторінку інтерактивною. Але при SSR користувач може почати перегляд сторінки, поки все це відбувається, тоді ж як при CSR потрібно дочекатися, коли все вищесказане відбудеться, а потім перенести віртуальний DOM у веббраузер, щоб сторінка стала доступною для перегляду.

Крім часу завантаження, тип візуалізації впливає на SEO.

SEO (англ. Search engine optimization) – оптимізація пошукових систем – це практика надходження цільового трафіку на вебсайт з органічного (неоплачуваного) розділу пошукової системи.

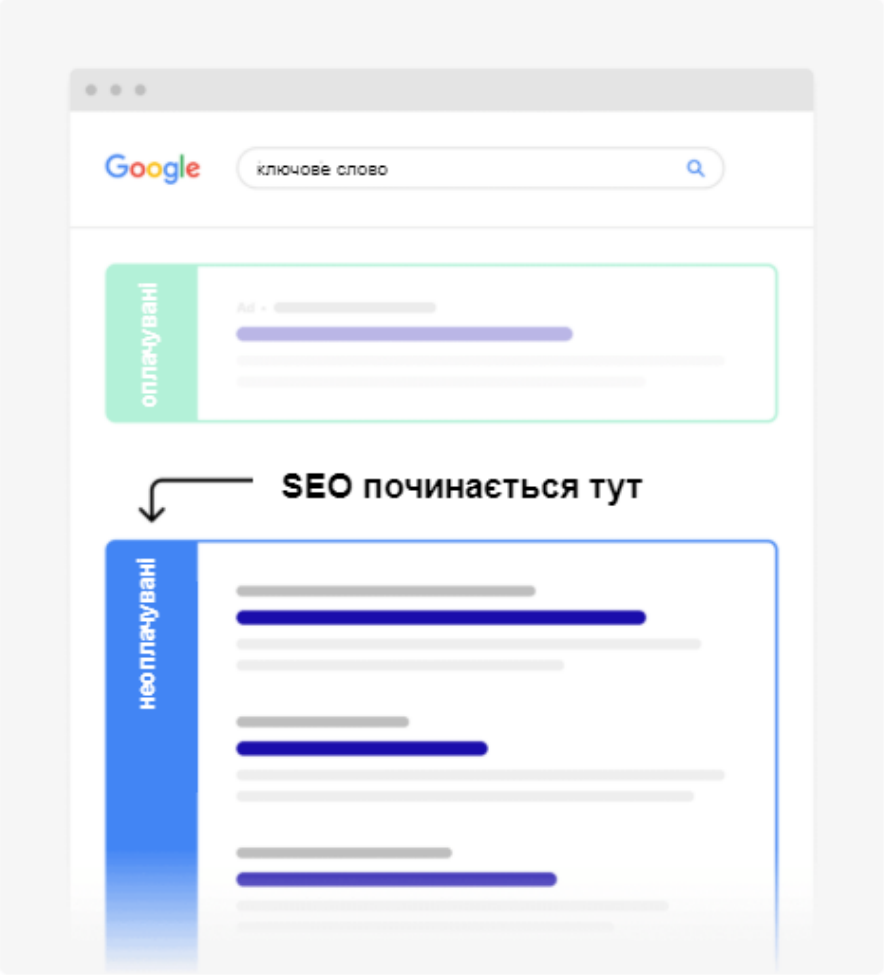


Рис. 1.5. Приклад розділів пошукової системи

Загальні завдання, пов'язані з SEO, включають створення високоякісного вмісту, оптимізацію вмісту навколо певних ключових слів та створення зворотних посилань для позиціонування вебсайту у верхніх результатах пошуку.

Під час використання CSR вміст сайту автоматично генерується за допомогою JavaScript. Це означає, що зміна метаданих з однієї сторінки на іншу залежить від виконання JavaScript. У свою чергу, це вимагає використання плагінів або бібліотечних модулів для встановлення метаданих для кожної сторінки, щоб вони відображалися на стороні клієнта.

Тим часом, за допомогою SSR, усі сторінки компілюються з правильними метаданими. Після отримання остаточного вмісту HTML вони надсилаються на front-end. Цей метод гарантує, що метадані сторінки залишаються точними незалежно від того, чи дозволяють сканери пошукових систем використовувати JavaScript чи ні. З огляду на це, сайти SSR більш сприятливі для SEO, ніж їх аналоги з CSR.

1.3. Мова програмування JavaScript

Розроблюваний інформаційний сайт буде реалізовано за допомогою мови програмування JavaScript, тому познайомимося з нею ближче.

JavaScript – це мова програмування, яка спочатку була розроблена для взаємодії з елементами вебсторінок. У веббраузерах JavaScript складається з трьох основних частин:

- ECMAScript – забезпечує основну функціональність;
- Об'єкта модель документа (DOM) – надає інтерфейси для взаємодії з елементами на вебсторінках;
- Об’єктна модель браузера (BOM) – надає API браузера для взаємодії з веббраузером.

JavaScript дозволяє додати інтерактивність до вебсторінки. Він часто використовується з HTML та CSS для покращення функціональних можливостей

вебсторінки, таких як перевірка форм, створення інтерактивних карт та відображення анімованих діаграм.

Коли вебсторінка завантажується, тобто після скачування HTML і CSS, двигун JavaScript у веббраузері виконує код JavaScript. Потім код JavaScript модифікує HTML і CSS для динамічного оновлення інтерфейсу користувача.

JavaScript може працювати не лише у веббраузерах, а й на серверах. Популярним серверним середовищем JavaScript є Node.js. На відміну від клієнтського JavaScript, серверний JavaScript виконується на сервері, що дозволяє отримати доступ до баз даних, файлових систем тощо [2].

Розглянемо переваги та недоліки цієї мови програмування.

Переваги:

- менші взаємодії з сервером – введені користувачем дані можна перевірити перед відправкою сторінки на сервер. Це економить серверний трафік, що означає менше навантаження на сервер;
- негайний зворотний зв’язок з відвідувачами – їм не потрібно чекати перезавантаження сторінки, щоб побачити, чи не забули вони щось ввести;
- підвищена інтерактивність – можна створювати інтерфейси, які реагують, коли користувач наводить на них курсор миші або активує їх за допомогою клавіатури;
- множина інтерфейсів – можна використовувати JavaScript, щоб включити такі елементи, як компоненти перетягування та повзунки, щоб надати розширений інтерфейс відвідувачам сайту.

Недоліки:

- клієнтський JavaScript не дозволяє читати або писати у файли. Це було зроблено з міркувань безпеки;
- JavaScript не можна використовувати для мережевих додатків, оскільки така підтримка відсутня;
- JavaScript не має багатопотокових або багатопроцесорних можливостей [3].

1.4. Вибір технології

Сучасні вебсайти найчастіше реалізовують за допомогою фреймворків, які базуються на JavaScript.

Фреймворк – це інструмент, який дозволяє розробляти програмне забезпечення та створювати системи. Сам цей інструмент являє собою набір готових функцій, компонент, а також заздалегідь визначену структуру проекту. Це спрощує процес розробки, оскільки програмістам не потрібно «винаходити велосипед» кожного разу, коли вони розробляють нову програму.

1.4.1. React.js

React.js вже третій рік поспіль займає першу сходинку у всіх рейтингах. Деякі розробники зовсім не мають нарікань в бік цього фреймворку, оскільки він стрімко розвивається і стає більш стабільним.

Цікаво, що насправді, React.js – це не фреймворк, а бібліотека. Але вона має настільки широкий функціонал, що найчастіше цей інструмент можна використовувати без додаткових інтеграцій. Front-end в результаті буде не менш ефектним.

React.js був створений командою Facebook в 2013 році з метою поділу призначеного для користувача інтерфейсу на набір компонентів, щоб спростити процес розробки.

Переваги:

- найбільша спільнота розробників;
- підтримка зі сторони Facebook;
- насичена екосистема;
- може використовуватись для нативних та вебдодатків.

Недоліки:

- перед використанням необхідно вивчити JSX;
- низькоякісна документація (розробники зазвичай пишуть її самі);
- забагато несподіваних оновлень.

React.js – дуже насичена середа розробки. Більшість користувачів відкликаються про неї позитивно. Однак, останнім часом Facebook випускає занадто багато оновлень, через що застарівають деякі інструменти. Деяким розробникам такий хід подій подобається. Але в більшості своїй, спільнота висловлює невдоволення з цього приводу. Адже таким чином їм необхідно постійно освоювати нові методи.

Одна з ключових причин популярності React.js – підтримка з боку авторитетної компанії Facebook. На базі цього середовища розробки сьогодні працюють такі сервіси, як Instagram, Whatsapp і Twitter. Всі вони дуже швидкі, привабливі зовні. Це забезпечує високий рівень довіри до React.js.

Незважаючи на численні переваги, є лише кілька випадків, коли це середовище розробки буде найбільш підходящим:

- швидка розробка легких додатків корпоративного рівня;
- створення SPA або кросплатформенних додатків;
- розширення функціоналу наявної програми.

Це не означає, що React.js поганий в інших випадках. Швидше за все, альтернативні варіанти будуть кращими [4].

1.4.2. Next.js

Next.js – це фреймворк на основі React.js з можливістю візуалізації на стороні сервера. За допомогою нього можна швидко створювати динамічні програми з високою швидкодією та надійністю.

Основними перевагами Next.js є:

- «Гаряче» перезавантаження коду – сервер Next.js виявляє змінені файли та автоматично завантажує їх без перезапуску всього додатку;
- Автоматична маршрутизація – немає необхідності налаштовувати URL-адреси для маршрутизації. Файли розміщуються в папці «pages», які і формують усі доступні URL-адреси;
- Спеціальні стилі для компонентів – стиль jsx забезпечує підтримку як глобальних, так і ізольованих компонентних стилів;

- Візуалізація на стороні сервера – React-компоненти попередньо візуалізуються на сервері, отже, швидше завантажуються для клієнта;
- Екосистема Node.js – Next.js будучи React-базовим фреймворком, добре співпрацює з екосистемою Node.js. Це дає змогу писати як клієнтський, так і серверний код в одному місці;
- Автоматичний поділ коду – Next.js відображає сторінки з потрібними бібліотеками. Next.js замість створення одного великого файлу JavaScript створює кілька ресурсів. Коли сторінка завантажується, з нею завантажується лише необхідна сторінка JavaScript.;
- Динамічні компоненти – Next.js дозволяє динамічно імпортувати модулі JavaScript та компоненти React;
- Вбудована підтримка TypeScript – Next.js написаний на TypeScript і тому забезпечує чудову його підтримку [5].

1.4.3. Vue.js

Vue.js став відкриттям останніх років. Раптово, із рядового середовища розробки він став одним із найулюбленіших серед професіоналів.

Показово, що популярність виросла без підтримки великих компаній. Гіганти ринку звернули увагу на Vue.js тільки після різкого стрибка. Саме тому він виявився третім фреймворком в тривалому протистоянні React.js і Angular.

Переваги:

- невеликий розмір фреймворку;
- двостороння прив'язка даних;
- легкий у вивченні;
- докладна і грунтовна документація.

Недоліки:

- нестача підтримки масштабних проектів;
- нестача ресурсів;
- проблема роботи додатків для iOS та Safari;
- невелика спільнота.

Vue.js зараз виділяється тим, що він має величезну кількість особливостей. Деякі навіть можуть виступати в якості недоліків. Наприклад, Vue.js дуже гнучкий. Але в разі роботи над великими проектами в великій команді це може посприяти більшій кількості помилок.

Після того, як Vue.js почав проявляти свої особливості, на нього звернули увагу багато гігантів ринку, серед яких Gitlab, WizzAir, EuroNews. Grammarly повністю створений на базі Vue.js, а Alibaba і Xiaomi в 2018 році оголосили про повний перехід на Vue.js. Крім того, сьогодні фреймворк користується великим попитом на азіатському ринку. Тому переважна кількість обговорень і хитрощів використання описані на китайській мові.

В яких випадках Vue.js буде кращим вибором?

- для розробки «розумних» і високопродуктивних додатків;
- для раннього виходу програми на ринок;
- створення невеликих і легких додатків [4].

1.4.4. Angular

Angular довгий час був кращим вибором для розробки користувальницького інтерфейсу. Так, він став частиною популярного стека MEAN. Проте, за останні кілька років розробники все більше скаржилися на неповноцінність фреймворку в порівнянні з конкурентами. Як результат, багато компаній поступово відмовляються від використання Angular.

Переваги:

- двостороння прив'язка даних;
- директиви, що дозволяють створювати динамічний та насичений контент за допомогою HTML;
- впровадження залежності;
- за допомогою додаткового пакету Angular Universal можна відобразити додаток Angular на стороні сервера;
- велика спільнота.

Недоліки:

- погано підходить для створення трендових SPA;
- обмежена функціональність для SEO;
- погана документація у порівнянні з конкурентами.

Багато в чому, фреймворк втратив свою популярність через появу нових трендів веброзробки. Команда Angular не реалізувала необхідний функціонал в нових версіях середовища розробки. Тому сьогодні ми спостерігаємо, що Vue.js і React.js стають більш привабливими. Якщо виходити за рамки цих трьох фреймворків, то Angular іноді поступається і іншим інструментам (наприклад, Svelte). Тим не менш, Angular все ще використовується для підтримки багатьох популярних сайтів і вебдодатків. Серед них виявилися The Guardian, UpWork, PayPal і Sony. Всі вони є великими сайтами, в яких Angular проявляє себе досить добре.

У яких випадках варто звернути увагу на Angular.js?

- створення масштабних програм;
- необхідність наявності легко-масштабованої архітектури;
- створення месенджерів та інших додатків «в реальному часі» [4].

1.5. Вибір бібліотеки для інтернаціоналізації додатку

Інтернаціоналізація – це процес адаптації вебсайту під різні мови та регіони. До цього процесу відноситься переклад тексту, переведення форматів дат, грошових значень і так далі.

1.5.1 React-Intl

Для додатків, реалізованих на React, є чудова бібліотека React-Intl, яка надає готові компоненти та API для форматування дат, чисел та рядків, включаючи плюралізацію та обробку перекладів.

Всі переклади зберігаються в окремих файлах: по одному файлу JSON на локаль. Це дає змогу легко додати нову мову за необхідності.

Компонент IntlProvider від React-Intl забезпечує контекст інтернаціоналізації програми. Ним огортається головний компонент додатку. Це дозволяє отримати доступ до інформації про інтернаціоналізацію з будь-якого дочірнього компонента React. Доступ отримується за допомогою компонента FormattedMessage, в якому прописується id, визначене у файлі JSON.

1.5.2 Next-translate

Основною метою цієї бібліотеки є максимально простий переклад у середовищі Next.js. Next-translate має дві частини: плагін Next.js + i18n API.

В спеціальному конфігураційному файлі i18n.json визначається перелік доступних мов, мова за замовчуванням та список сторінок з необхідною множиною імен файлів для кожної.

Далі створюється ця множина файлів у форматі json, в яких будуть записані переклади. Всі файли розділені по папках, де назва папки відповідає певній мові.

У самому компоненті використовується функція useTranslate(‘назва_файлу’), яка повертає змінну t, в яку безпосередньо передається назва змінної в якій записаний текст необхідний для відображення.

1.5.3 Vue l18n

Vue I18n – це найпоширеніший плагін Vue для інтернаціоналізації додатку. Перед початком роботи потрібно встановити цей пакет і відповісти на запитання стосовно налаштувань: мова локалізації проекту; резервна мова; каталог, де зберігаються повідомлення тощо. Після цього створяться всі необхідні файли та залежності. Переклади зберігатимуться у файлах формату JSON – мова буде відповідати назві файлу. У компонентах Vue потрібно буде використовувати функцію $t і в неї передати ключ перекладу.

1.5.4 Ngx-translate

Ngx-translate – це інтернаціоналізована бібліотека для Angular.

Вона дозволяє визначати переклади вмісту різними мовами та легко переключатися між ними.

Ця бібліотека імпортується в головний модуль з певними налаштуваннями. Доступні мови визначаються у файлі environment.ts. Щоб сервіс перекладу запрацював, його потрібно викликати в AppComponent. Далі додаються файли формату JSON, де назва файлу відповідає певній мові.

Для використання перекладу у компоненті, бібліотека пропонує функцію під назвою translate, тобто це буде прописуватись як {{‘назва_змінної’|translate}}

1.6. Вибір препроцесору

SASS (Syntactically Awesome Style Sheets) – це скриптова мова та препроцесор CSS, який компілюється в CSS для того, щоб зробити швидші, простіші та елегантніші таблиці стилів. SASS додає нові функції та інструменти поверх основного CSS, щоб допомогти організовувати таблиці стилів для великих баз коду у зручний для обслуговування та корисний спосіб.

1.6.1. Порівняльна характеристика синтаксисів Sass і SCSS

Для SASS доступно два синтаксиси. Перший, відомий як SCSS (Sassy CSS), – це сучасний стандарт CSS, який використовує дужки та крапки з комою. SCSS був представлений у версії 3 SASS як надмножина CSS. SCSS не розширює стандарт CSS, а просто вдосконалює його синтаксис. Насправді CSS є правильним у синтаксисі SCSS, тому легко трансформуватись між ними. Файли, які використовують цей синтаксис, мають .scss розширення.

SCSS

```scss
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

CSS

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Рис. 1.6. Синтаксис SCSS

Другий і більш старий синтаксис, також відомий як краєний синтаксис або іноді просто Sass, дає більш стислу можливість роботи з CSS. Він використовує відступ для організації та розділення блоків коду і нові рядки замість точок з комою для розділення властивостей. По суті, це забезпечує стислий спосіб написання CSS, що розширює його функціональність. Файли, які використовують цей синтаксис, мають розширення .sass.

Sass

```sass
nav
  ul
    margin: 0
    padding: 0
    list-style: none

  li
    display: inline-block

  a
    display: block
    padding: 6px 12px
    text-decoration: none
```

CSS

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Рис. 1.7. Синтаксис Sass

Розглянемо деякі особливості препроцесору більш детально.

Змінні – забезпечують спосіб зберігання інформації, яка може використовуватись повторно. Наприклад, можна зберігати значення кольорів або шрифтів, які потім можна використовувати в будь-який момент у коді CSS. Це

означає, що якщо захочеться змінити якесь значення, то його потрібно буде оновити лише один раз.

Змінні заощаджують багато часу на переписування коду, а також запобігають появі помилок. Змінні особливо корисні для великих проектів.

Визначити змінну досить просто в SASS. Потрібен лише символ $.

SCSS

```scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Sass

```sass
$font-stack:    Helvetica, sans-serif
$primary-color: #333

body
  font: 100% $font-stack
  color: $primary-color
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Рис. 1.8. Приклад використання змінних

Імпорт – дозволяє розділити CSS-файл на дрібніші файли. Це корисно, якщо файл SASS стає занадто великим. Назва часткового файлу починається зі знаку нижнє підкреслення («_»). Потім цей файл можна імпортувати в будь-який інший за допомогою директиви @import. В результаті виходить один CSS-файл, зкомпільований із декілька фрагментів.

SCSS

```scss
// _reset.scss
html,
body,
ul,
ol {
  margin:  0;
  padding: 0;
}
```

```scss
// base.scss
@import 'reset';
body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

CSS

```css
html,
body,
ul,
ol {
  margin:  0;
  padding: 0;
}
body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

Sass

```sass
// _reset.sass
html,
body,
ul,
ol
  margin:  0
  padding: 0
```

```sass
// base.sass
@import reset
body
  font: 100% Helvetica, sans-serif
  background-color: #efefef
```

CSS

```css
html,
body,
ul,
ol {
  margin:  0;
  padding: 0;
}
body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

Рис. 1.9. Приклад використання імпорту файлів

Міксіни – дозволяють згрупувати кілька рядків коду, які можна перевикористати. Вони схожі на функції інших мов програмування. Для більшої гнучкості, у них також можна передавати змінні.

SCSS

```scss
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}
.box { @include transform(rotate(30deg)); }
```

```css
.box {
  -webkit-transform: rotate(30deg);
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}
```

Sass

```sass
=transform($property)
  -webkit-transform: $property
  -ms-transform: $property
  transform: $property
.box
  +transform(rotate(30deg))
```

```css
.box {
  -webkit-transform: rotate(30deg);
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}
```

Рис. 1.10. Приклад використання міксінів

1.7. Вибір середовища розробки

При розробці великих проектів інтегровані середовища розробки (англ. Integrated Development Environment – IDE) значно полегшують роботу програмістові і скорочують час на розробку. Тому дуже важливо правильно підібрати IDE під свій проект та задачі.

Розглянемо два середовища, які найкраще підходять для розробки на JavaScript.

1.7.1. Середовище розробки WebStorm

WebStorm – інтегроване середовище розробки від компанії JetBrains, розроблене на основі платформи IntelliJ IDEA, як висококласне IDE для проектів веброзробки.

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію по коду, рефакторинг, зневадження та інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладений Javascript) — в таких конструкціях підтримується коректний рефакторинг [6].

1.7.2. Середовище розробки Visual Studio Code

Visual Studio Code – редактор коду для кросплатформенної розробки веб і хмарних додатків. Він підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію по коду, підтримку Git та інші можливості.

Багато можливостей Visual Studio Code недоступні через графічний інтерфейс, найчастіше вони використовуються через палітру команд, яка

викликається поєднанням клавіш, або JSON файли (наприклад, налаштування користувача).

Visual Studio Code має підтримку плагінів, доступних через Visual Studio Marketplace. Вони можуть включати в себе доповнення до редактора, підтримку додаткових мов програмування, статичні аналізатори коду [7].

Висновок до розділу 1

У першому розділі було розглянуто найпопулярніші засоби розробки для вирішення поставленої задачі.

Даний проект буде реалізовуватись за допомогою мови програмування JavaScript. Розглянувши два підходи візуалізації, оберемо візуалізацію на стороні сервера, оскільки вона має значні переваги при завантаженні сторінки та оптимізації для пошукових систем. Відповідно до цієї візуалізації оберемо фреймворк Next.js, оскільки він підтримує SSR, а також базується на бібліотеці React, яка легко вивчається, гнучка у використанні та має розширену функціональність. Також буде використовуватись бібліотека для інтернаціоналізації додатку Next-translate, яка сумісна з даним фреймворком.

Розглянувши препроцесор SASS та всі його можливості, оберемо синтаксис SCSS, тому що він більш зрозумілий.

Для розробки було розглянуто два середовища, які чудово підходять для написання коду і не тільки, адже мають багато різних функціональних можливостей. Для розробки даного проекту буде використовуватись середовище Visual Studio Code.

РОЗДІЛ 2. СТВОРЕННЯ ПРОЕКТУ І ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1. Визначення функціональних вимог

Інформаційна система басейнобудівної компанії може мати багатьох акторів, якими є звичайні користувачі (клієнти) або адміністратори інформаційної системи, які керують даними.

Розглянемо можливості системи, які надаються користувачам та адміністраторам. На рисунках 2.1. і 2.2. представлені діаграми прецедентів, які описує дії і функції адміністратора та користувача відповідно.

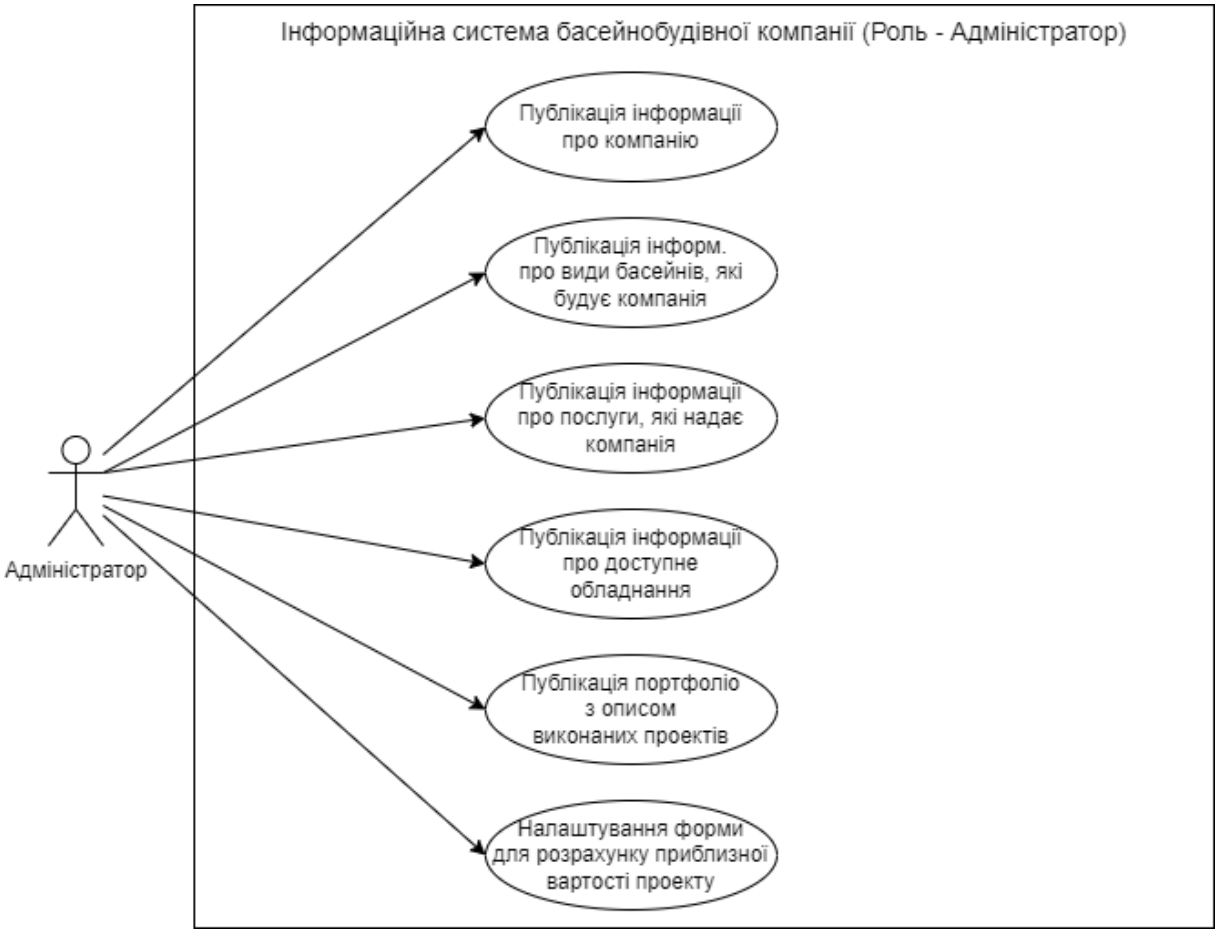


Рис. 2.1. Діаграма прецедентів інформаційної системи (Роль – Адміністратор)

Адміністратор відповідає за дані інформаційної системи, які буде переглядати користувач. Адміністратору надаються такі можливості:

- публікація інформації про компанію;
- публікація інформації про види басейнів, які будує компанія;
- публікація інформації про послуги, які надаються;
- публікація інформації про доступне обладнання;
- публікація портфоліо з описом виконаних проектів;
- налаштування форми для розрахунку приблизної вартості.

Рис. 2.2. Діаграма прецедентів інформаційної системи (Роль – Користувач)

Користувачеві надаються такі можливості:

- перегляд інформації про компанію;
- перегляд інформації про види басейнів, які будує компанія разом з можливістю розрахунку приблизної вартості проекту;
- перегляд інформації про послуги, які надаються;
- перегляд інформації про доступне обладнання;
- перегляд портфоліо виконаних проектів, включаючи опис кожного проекту;
- можливість переглянути інформацію вебсайту різними мовами: українською та російською;
- можливість зробити замовлення безпосередньо через вебсайт за допомогою номеру мобільного телефону чи електронної пошти.

2.2. Створення прототипу

Перш ніж розробити прототип, потрібно дізнатись більше інформації про компанію та її цільову аудиторію.

Компанія Swimming Pools складається з професійних інженерів, які займаються виробництвом та установкою різних типів басейнів, а також постачають спеціальне обладнання для об’єктів по всій Україні. Незалежно від того, чи це інноваційний багатоцільовий водний об’єкт, висотний спеціалізований басейн, екзотичний курортний басейн чи унікальний водний об’єкт, компанія Swimming Pools пропонує найсучасніші методи будівництва та проектування, надає технічну експертизу, якісне обладнання та основні компоненти для басейнів.

Зі всіма замовниками веде зв’язок генеральний директор, який збирає всі деталі та вимоги клієнта, формує кошторис витрат на реалізацію проекту, укладає договір. Після цього інженери починають свою роботу. Замовниками можуть бути як фізичні особи, так і корпорації, які прагнуть побудувати басейн на своїй території. Для таких клієнтів сайт повинен бути у світлих тонах з

певними яскравими акцентами. За світлі кольори візьмемо базові: білий та світло-сірий, за яскраві – жовтий, як символ сонця, та відтінки синього, як символу води. Такі кольори чудово підходять під концепцію компанії.

Розроблювана інформаційна система буде багатосторінковою, адже велика кількість інформації повинна бути представлена. Таким чином, потрібно зробити зручну та інтуїтивно зрозумілу навігацію. Також додати анімацію до різних кнопок, статичних елементів тощо, щоб користувачеві було цікаво взаємодіяти з сайтом.

Виділимо основні сторінки інформаційної системи та їх призначення:

- головна – сторінка, яка повинна дати стислу інформацію про компанію;
- басейни – сторінка, яка повинна містити детальну інформацію про всі види басейнів, які клієнт може замовити;
- послуги – сторінка, яка повинна містити детальну інформації про послуги, які надає компанія;
- обладнання – сторінка, яка повинна містити детальну інформацію про обладнання для басейнів;
- портфоліо – сторінка, на якій розміщуватимуться фотографії виконаних проектів з коротким описом, що було зроблено.

Кожна сторінка матимете «шапку», на якій буде можливість переключення мови сайту, повернення на головну сторінку, а також вказаний номер телефону, по натиску на який, можна буде зателефонувати представникам компанії; навігацію, за допомогою якої можна буде перейти на будь-яку сторінку; «підвал» на якому буде розміщено копірайт, номер телефону та електронна пошта.

Перейдемо до розробки прототипів сторінок інформаційної системи.

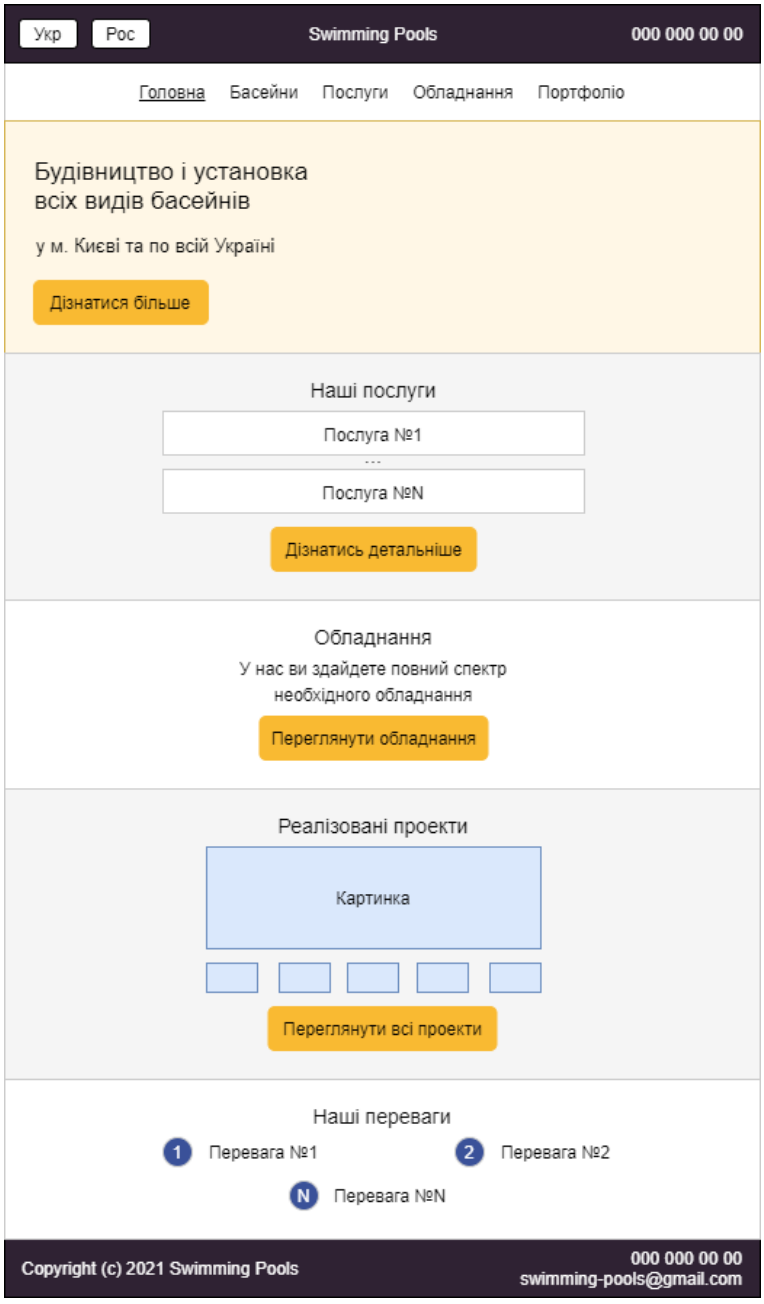


Рис. 2.3. Прототип головної сторінки

Рис. 2.4. Прототип сторінки басейнів

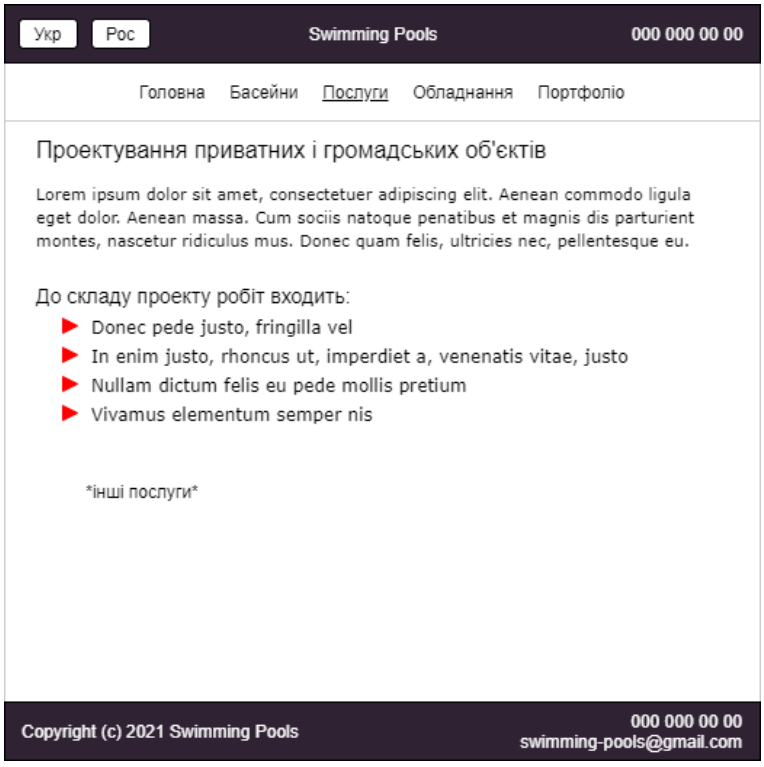


Рис. 2.5. Прототип сторінки послуг

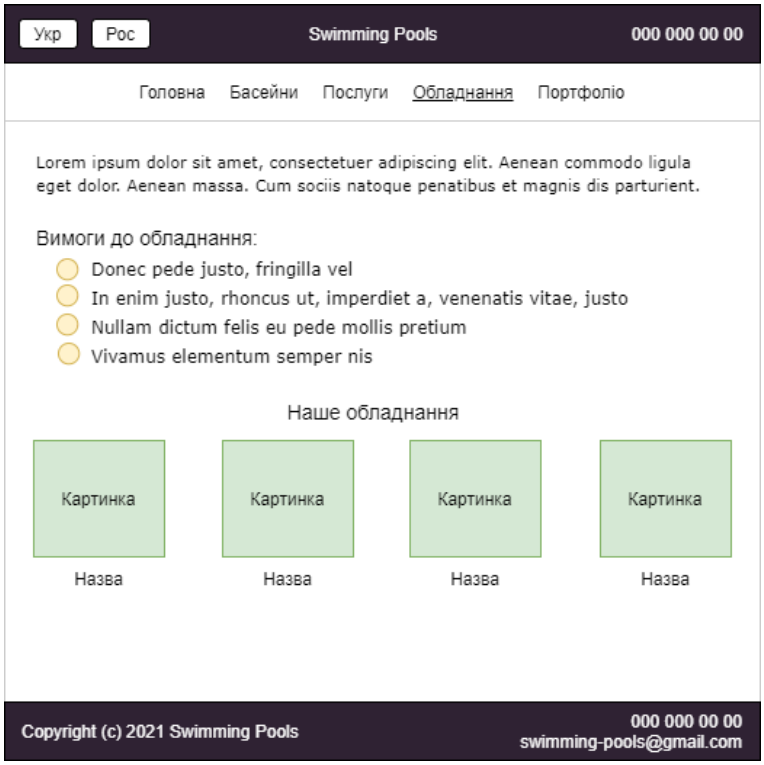


Рис. 2.6. Прототип сторінки обладнання

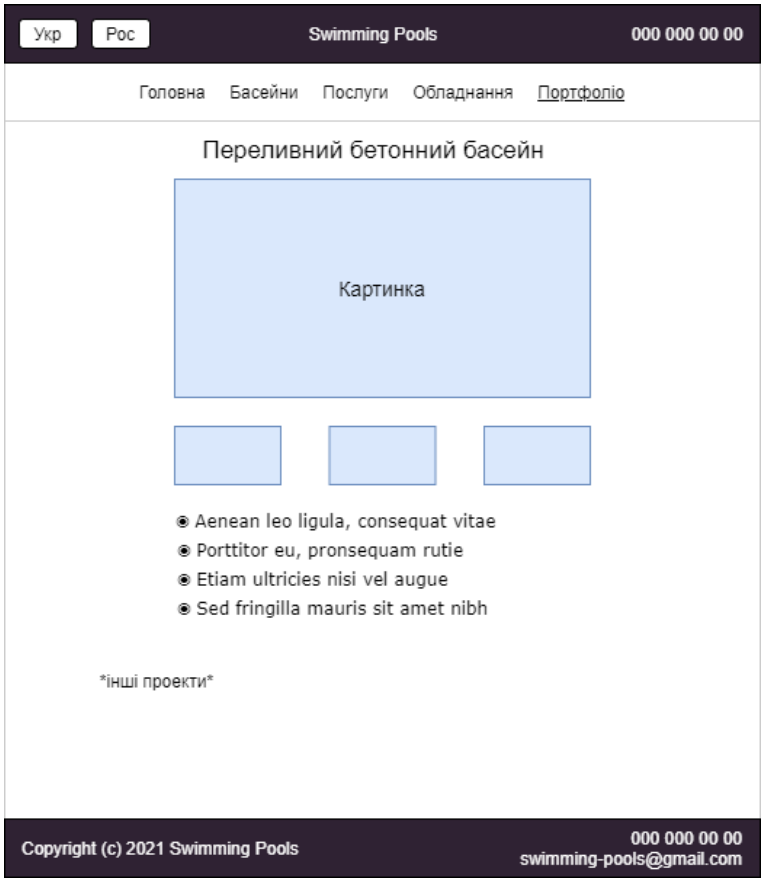


Рис. 2.7. Прототип сторінки портфоліо

2.3. Ініціалізація проекту та створення базової структури

Ініціалізація проекту відбувається за допомогою команди *npx create-next-app,*яка прописується в командному рядку. За допомогою цієї команди все автоматично налаштовується: створюються необхідні папки з файлами, встановлюються залежності, прописуються базові команди для розробки.

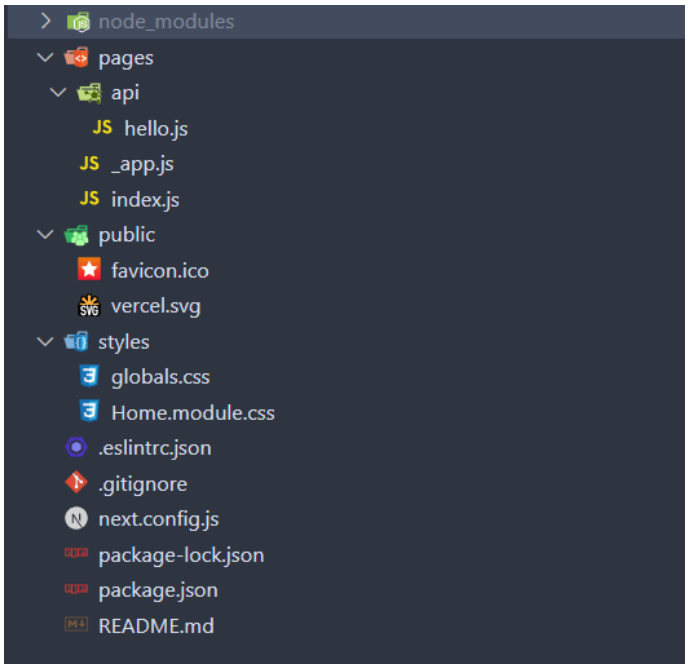


Рис. 2.8. Ініціалізація проекту

У папці *pages* знаходяться файли, які відповідають певним сторінкам. *_app.js* є ядром нашого проекту, *index.js* – це початкова точка або інакше кажучи головна сторінка.

У папці *pages* > *api* зберігаються прикладні інтерфейси, які необхідні для бекенд розробки.

У папці *public* зберігаються статичні елементи проекту, а у *styles* – глобальні стилі або файли стилів, які необхідні для більшості сторінок.

У *next.config.js* прописуються глобальні конфігурації для проекту, а package.json зберігає важливі метадані та визначає функціональні атрибути проекту.

Далі необхідно встановити через командний рядок додаткові бібліотеки, які використовуватимуться для розробки проекту:

- *npm install sass* – для написання стилів за допомогою препроцесору Sass;
- *npm install classnames* –для умовного об’єднання назв класів разом;
- *npm install embla-carousel-react* – для створення адаптивної, зручної, сучасної каруселі фотографій;
- *npm install next-translate* – для інтернаціоналізації додатку.

Створимо у папці *pages* файли для всіх сторінок розроблюваної інформаційної системи:

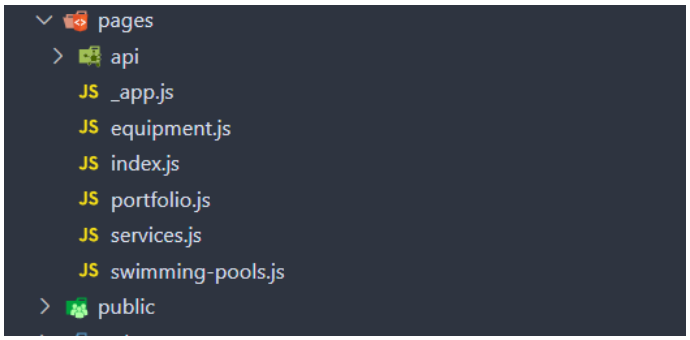


Рис. 2.9. Створення сторінок

У папку *styles* додамо глобальні стилі, а також допоміжні стилі, які необхідні для більшості компонентів проекту.

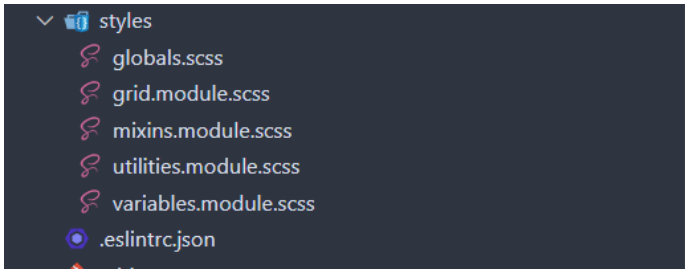


Рис. 2.10. Створення стилів

Рис. 2.11. Приклад глобальних стилів

У папку *public* додамо картинки.

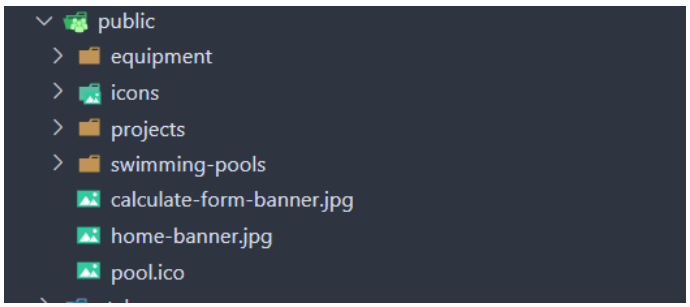


Рис. 2.12. Додавання картинок

На глобальному рівні створимо папку *components*, в якій зберігатимуться компоненти, з яких будуть будуватись сторінки. Глобальна папка буде мати

множину підкаталогів, в яких у свою чергу буде зберігатись головний файл компоненту та стилі за необхідністю.

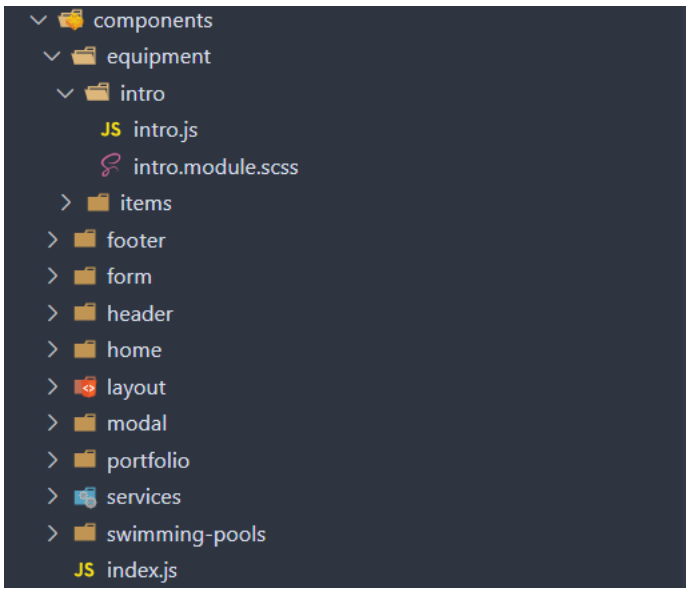


Рис. 2.13. Створення компонентів

Для функціональності інтернаціоналізації додатку, створимо на глобальному рівні папку *locales* з двома підкаталогами: *uk* і *ru* – це коди мов української та російської відповідно до всесвітнього стандарту. У кожному каталозі створимо однойменні *json* файли окремо для кожної сторінки, в них буде прописуватись весь контент.

Рис. 2.14. Створення файлів локалізації

Далі на глобальному рівні створимо файл *i18n.json* в якому пропишемо наступну конфігурацію:

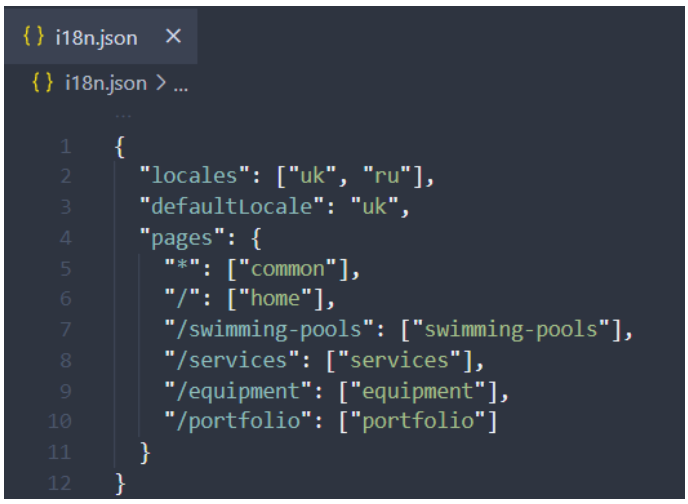


Рис. 2.15. Створення конфігурації для локалізації

В ньому ми вказали доступні локалі для нашого проекту, українську мову за замовчуванням, а також для кожної сторінки проекту прописали назву файлу, в якому зберігатиметься її контент. Для всіх сторінок за допомогою знаку * визначили файл *common.json* – це означає, що кожна сторінка матиме до нього доступ.

Після цього перейдемо в *next.config.js* і викличемо функцію *nextTranslate*, щоб наша локалізація запрацювала.

Рис. 2.16. Глобальне налаштування для локалізації

2.4. Розробка компонентів

Весь проект складається з множини компонентів, з яких і будується розроблювана інформаційна система. В цьому підрозділі розглянемо приклад розробки лише деяких з них.

2.4.1. Розробка контейнеру

Контейнер (*англ. layout*) – це такий компонент, який буде зберігати у собі основні компоненті («шапка», навігація, «підвал») та в який динамічно можливо буде вставляти й інші компоненти. Кожна сторінка буде мати цей контейнер і в нього передавати інші компоненти, які відрізнятимуть її від інших сторінок.

В першу чергу розробимо основні елементи контейнеру: «шапку» з навігацією та «підвал».

«Шапка» з навігацією буде знаходитись у папці *header*. У файлі *header.js* заімпортуємо всі необхідні залежності:

- *useState* і *useEffect* – для того, щоб керувати станом компонента, та реагувати на зміни при життєвому циклі компонента;
- *useTranslation* – для доступу до контенту з файлів інтернаціоналізації;
- *Link* – для створення посилань;
- *useRouter* – для керування маршрутизацією;
- *classnames* – для написання умовних класів;
- *styles* та *grid* – власні стилі компонента та стилі глобальної розмітки.

Рис. 2.17. Імпортування залежностей

За допомогою функції *export* експортуємо даний компонент. Це потрібно для того, щоб інші компоненти мали до нього доступ. Всередині використовується функція *return* в яку передається конкретна розмітка компонента – у даному випадку HTML-тег *<header></header>*, який розділений на дві частини: саму «шапку» і навігацію. Також викличемо функції маршрутизації та локалізації.

```
export const Header = () => {
  const router = useRouter();
  const { t } = useTranslation('common');

  return (
    <header>
      <div className={styles['header__top-wrapper']}>···
      </div>

      <div className={cn(styles['header__bottom-wrapper'], {···
      </div>
    </header>
  );
};
```

Рис. 2.18. Експорт компонента

У верхній частині будуть знаходитись кнопки для перемикання мови сайту, логотип та номер телефону.

Для кнопок будемо використовувати HTML-тег списку *
* з дочірніми елементами *
*. Попередньо викликана функція *router* дає нам доступ до доступних локалей через *router.locales*. Цей метод поверне у свою чергу масив, тому ми можемо викликати на ньому функцію масиву *map*, щоб отримати кожну локаль окремо. Кнопку огорнемо в тег *Link*, в який потрібно передати саму локаль та шлях. *router.locale* містить в собі активну локаль, тому за допомогою порівняння на цю змінну встановимо активний клас для кнопки.

```
<ul className={styles['header__languages']}>
  {router.locales.map(locale => (
    <li key={locale} className={styles['header__languages-item']}>
      <Link href={router.asPath} locale={locale}>
        <a className={cn(styles['header__languages-link'], {
          [styles['header__languages-link--active']]: router.locale === locale
        })}>
          {locale === 'ru' ? 'Рус' : 'Укр'}
        </a>
      </Link>
    </li>
  ))}
</ul>
```

Рис. 2.19. Створення кнопок для перемикання мови

В логотипі вкажемо назву компанії та по кліку на нього зробимо перенаправлення на головну сторінку. Для цього будемо використовувати тег *Link*.

```
<Link href='/'>
  <a className={styles['header__logo']}>Swimming Pools</a>
</Link>
```

Рис. 2.20. Створення логотипу

Далі додамо номер телефону, вкажемо його у тезі **. Також за допомогою тега *<img />* додамо картинку телефона, яка знаходиться у папці *public*. Все це огорнемо в *Link* де в атрибуті *href* вкажемо *tel*:номер. Це потрібно для того, щоб по кліку на номер автоматично запустився додаток, який відповідає за телефонні дзвінки.

```
<Link href='tel:0931112233'>
  <a className={styles['header__phone']}>
    <img src='/icons/phone.svg' alt='phone' width='30' height='30' />
    <span>093 111 22 33</span>
  </a>
</Link>
```

Рис. 2.21. Створення номеру телефона

Верхня частина компонента готова. Перейдемо до нижньої, яка складатиметься лише з навігації. Будемо використовувати тег *<nav></nav>* всередині якого буде знаходитись вже відомий нам список. Кожен елемент навігації складається з назви сторінки та безпосередньо посилання. Всі назви

визначимо у файлі *common.json*. Доступ до них у компоненті буде через функцію *t*, яка була оголошена напочатку. В неї потрібно передати лише назву змінної.

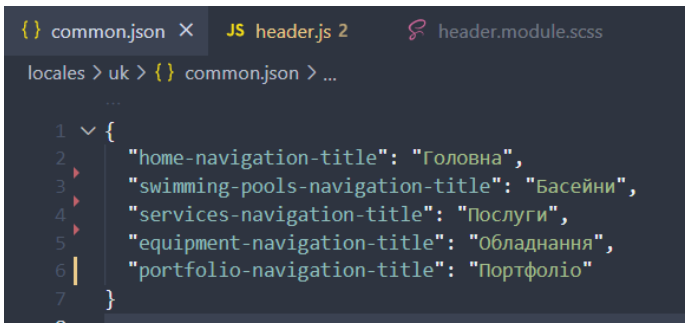


Рис. 2.22. Визначення назв для навігації

Для посилання буде використовуватись тег *Link*, де в *href* буде вказаний шлях на сторінку (насправді це назва файлів, визначених в *pages*). Активний клас для елемента встановимо, якщо поточний шлях дорівнює ця сторінка. Для визначення поточного шляху використовується функція *router.pathname*.

```
<div className={`${grid.container} ${styles['header__bottom']}`}>
  <nav className={styles['navigation__list-wrapper']}>
    <ul className={styles['navigation__list']}>
      <li className={styles['navigation__item']}>
        <Link href='/'>
          <a className={cn(styles['navigation__link'], {
            [styles['navigation__link--active']]: router.pathname === '/'
          })}>
            {t('home-navigation-title')}
          </a>
        </Link>
      </li>
      <li className={styles['navigation__item']}>
        <Link href='/swimming-pools'>
          <a className={cn(styles['navigation__link'], {
            [styles['navigation__link--active']]: router.pathname === '/swimming-pools'
          })}>
            {t('swimming-pools-navigation-title')}
          </a>
        </Link>
      </li>
      <li className={styles['navigation__item']}>
        <Link href='/services'>
          <a className={cn(styles['navigation__link'], {
            [styles['navigation__link--active']]: router.pathname === '/services'
          })}>
            {t('services-navigation-title')}
          </a>
        </Link>
      </li>
      <li className={styles['navigation__item']}>
        <Link href='/equipment'>
          <a className={cn(styles['navigation__link'], {
            [styles['navigation__link--active']]: router.pathname === '/equipment'
          })}>
            {t('equipment-navigation-title')}
```

Рис. 2.23. Створення навігації

Також для зручності користування сайтом було б добре, щоб навігація завжди була перед очима у користувача, тобто неважливо наскільки вниз він проскролить, навігація завжди повинна залишатися зверху. В цьому нам допоможе *css*. Створимо клас *header--sticky* у *header.module.scss* з наступними властивостями:

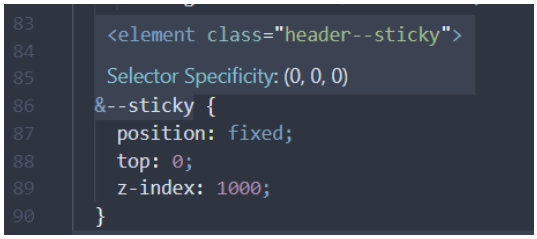


Рис. 2.24. Визначення стилів для фіксованої навігації

position: fixed відповідає саме за те, що елемент повинен бути зафіксованим.

Цей клас до елементу маємо додавати динамічно. Умовою для класу буде якщо сторінку проскролено на 60пх. Це число є фіксованою висотою для «шапки». З самого початку за допомогою функції *useState* встановимо висоту рівну нулю. Після ініціалізації сторінки викличемо функцію *useEffect*, яка підпишеться на подію браузера “scroll”. При спрацюванні цієї події будемо оновлювати висоту. *Classnames (cn)* буде спостерігати за висотою і відповідно встановлювати або прибирати клас *header--sticky*.

Рис. 2.25. Створення фіксованої навігації

Перейдемо до створення «підвалу». Він буде знаходитись у папці footer та складатись з тексту на авторське право, номеру телефона та електронної пошти. Для тексту використаємо тег * *, а також функцію *new Date().getFullYear()*, яка поверне поточний рік. Для номеру телефона та пошти будемо використовувати тег *<a>*, в атрибут *href* якого передамо *tel*:номер і *mailto*:пошта відповідно, щоб після кліку на них автоматично відкрились додатки для дзвінків та пошти.

Рис. 2.26. Створення компонента «підвал»

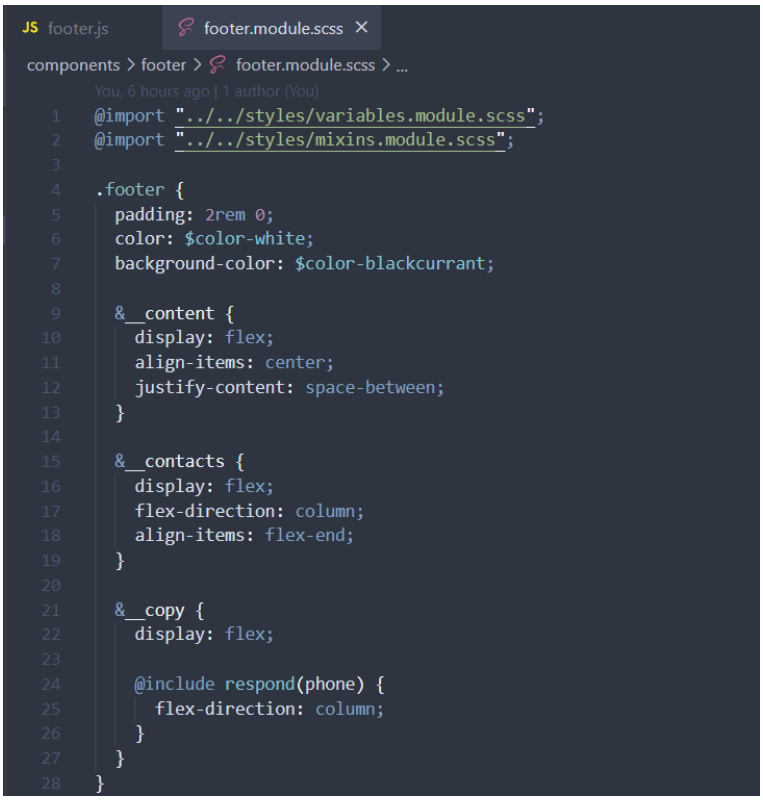


Рис. 2.27. Приклад стилів для компонента «підвал»

Після створення основних компонентів можемо перейти до створення контейнеру безпосередньо, який знаходитиметься у папці *layout*.

Рис. 2.28. Створення компоненту «контейнер»

В цей компонент заімпортували щойно створені компоненти *Header* та *Footer*, а також компонент *Head*, який є аналогом HTML-тегу *<head></head>*. В *Head* визначаємо *title* сторінки – це саме та назва, яка буде відображатись на табі браузера; мета-теги *description* та *keywords*, які необхідні для SEO-оптимізації. Ці атрибути ми будемо приймати динамічно з інших сторінок, якщо вони не передаються, то буде встановлено значення за замовчуванням, яке йде після знаку «=». *Children* – це всі ті динамічні компоненти, які поміщатимуться в основний тег *<main></main>*.

Розглянемо приклад використання контейнеру на домашній сторінці.

Рис. 2.29. Приклад використання контейнеру

У домашню сторінку імпортується *Layout* разом з рештою компонентів. Він має бути найпершим у структурі. У контейнер передаються такі атрибути як *title*, *description* та *keywords*, а також дочірні елементи, якими динамічно заповниться основний контент сторінки.

2.4.2. Розробка сторінки басейнів

На сторінці басейнів буде розміщена інформація про типи басейнів, будівництвом яких займається компанія, – бетонний, композитний та басейн з нержавіючої сталі. Для кожного типу створимо окрему папку та файл в *components* > *swimming-pools* та загальний для всіх типів файл стилів.

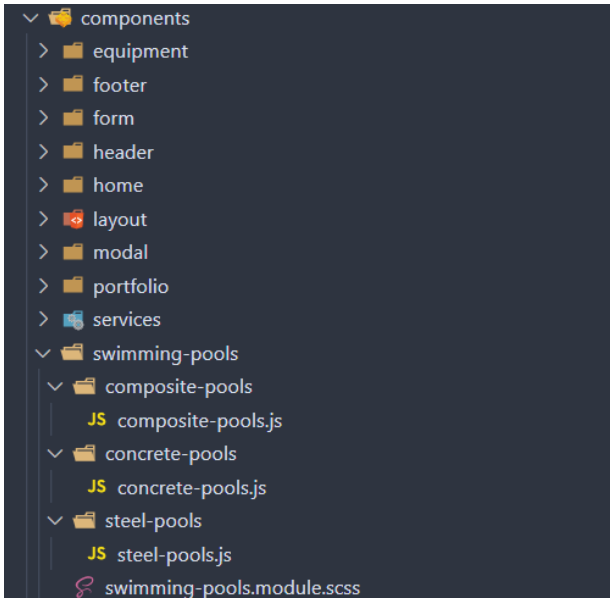


Рис. 2.30. Створення базової структури для компонентів басейнів

Перейдемо у файл *concrete-pools.js* та почнемо створювати компонент бетонного басейну. Цей компонент буде складатись з загальної інформації про бетонний басейн, типів можливого водообміну та видів оздоблень.

Перейдемо у файли локалізації *swimming-pools.json* та створимо множину змінних з необхідним текстом, який буде використовуватись у цьому компоненті.

```
"concrete-pools-title": "Бетонні басейни",
"concrete-pools-description": "Бетонний басейн – складна гідротехнічна споруда, відноситься до класичного типу залізобетонних конструкцій
"concrete-pools-water-exchange-title": "Тип водообміну",
"concrete-pools-water-exchange-type-1": "Скімерні басейни",
"concrete-pools-water-exchange-type-2": "Переливні басейни",
"concrete-pools-water-exchange-type-3": "Нескінченний перелив",
"concrete-pools-water-exchange-type-4": "Грецький перелив",
"concrete-pools-finishies-title": "Види оздоблень",
"concrete-pools-finishies-type-1": "Мозаїка",
"concrete-pools-finishies-type-2": "Плитка",
"concrete-pools-finishies-type-3": "Плівка ПВХ",
```

Рис. 2.31. Приклад створення українського тексту
для компонента бетонного басейну

```
"concrete-pools-title": "Бетонные бассейны",
"concrete-pools-description": "Бетонный бассейн – сложное гидротехническое сооружение, относится к классическому виду железобетонных конст
"concrete-pools-water-exchange-title": "Тип водообмена",
"concrete-pools-water-exchange-type-1": "Скиммерные бассейны",
"concrete-pools-water-exchange-type-2": "Переливные бассейны",
"concrete-pools-water-exchange-type-3": "Бесконечный перелив",
"concrete-pools-water-exchange-type-4": "Греческий перелив",
"concrete-pools-finishies-title": "Виды отделок",
"concrete-pools-finishies-type-1": "Мозаика",
"concrete-pools-finishies-type-2": "Плитка",
"concrete-pools-finishies-type-3": "Пленка ПВХ",
```

Рис. 2.32. Приклад створення російського тексту
для компонента бетонного басейну

Заімпортуємо базові залежності: елемент картинки, перекладу та файли стилів, та за допомогою функції *export* експортуємо даний компонент, щоб мати до нього доступ з інших файлі.

Рис. 2.33. Початок створення компонента бетонного басейну

З самого початку додамо заголовок, а також картинку бетонного басейну і його загальну характеристику. Картинки, пов’язані зі сторінкою басейнів, будуть знаходитись у папці *public* > *swimming-pools*. Вбудований елемент *Image* дає змогу додавати картинки на сайт у ефективний спосіб, адже «під капотом» буде працювати механізм *lazy loading*. Його головною перевагою є те, що зображення не завантажуватимуться доти, доки не знадобляться відвідувачу. Таким чином, це значно посприяє економії інтернет-трафіку. В елемент *Image* можна передавати такі властивості як ширина (*width*), висота (*height*), якість зображення (*quality*), а також атрибут *priority*, який говорить про те, що картинка є пріоритетною і її потрібно завантажити в першу чергу.

```jsx
<h3 className={utilities['u-margin-small']}>
  {t('concrete-pools-title')}
</h3>

<div className={`${styles['swimming-pools__content']} ${styles['swimming-pools__content-40-60']}`}>
  <div className={styles['swimming-pools__content-left']}>
    <Image
      src='/swimming-pools/concrete-pool.jpg'
      alt='Concrete Pool'
      width={720}
      height={360}
      quality={100}
      priority
    />
  </div>
  <div className={styles['swimming-pools__content-right']}>
    <p>{t('concrete-pools-description')}</p>
  </div>
</div>
```

Рис. 2.34. Розробка верхньої частини компонента бетонного басейну

swimming-pools.module.scss

components > swimming-pools > swimming-pools.module.scss > .swimming-pools > &__color >

You, 6 days ago | 1 author (You)

```scss
@import "../../styles/mixins.module.scss";
@import "../../styles/variables.module.scss";

.swimming-pools {
  &__content {
    display: flex;
    align-items: center;
    justify-content: space-between;

    @include respond(tab-port) {
      flex-direction: column;
    }

    &-img {
      @include respond(tab-port) {
        order: -1;
      }
    }

    img {
      width: 100%;
      border-radius: 2px;
    }
  }

  &__content-40-60 {
    .swimming-pools__content-left {
      flex-basis: 40%;
    }

    .swimming-pools__content-right {
      flex-basis: 55%;
    }
  }

  &__content-60-40 {
    .swimming-pools__content-left {
      flex-basis: 55%;
    }

    .swimming-pools__content-right {
      flex-basis: 40%;
    }
  }
```

Рис. 2.35. Приклад стилів для верхньої частини компонента бетонного басейну

Далі перейдемо до частини про типи водообміну. Кожен тип буде складатись з назви та картинки. Розміщені вони будуть в ряд від одного до чотирьох елементів в залежності від ширини екрану.

```
<h4 className={utilities['u-margin-small']}>
  {t('concrete-pools-water-exchange-title')}
</h4>

<div className={grid.row}>
  <div className={`${grid['col-1-of-4']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('concrete-pools-water-exchange-type-1')}
    </p>
    <Image
      src='/swimming-pools/type-of-water-exchange-1.jpg'
      alt='Skimmer Pools'
      width={176}
      height={119}
      quality={100}
    />
  </div>
  <div className={`${grid['col-1-of-4']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('concrete-pools-water-exchange-type-2')}
    </p>
    <Image
      src='/swimming-pools/type-of-water-exchange-2.jpg'
      alt='Overflow pools'
      width={176}
      height={119}
      quality={100}
    />
  </div>
  <div className={`${grid['col-1-of-4']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('concrete-pools-water-exchange-type-3')}
    </p>
    <Image
      src='/swimming-pools/type-of-water-exchange-3.jpg'
      alt='Endless overflow'
      width={176}
```

Рис. 2.36. Розробка типів водообміну компонента бетонного басейну

```
46  &__item {
47    position: relative;
48    padding: 3rem 1rem 2rem;
49    text-align: center;
50    border: 1px solid $color-whisper;
51    border-radius: 2px;
52
53    &-title {
54      top: -1.5rem;
55      left: 2rem;
56      position: absolute;
57      padding: 0 4px;
58      font-weight: 700;
59      letter-spacing: normal;
60      background-color: $color-white;
61    }
62
63    img {
64      border-radius: 2px;
65    }
66  }
```

Рис. 2.37. Приклад стилів для елементів

Види оздоблень аналогічно будуть складатись з назви та картинки. Розміщені вони будуть в ряд від одного до трьох елементів в залежності від ширини екрану.

```
<h4 className={utilities['u-margin-bottom-small']}>
  {t('concrete-pools-finishies-title')}
</h4>

<div className={grid.row}>
  <div className={`${grid['col-1-of-3']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('concrete-pools-finishies-type-1')}
    </p>
    <Image
      src='/swimming-pools/type-of-finish-1.jpg'
      alt='Mosaic'
      width={250}
      height={119}
      quality={100}
    />
  </div>
  <div className={`${grid['col-1-of-3']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('concrete-pools-finishies-type-2')}
    </p>
    <Image
      src='/swimming-pools/type-of-finish-2.jpg'
      alt='Tile'
      width={250}
      height={119}
      quality={100}
    />
  </div>
  <div className={`${grid['col-1-of-3']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('concrete-pools-finishies-type-3')}
    </p>
    <Image
      src='/swimming-pools/type-of-finish-3.jpg'
```

Рис. 2.38. Розробка видів оздоблень компонента бетонного басейну

З розробкою компонента бетонного басейну завершено. Перейдемо у файл *composite-pools.js* та почнемо розроблювати компонент композитного басейну. Цей компонент буде складатись з загальної інформації про композитний басейн, можливих форм та габаритів виконання чаші, а також кольорових покриттів.

Спершу у файлі *swimming-pools.json* створимо весь необхідний текст.

```
"composite-pools-title": "Композитні басейни",
"composite-pools-description": "Композитний басейн – це готова чаша басейну з міцного матеріалу скловолокна, яка швидко монтується і прост
"composite-pools-bowl-forms": "Можливі форми виконання чаші",
"composite-pools-bowl-forms-type-1": "Прямокутна",
"composite-pools-bowl-forms-type-2": "Овальна",
"composite-pools-bowl-forms-type-3": "Кругла",
"composite-pools-bowl-forms-type-4": "Нестандартна",
"composite-pools-bowl-forms-type-5": "Купіль",
"composite-pools-sizes": "Габаритні розміри виконання",
"composite-pools-sizes-description": "У лінійці композитних басейнів існують різні габарити чаш басейнів",
"composite-pools-sizes-width": "Ширина від 1,4м до 4,25м",
"composite-pools-sizes-length": "Довжина від 2м до 12,9м",
"composite-pools-sizes-depth": "Глибина від 0,85м до 2,0м",
"composite-pools-colors": "Кольорові рішення в лінійці композитних басейнів",
"composite-pools-colors-in-water": "у воді",
"composite-pools-colors-type-1": "Покриття 'MARBLE 3D'",
"composite-pools-colors-type-1-1": "ПЕРВАНШ",
"composite-pools-colors-type-1-2": "ХУКО",
"composite-pools-colors-type-1-3": "РІЦА",
"composite-pools-colors-type-1-4": "ФАРАОН",
"composite-pools-colors-type-1-5": "ПАПІРУС",
"composite-pools-colors-type-2": "Покриття 'DIAMOND SHINE'",
"composite-pools-colors-type-2-1": "БРИЛІАНТ",
"composite-pools-colors-type-2-2": "СТРАТОСФЕРА",
"composite-pools-colors-type-2-3": "ІЗУМРУД",
"composite-pools-colors-type-2-4": "КРИСТАЛ",
"composite-pools-colors-type-2-5": "ПЛАТИНУМ",
"composite-pools-colors-type-3": "Класичне покриття",
"composite-pools-colors-type-3-1": "БЕРЛІНСЬКА ЛАЗУРЬ",
"composite-pools-colors-type-3-2": "БЛАНЖЕ",
"composite-pools-colors-type-4": "Покриття 'CHAMELEON'",
"composite-pools-colors-type-4-1": "ЖЕМЧУЖНИЙ",
"composite-pools-colors-type-4-2": "РОЗОВИЙ ЖЕМЧУГ",
"composite-pools-colors-type-4-3": "ТРИТОН",
```

Рис. 2.39. Приклад створення українського тексту
для компонента композитного басейну

Заімпортуємо базові залежності: елемент картинки, перекладу та файли стилів, та за допомогою функції *export* експортуємо даний компонент, щоб мати до нього доступ з інших файлі.

```
JS composite-pools.js 2 ×
components > swimming-pools > composite-pools > JS composite-pools.js > ...
      You, 6 days ago | 1 author (You)
1     import Image from 'next/image';
2     import useTranslation from 'next-translate/useTranslation';
3     import styles from '../swimming-pools.module.scss';
4     import grid from '../../../styles/grid.module.scss';
5     import utilities from '../../../styles/utilities.module.scss';
6
7     export const CompositePools = () => {
8       const { t } = useTranslation('swimming-pools');
9
10      return (
11    >   <div className={utilities['u-margin-bottom-huge']}>···
421     </div>
422     );
423   };
```

Рис. 2.40. Початок створення компонента композитного басейну

З самого початку додамо заголовок, а також картинку композитного басейну і його загальну характеристику.

```
<h3 className={utilities['u-margin-small']}>
  {t('composite-pools-title')}
</h3>

<div className={`${styles['swimming-pools__content']} ${styles['swimming-pools__content-60-40']}`}>
  <div className={styles['swimming-pools__content-left']}>
    {t('composite-pools-description')}
  </div>
  <div className={`${styles['swimming-pools__content-right']} ${styles['swimming-pools__content-img']}`}>
    <Image
      src='/swimming-pools/composite-pool.jpg'
      alt='Composite Pool'
      width={720}
      height={360}
      quality={100}
    />
  </div>
</div>
```

Рис. 2.41. Розробка верхньої частини компонента композитного басейну

Далі додамо приклади можливих форм виконання чаші басейну. Це будуть елементи, які складаються з картинки та назви. Розміщені вони будуть в ряд від одного до п’яти елемента в залежності від ширини екрану.

```
<h4 className={utilities['u-margin-small']}>
  {t('composite-pools-bowl-forms')}
</h4>

<div className={grid.row}>
  <div className={`${grid['col-1-of-5']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('composite-pools-bowl-forms-type-1')}
    </p>
    <Image
      src='/swimming-pools/forma-1.png'
      alt='Rectangular shape'
      width={151}
      height={151}
      quality={100}
    />
  </div>
  <div className={`${grid['col-1-of-5']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('composite-pools-bowl-forms-type-2')}
    </p>
    <Image
      src='/swimming-pools/forma-2.png'
      alt='Oval shape'
      width={151}
      height={151}
      quality={100}
    />
  </div>
  <div className={`${grid['col-1-of-5']} ${styles['swimming-pools__item']}`}>
    <p className={styles['swimming-pools__item-title']}>
      {t('composite-pools-bowl-forms-type-3')}
    </p>
    <Image
      src='/swimming-pools/forma-3.png'
      alt='Round shape'
```

Рис. 2.42. Розробка прикладу форм виконання чаші компонента композитного басейну

У будові композитного басейну є певні обмеження по габаритах. Цю інформацію потрібно показати клієнтові. Зробимо це у вигляді списку.

```
<h4 className={utilities['u-margin-bottom-small']}>
  {t('composite-pools-sizes')}
</h4>

<div className={styles['swimming-pools__list-wrapper']}>
  <p className={styles['swimming-pools__list-title']}>
    {t('composite-pools-sizes-description')}:
  </p>

  <ul className={styles['swimming-pools__list']}>
    <li>{t('composite-pools-sizes-width')}</li>
    <li>{t('composite-pools-sizes-length')}</li>
    <li>{t('composite-pools-sizes-depth')}</li>
  </ul>
</div>
```

Рис. 2.43. Розробка прикладу габаритів компонента композитного басейну

```
&__list {
  list-style-type: circle;

  &-wrapper {
    display: flex;
    flex-direction: column;
    align-items: center;
  }

  &-title {
    font-size: 2rem;
    margin: -1rem 0 1rem;
  }
}
```

Рис. 2.44. Приклад стилів списку

Лінійка композитних басейнів має безліч кольорових рішень. Відобразимо ці елементи у вигляді картинки з назвою. Картинка буде поділена на дві частини: у верхній частині показаний сам колір, у нижній – як він виглядає у воді. Елементи будуть розміщуватись в ряд від одного до п’яти в залежності від ширини екрану.

```
<h4 className={utilities['u-margin-small']}>
  {t('composite-pools-colors')}
</h4>

<div className={`${styles['swimming-pools__item']} ${utilities['u-padding-bottom-none']}`}>
  <p className={styles['swimming-pools__item-title']}>
    {t('composite-pools-colors-type-1')}
  </p>

  <div className={grid.row}>
    <div className={`${grid['col-1-of-5']} ${styles['swimming-pools__color-item']}`}>
      <div className={styles['swimming-pools__color-item-inner']}>
        <p className={`${styles['swimming-pools__color-text']} ${styles['swimming-pools__color-title']}`}>
          {t('composite-pools-colors-type-1-1')}
        </p>
        <Image
          src='/swimming-pools/color_pervansh.jpg'
          alt='Pervansh'
          width={189}
          height={151}
          quality={100}
        />
        <p className={`${styles['swimming-pools__color-text']} ${styles['swimming-pools__color-subtitle']}`}>
          {t('composite-pools-colors-in-water')}
        </p>
      </div>
    </div>
    <div className={`${grid['col-1-of-5']} ${styles['swimming-pools__color-item']}`}>
      <div className={styles['swimming-pools__color-item-inner']}>
        <p className={`${styles['swimming-pools__color-text']} ${styles['swimming-pools__color-title']}`}>
          {t('composite-pools-colors-type-1-2')}
        </p>
        <Image
          src='/swimming-pools/color_huko.jpg'
          alt='Huko'
          width={189}
          height={151}
          quality={100}
        />
        <p className={`${styles['swimming-pools__color-text']} ${styles['swimming-pools__color-subtitle']}`}>
          {t('composite-pools-colors-in-water')}
        </p>
```

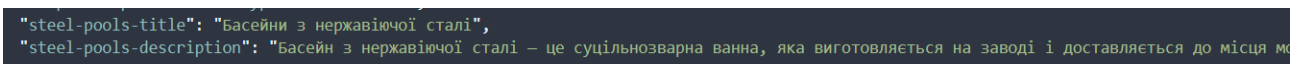
Рис. 2.45. Розробка кольорових рішень компонента композитного басейну

```
&__color {
  &-item {
    margin-bottom: 2rem;

    @include respond(xs-phone) {
      margin-right: 6rem;
      margin-left: 6rem;
    }

    &-inner {
      position: relative;
      max-width: 190px;
      margin: 0 auto;
    }

    img {
      width: 100%;
    }
  }

  &-text {
    position: absolute;
    right: 10px;
    z-index: 2;
    max-width: 150px;
    padding: 0 1rem;
    font-weight: 700;
    text-align: end;
    line-height: 1.4;
    border-radius: 2px;
    background-color: rgba($color-whisper, 0.8);
  }

  &-title {
    top: 10px;
    z-index: 2;
  }

  &-subtitle {
    bottom: 15px;
    line-height: 1.5;
  }
}
```

Рис. 2.46. Приклад стилів для кольорових рішень
компонента композитного басейну

З розробкою компонента композитного басейну завершено. Перейдемо у файл *steel-pools.js* та почнемо розроблювати компонент басейну з нержавіючої сталі. Цей компонент буде найпростішим і складатись лише з загальної інформації про басейн.

Спершу додамо текст у файл *swimming-pools.json*.



```
"steel-pools-title": "Басейни з нержавіючої сталі",
"steel-pools-description": "Басейн з нержавіючої сталі – це суцільнозварна ванна, яка виготовляється на заводі і доставляється до місця мо
```

Рис. 2.47. Приклад створення українського тексту
для компонента басейну з нержавіючої сталі

Далі створимо компонент басейну по аналогії компонентів створених раніше.

Рис. 2.48. Розробка компонента басейну з нержавіючої сталі

Всі компоненти створені. Тепер можемо перейти у головний файл сторінки, який знаходиться в папці *pages* > *swimming-pools.js*. В нього заімпортимо контейнер, в який передамо необхідні параметри такі як назва, опис та ключові слова для SEO-оптимізації. Також всередину контейнера передамо щойно створені компоненти басейнів.

Рис. 2.49. Розробка сторінки басейнів

2.4.3. Розробка функціональності «Розрахунок вартості проекту»

Функціональність «Розрахунок вартості проекту» дасть клієнтові зрозуміти у скільки йому обійдеться будівництво басейна на власній території. Інтерфейс буде розроблено у зручний для клієнта манер і під кожен тип басейну окремо. Відповідно до прототипу з підрозділу 2.2 це буде кнопка внизу кожного компонента басейну. По кліку на цю кнопку, з'являтиметься модальне вікно. В ньому буде знаходитись форма, в якій потрібно буде вказати габарити проекту, а також інші додаткові особливості. Після заповнення всіх полів натискається кнопка і виводиться результат вартості проекту. Перейдемо до розробки.

Спершу створимо компонент модального вікна. Основна мета таких компонентів – відображатися поверх решти елементів сторінки. З цієї причини, як правило, добре розміщувати модальне вікно в окремому вузлі DOM, за межами решти інтерфейсів користувача. Однак у класичному React SPA програма відтворюється всередині вказаного вузла DOM, тож нам у таких сценаріях API React пропонує функцію під назвою Portal. Вона дозволяє компонентам відображати в певному вузлі DOM за межами кореневого вузла,

який використовує програма. Додамо обгортку для модального вікна у компонент контейнера. У цю обгортку і буде вставлятись модальне вікно.

Рис. 2.50. Створення обгортки для модального вікна

Тепер перейдемо до створення самого модального вікна. У папці *components* > *modal* створюємо два файли: *js* для компонента та *scss* для стилів. Базова структура компоненту буде виглядати так:

Рис. 2.51. Базова структура модального вікна

Оскільки ми працюємо зі середовищем SSR, наш код може працювати як на стороні сервера, так і на стороні браузера. І перед тим як створювати портал, ми повинні переконатися, що знаходимось у браузері, адже об'єкт *window.document* доступний тільки там. Робимо це за допомогою функції *useEffect*. Після того, як базова структура створена, можемо додати й іншу логіку. Компонент модального вікна буде приймати чотири параметри: *show*, *onClose*, *title* і *children*. За допомогою *show* будемо визначати чи показувати компонент, чи ні. *onClose* – функція, яка відповідає за закриття модального вікна. *Title* – назва і *children* – внутрішні компоненти.

```
export const Modal = ({ show, onClose, title, children }) => {
  const [isBrowser, setIsBrowser] = useState(false);
  const { t } = useTranslation('swimming-pools');

  useEffect(() => {
    setIsBrowser(true);
  }, []);

  const handleCloseClick = () => {
    onClose();
  };

  const modalContent = show ? (
    <div
      className={styles.overlay}
      onClick={(e) => (e.target.className.includes('overlay') ? onClose() : null)}>
      <div className={styles.modal}>
        <button className={styles['modal__button']} onClick={handleCloseClick}>
          <img src='/icons/close.png' alt='close' width='25' height='25' />
        </button>
        <div className={styles['modal__header']}>
          <h3>{title}</h3>
          <p>{t('calculate-cost-description')}</p>
        </div>
        <div className={styles['modal__body']}>
          {children}
        </div>
      </div>
    </div>
  ) : null;

  if (isBrowser) {
    return ReactDOM.createPortal(
      modalContent,
      document.getElementById("modal-root")
    );
  } else {
    return null;
  }
```

Рис. 2.52. Розробка модального вікна

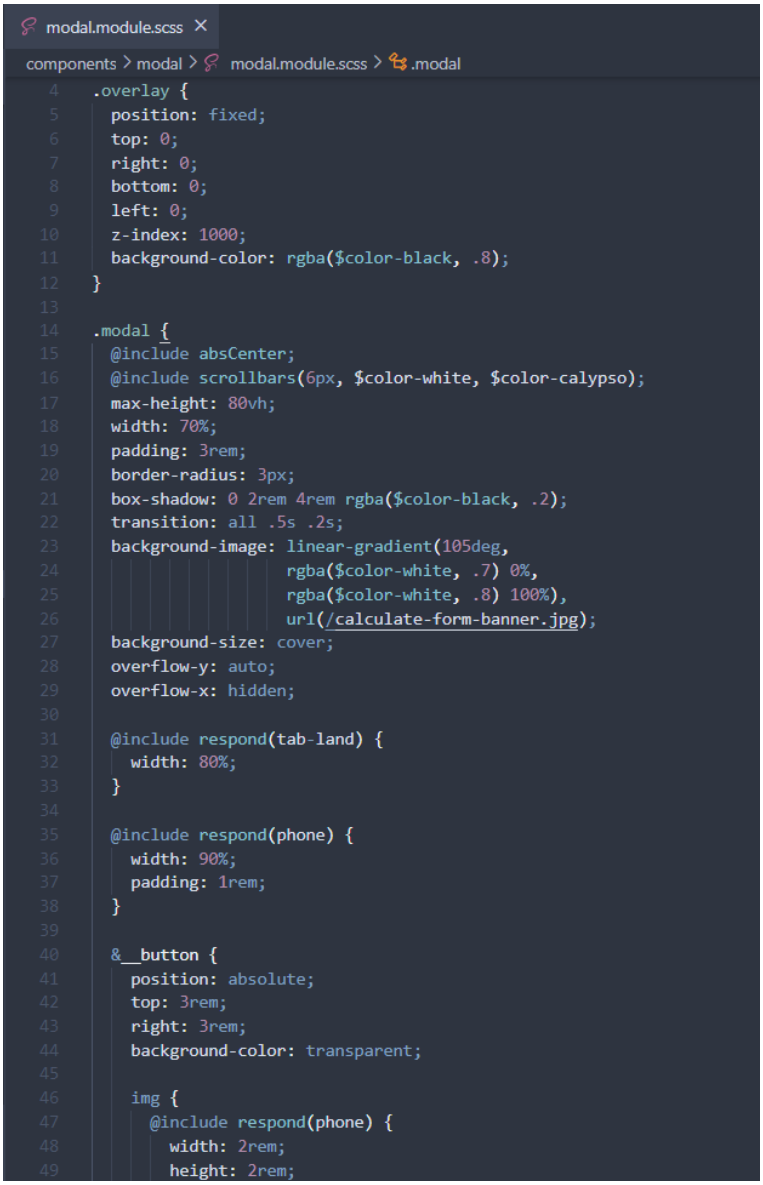


Рис. 2.53. Приклад стилів для модального вікна

Додамо модальне вікно у всі компоненти басейнів. Для прикладу візьмемо композитний басейн. Зверху заімпортуємо компонент модального вікна. У компоненті басейнів створимо умову і функції для відображення/закриття модального вікна і передамо їх у компонент. Створимо також кнопку, по кліку на яку і буде відкриватись модальне вікно.

```
import { Modal } from '../../../components';

export const CompositePools = () => {
  const { t } = useTranslation('swimming-pools');
  const [showModal, setShowModal] = useState(false);

  const onButtonClick = () => {
    setShowModal(true);
    document.documentElement.style.overflow = 'hidden';
  };

  const onModalClose = () => {
    setShowModal(false);
    document.documentElement.style.overflow = 'auto';
  };
```

Рис. 2.54. Створення базових функцій для модального вікна

```
<div className={`${utilities['u-center-text']} ${utilities['u-margin-top-medium']}`}>
  <button className={utilities['u-button']} onClick={onButtonClick}>
    {t('swimming-pools-calculate-cost')}
  </button>
  <Modal
    onClose={onModalClose}
    show={showModal}
    title={t('calculate-composite-pools-cost-title')}
  >
  </Modal>
</div>
```

Рис. 2.55. Створення кнопки з модальним вікном

Перейдемо до створення форм, які будуть передаватись у тіло модального вікна. Ці компоненти будуть знаходитись у папці *components* > *form*. Всього буде три форми – окрема форма під кожен тип басейну. У всіх формах буде використовуватись елемент *<input />*, тому створимо його окремим компонентом з різними властивостями. Компонент вводу буде приймати тип, ідентифікатор, різні додаткові атрибути та функції, імена класів для стилів тощо.

Рис. 2.56. Створення компонента для форми

У локалізованих файлах *swimming-pools.json* додамо весь необхідний текстовий контент.

Рис. 2.57. Приклад створення українського тексту
для розрахункових форм

У всіх формах потрібно буде вводити габарити, а також різні додаткові властивості, такі як тип водообміну, нагрівача, освітлення тощо. Більшість однакових властивостей будуть у формах, але деякі й будуть відрізнятись, тому що кожен басейн має свої особливості. Для прикладу створимо форму для композитного басейну. Вона буде складатися з полів призначених для вводу ширини, висоти, глибини басейну. З особливостей це буде вибір форми чаші, кольору, типу закладних деталей, нагрівача, освітлення, системи управління та дезінфекції. Дані будемо встановлювати за допомогою функції *useState* при події *onChange*.

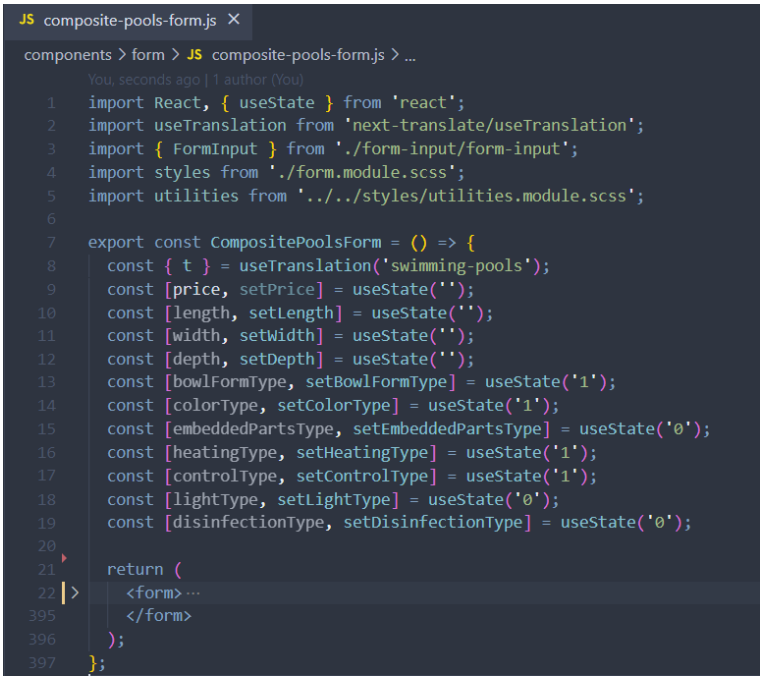


Рис. 2.58. Створення базової структури розрахункової форми композитного басейну

У формі будемо використовувати елемент *FormInput* і передавати в нього різні атрибути:

- required – у поле обов’язково потрібно ввести значення;
- min, max – мінімальне/максимальне значення;
- step – з яким кроком змінюється значення;
- value – значення поля;
- defaultChecked – початкове значення у випадку з радіо значеннями.

```
<FormInput
  type={'number'}
  id={'width'}
  inputWrapperClassName={styles['form__input-wrapper']}
  inputClassName={styles['form__input']}
  placeholder={t('calculation-form-width')}
  label={t('calculation-form-width')}
  labelClassName={styles['form__label']}
  onChange={value => setWidth(value)}
  required
  min={1}
  max={4.25}
  step={'.05'}
/>

<FormInput
  type={'number'}
  id={'depth'}
  inputWrapperClassName={styles['form__input-wrapper']}
  inputClassName={styles['form__input']}
  placeholder={t('calculation-form-depth')}
  label={t('calculation-form-depth')}
  labelClassName={styles['form__label']}
  onChange={value => setDepth(value)}
  required
  min={0.85}
  max={2}
  step={'.05'}
/>
</div>

<h4 className={styles['form__radio-title']}>{t('calculation-form-bowl-form-title')}</h4>
<div className={styles['form__radio-group']}>
<FormInput
  type={'radio'}
  id={'bowl-form-1'}
  inputWrapperClassName={styles['form__radio-input-wrapper']}
  inputClassName={styles['form__radio-input']}
  name={'bowl-form'}
  label={t('calculation-form-bowl-form-type-1')}
  labelClassName={styles['form__radio-label']}
  spanClassName={styles['form__radio-button']}
  defaultChecked={'1'}
  value={'1'}
  onChange={value => setBowlFormType(value)}
/>
```

Рис. 2.59. Створення полів для розрахункової форми композитного басейну

Внизу форми будемо виводити ціну. Також внизу буде знаходитись кнопка типу *submit*, після натиску на яку і почнеться відбуватись розрахунок вартості проекту.

```
    value={'0'}
    onChange={value => setDisinfectionType(value)}
  />
</div>

{price &&
  <h4 className={styles['form__price-title']}>{t('calculate-cost-result')}  
    <span className={styles['form__price']}>{price} &#8372;</span>
  </h4>
}

<div className={utilities['u-center-text']}>
  <button type='submit' className={`${styles['form__submit-button']} ${utilities['u-button']}`}>
    {t('swimming-pools-calculate-cost')}
  </button>
</div>
</form>
```

Рис. 2.60. Відображення ціни та кнопки розрахункової форми композитного басейну

Створимо API, яке буде займатись розрахунком вартості проекту у папці *pages* > *api*. У ньому буде знаходитись прейскурант, в якому буде вказана ціна за кожен тип робіт.

Рис. 2.61. Розробка прейскуранту

З тіла запиту отримаємо всі значення. З першу буде розраховуватись площа басейну за формулою $S = (a \cdot b) + 2 \cdot ((c \cdot a) + (c \cdot b)) = x$ ($м^2$), де a – довжина, b – ширина, c – глибина.

Після площі відповідно до типу басейну буде розраховуватись вартість певних особливостей, а внизу вартість особливостей, наявних у всіх типах басейнів.

Результат повернемо у форматі JSON з використанням функції *Intl.NumberFormat*, яка поверне число у правильному форматі.

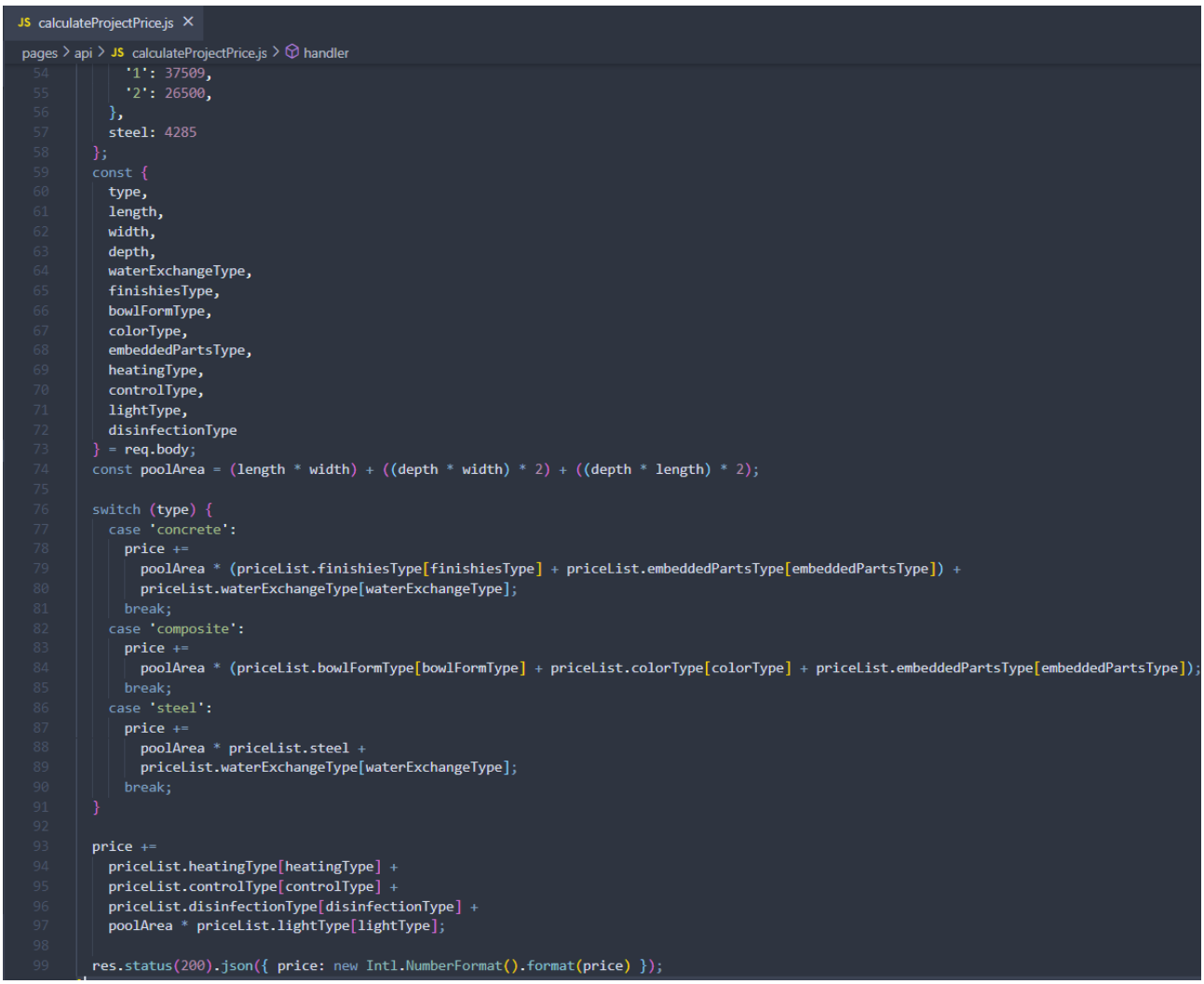


Рис. 2.62. Розробка функції для розрахунку вартості проекту

Викличемо щойно створене API у формі та виведемо результат. Для цього будемо використовувати конструкцію *async/await*. Ця конструкція призначена для написання асинхронного коду – це означає, що він не буде блокувати основний потік. Async – створює асинхронну функцію, await – очікує на результат. Для запиту на сервер буде використовуватись функція *fetch*, в яку передамо шлях до API; дані у JSON-форматі; заголовок, який сповіщає про тип даних контенту; метод запиту. Після отримання результату, встановимо ціну за допомогою функції *setPrice*.

```
const submitForm = async event => {
  event.preventDefault()

  const res = await fetch('/api/calculateProjectPrice', {
    body: JSON.stringify({
      type: 'composite',
      length,
      width,
      depth,
      bowlFormType,
      colorType,
      embeddedPartsType,
      heatingType,
      controlType,
      lightType,
      disinfectionType
    }),
    headers: {
      'Content-Type': 'application/json'
    },
    method: 'POST'
  })

  const result = await res.json();
  setPrice(result.price);
}

return (
  <form onSubmit={submitForm}>
    <div className={styles['form__group']}>
```

Рис. 2.63. Виклик функції розрахунку вартості та виведення результату

РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАЧА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Опис інтерфейсу інформаційної системи

В дипломній роботі представлено розробку веборієнтованої інформаційної системи компанії будівництва басейнів. Ця система є кросплатформною (тобто доступною з будь-якого браузера) та адаптивною для різних пристроїв. Таким чином, з нею буде приємно та легко взаємодіяти. Інформаційна система підтримує дві мови – українську та російську, між якими користувач може перемикатись. Даний проект розподілений на п’ять сторінок. Кожна сторінка містить в собі розгорнуту інформацію про діяльність компанії, головна сторінка – стислу інформацію. Почнемо з головної сторінки ознайомлення з результатом роботи.

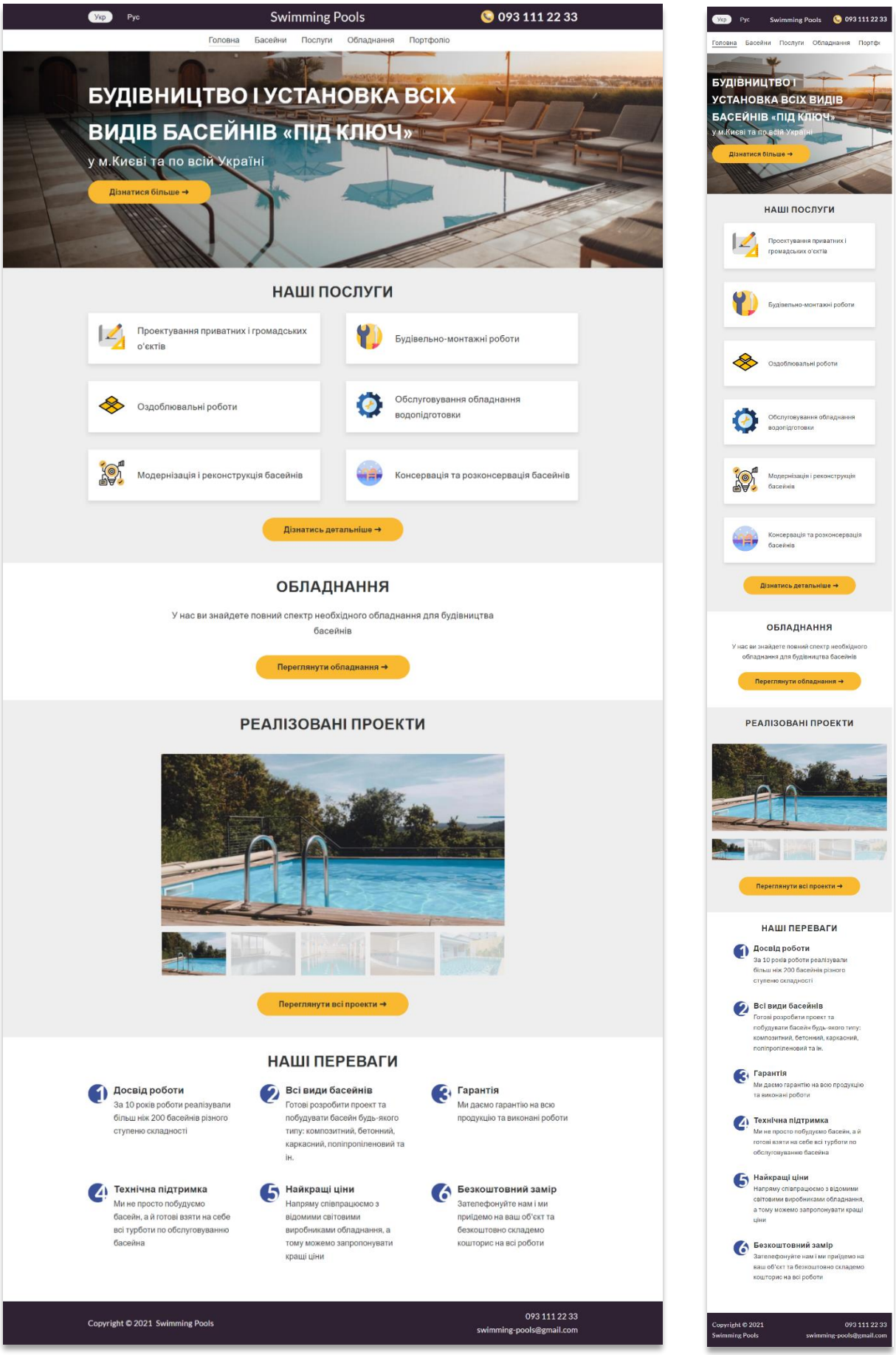


Рис. 3.1. Вигляд головної сторінки для настільних та мобільних пристроїв

Головна сторінка розподілена на секції. Кожна секція містить в собі лаконічну інформацію, а також кнопку, при натисканні на яку можна перейти на основну сторінку і прочитати детальну інформацію. Також на детальні сторінки можна перейти за допомогою основного меню, яке знаходиться зверху. Для зручності меню завжди буде перед очима користувача, наскільки б сильно вниз він не проскролив сторінку. У «шапці» користувач може перемкнути мову сайту. Білим кольором підсвічена активна локаль. Зверху і знизу знаходиться номер телефону компанії. При натисканні на нього, автоматично з’явиться застосунок для дзвінків, якщо його підтримує пристрій. Також знизу знаходиться адреса електронної пошти. Аналогічно при натисканні на неї з’являється застосунок для пошти, якщо його підтримує пристрій.

Розглянемо наступну сторінку про басейни.

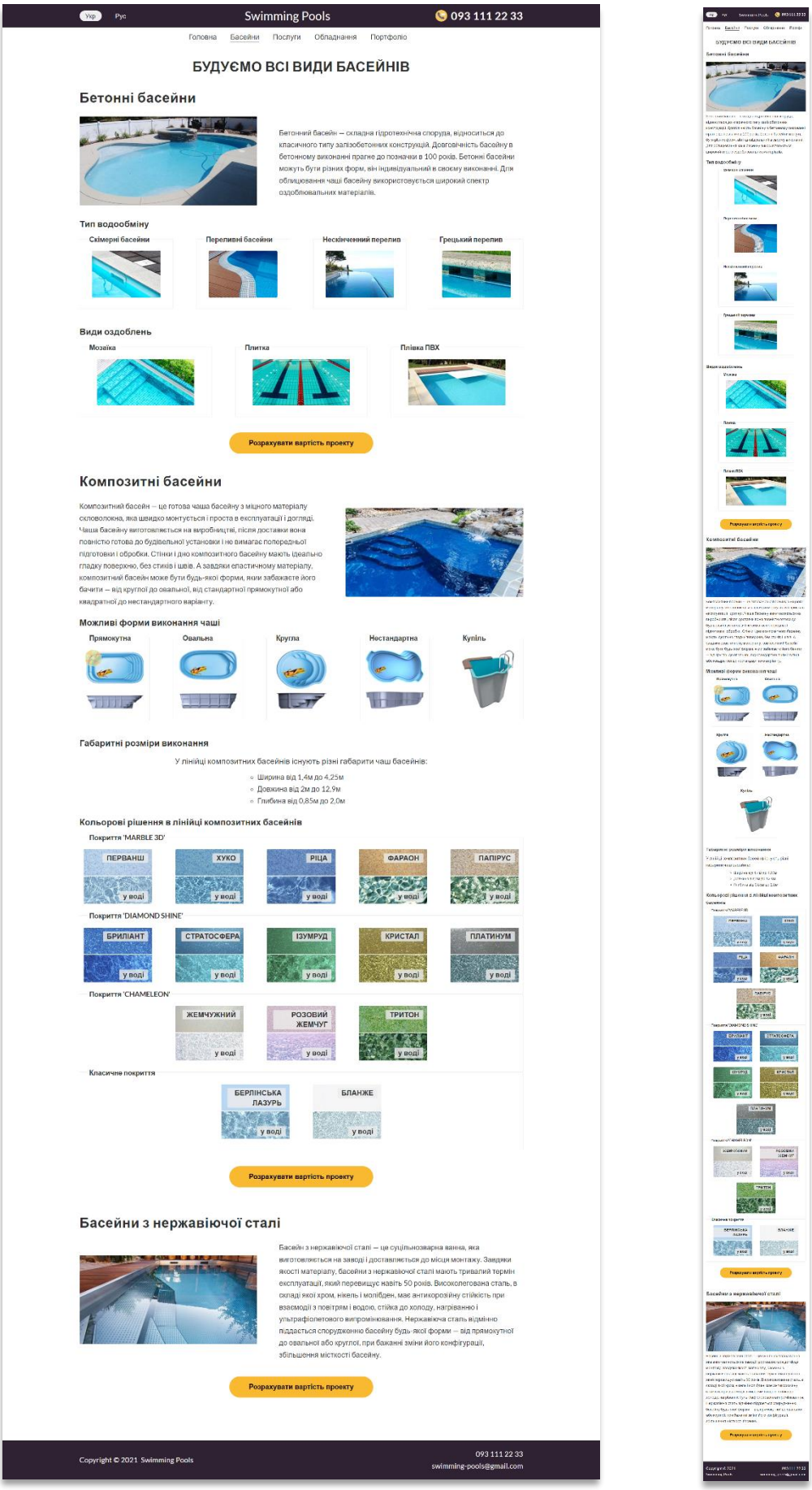


Рис. 3.2. Вигляд сторінки басейнів для настільних та мобільних пристроїв

На цій сторінці представлена детальна інформація про види басейнів будівництвом яких займається компанія «Swimming Pools». Всього маємо три секції. Кожна секція містить в собі назву басейну, фотографію та опис. В залежності від виду басейну, маємо додаткову інформацію. Наприклад, у випадку з композитними басейнами представлені можливі форми та габарити виконання чаші, а також доступні кольорові рішення.

Для кожного басейну можемо розрахувати приблизну вартість проекту. Ця функція дуже корисна, адже клієнтові відразу стане зрозуміло у скільки йому обійдеться проект. Щоб це зробити, потрібно натиснути на кнопку «Розрахувати вартість проекту» і заповнити поля форми. Форма виглядає наступним чином.

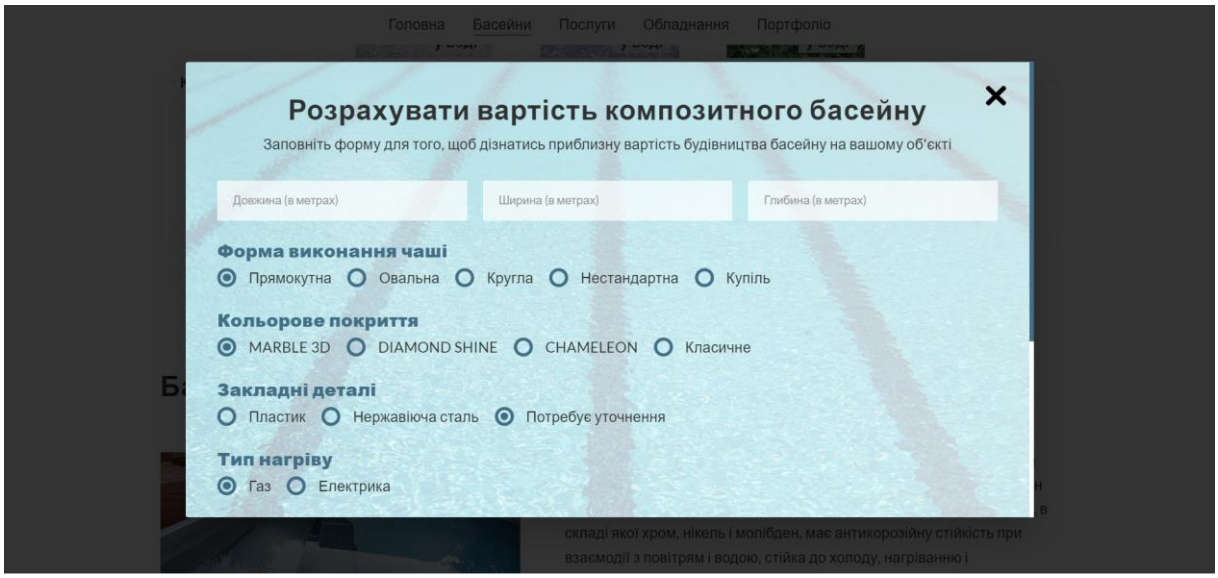


Рис. 3.3. Приклад форми розрахунку вартості проекту

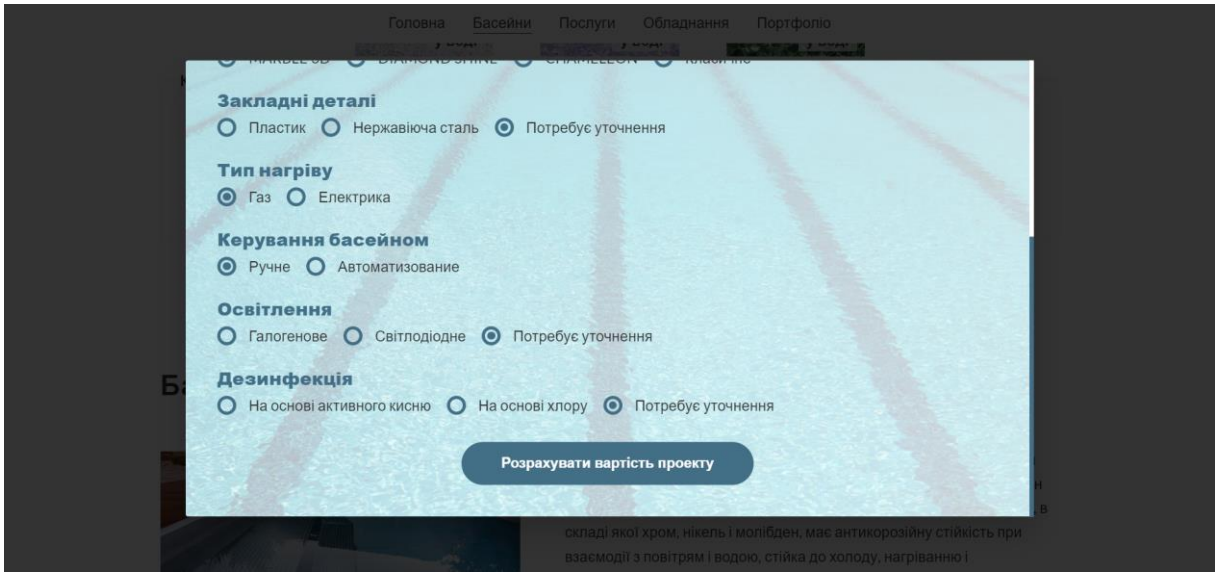


Рис. 3.4. Приклад форми розрахунку вартості проекту

Спершу потрібно ввести габарити проекту. При правильному введені числа, поле буде підсвічуватись зеленим кольором.

Рис. 3.5. Приклад успішного заповнення поля форми

При неправильному – червоним. До того ж буде виведено підказку, яке число очікується від користувача.

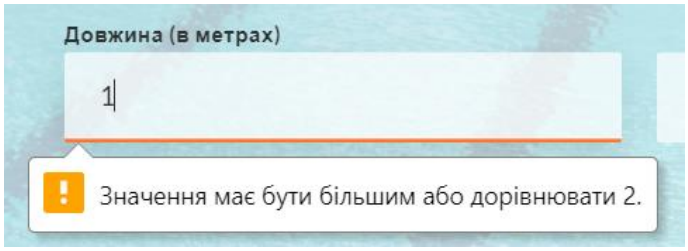


Рис. 3.6. Приклад помилкового заповнення поля форми

Після заповнення усіх даних можна натиснути кнопку «Розрахувати вартість проекту» і побачити результат.

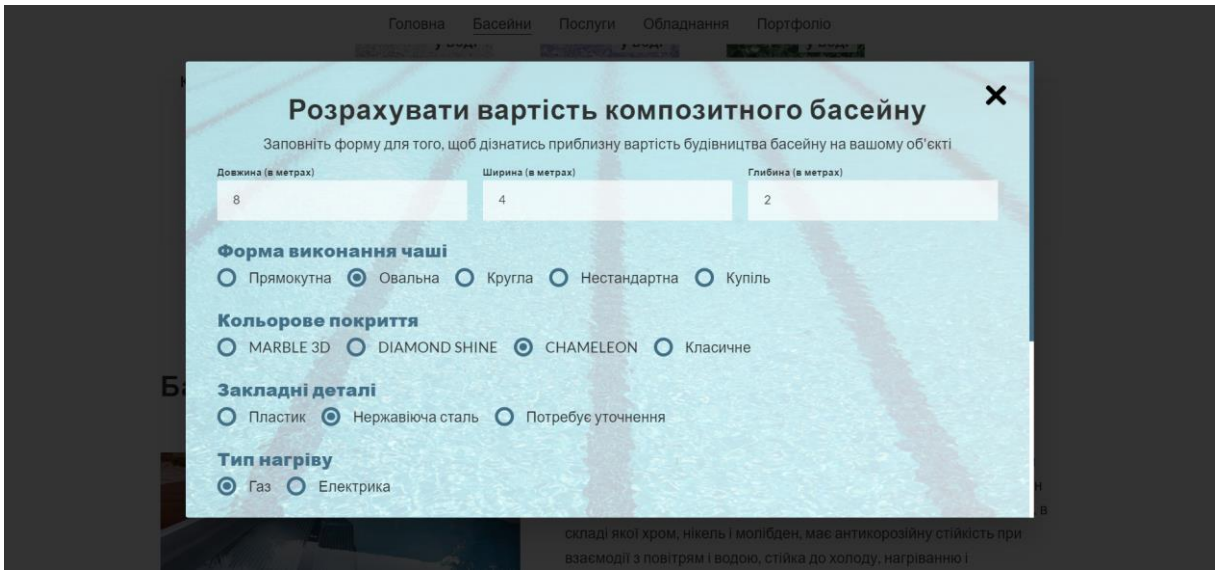


Рис. 3.7. Приклад заповнення форми розрахунку вартості проекту

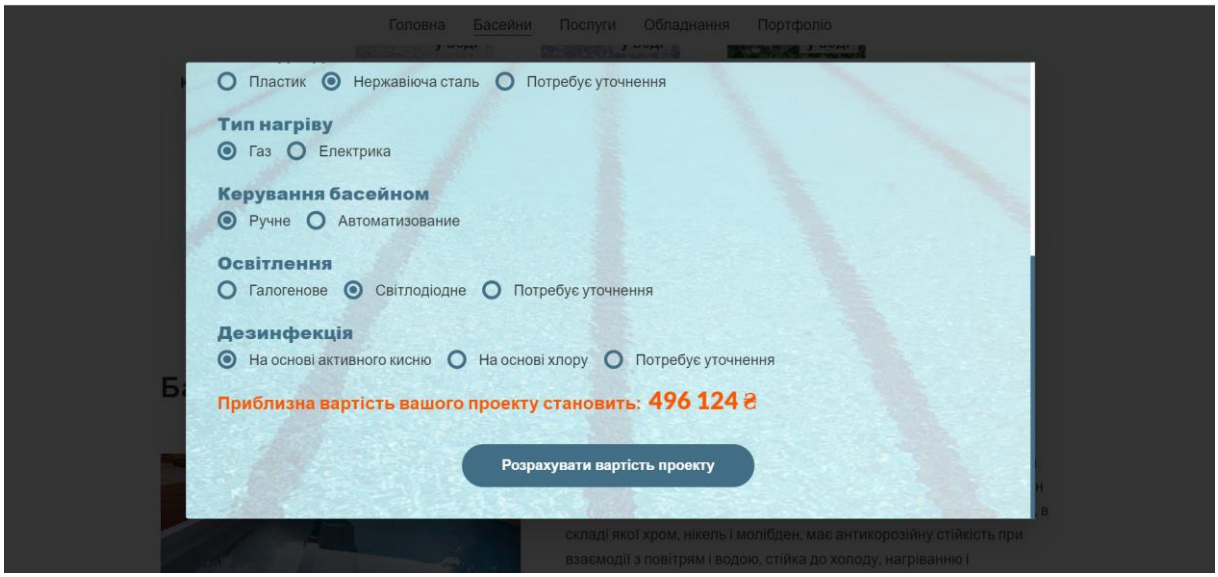
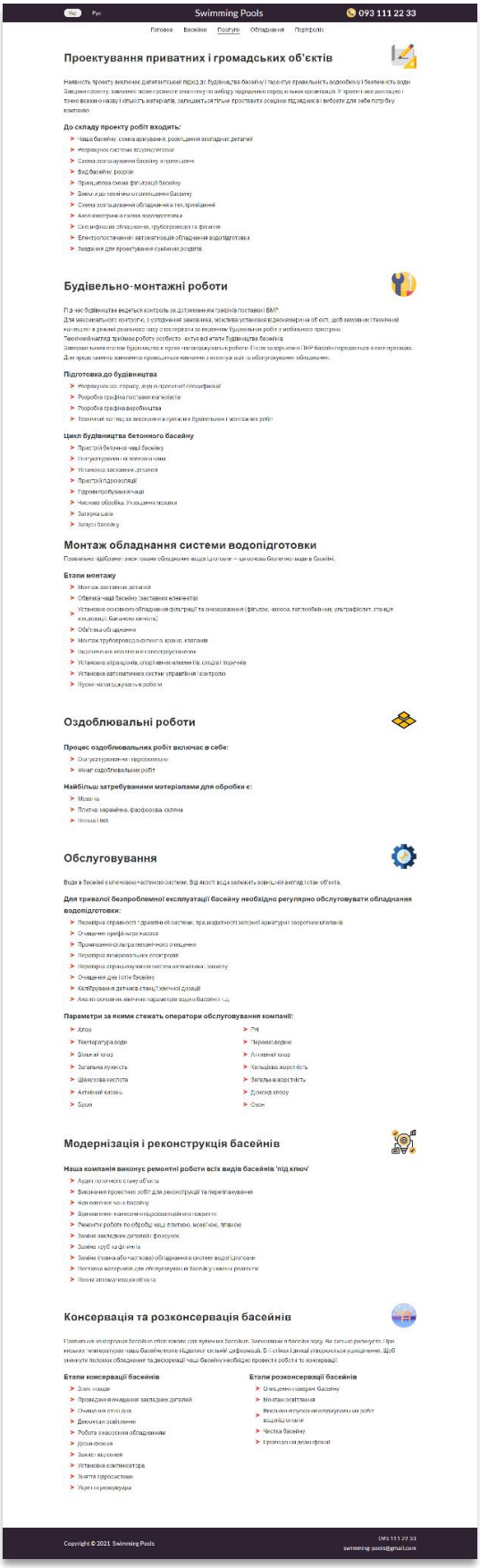


Рис. 3.8. Приклад заповнення форми розрахунку вартості проекту і виведення результату

Якщо користувач хоче змінити якесь значення, він може з легкістю це зробити і натиснути кнопку знову. Розрахунок відбудеться в автоматичному режимі і він відразу побачить новий результат. Після закриття модального вікна форма очищається і всі попередньо-введені дані зникають. Модальне вікно можна закрити за допомогою кнопки «×», яка знаходиться в правому верхньому кутку, або натиснути на темну область навколо модального вікна.

Перейдемо до наступної сторінки – сторінки послуг.



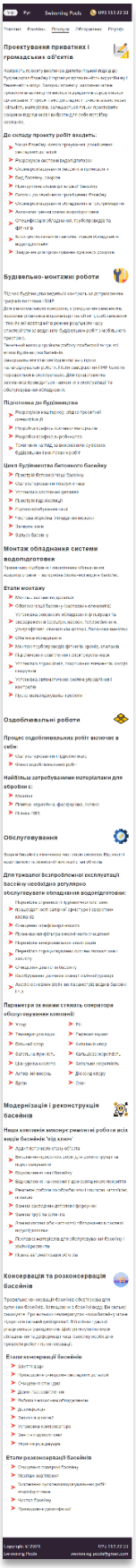


Рис. 3.9. Вигляд сторінки послуг для настільних та мобільних пристроїв

Під кожен вид послуги створена окрема секція, в якій детально розписана вся необхідна інформація.

Наступна сторінка про обладнання.

Рис. 3.10. Вигляд сторінки обладнання для настільних та мобільних пристроїв

Спершу вказана загальна інформація про вимоги до обладнання і про умови роботи компанії «Swimming Pools». Потім показано весь перелік доступного обладнання для будівництва басейнів з фотографіями.

Остання сторінка інформаційної системи – це сторінка портфоліо.

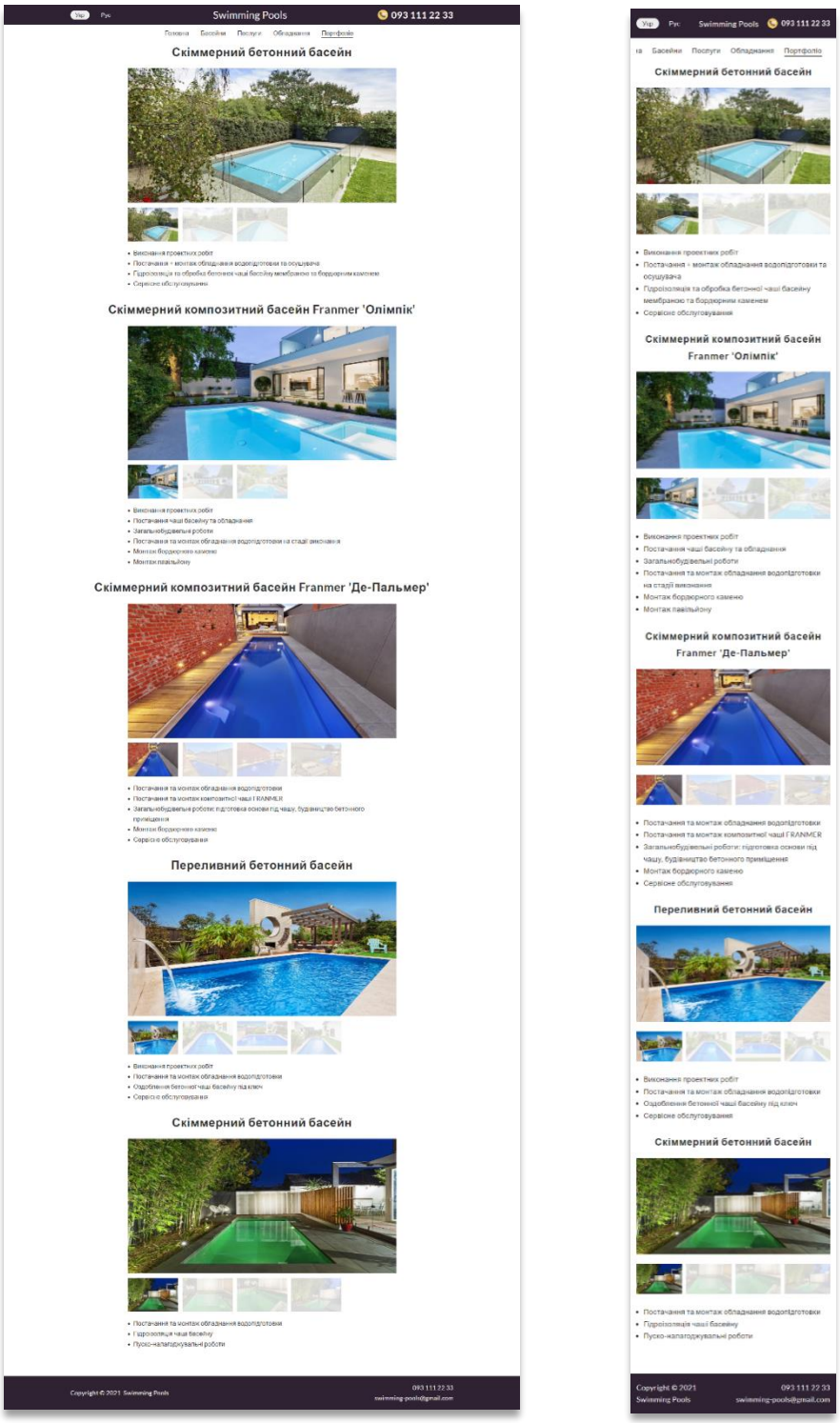


Рис. 3.11. Вигляд сторінки портфоліо для настільних та мобільних пристроїв

На цій сторінці представлені фото-результати робіт з коротким описом, що було зроблено. Фотогалерею можна передивлюватись за допомогою функції «свайп», яка є у сучасних мобільних пристроях, або при натисканні на маленькі приклади фотографій. Активна фотографія підсвічена яскраво, всі решта мають низьку прозорість.

3.2. Порівняльний аналіз вартості будівництва басейнів різних видів

Проведемо порівняльний аналіз вартості будівництва басейну в залежності від виду, адже для багатьох клієнтів ціна має вирішальне значення у виборі басейну. Для аналізу будемо враховувати басейни з однаковими габаритами та властивостями. По деяких властивостях можуть бути відмінності, тому що не кожен басейн ними володіє. Всі розрахунки будуть проводитись за допомогою розробленої розрахункової форми. Результат приведено у таблиці 3.1.

Таблиця 3.1

Порівняльний аналіз вартості будівництва басейнів

Властивість	Вид басейну		
	Бетонний	Композитний	З нержавіючої сталі
Довжина (в метрах)	8	8	8
Ширина (в метрах)	4	4	4
Глибина (в метрах)	1.8	1.8	1.8
Тип водообміну	Скімерний	–	Скімерний
Вид оздоблення	Мозаїка	–	–
Закладні деталі	Нержавіюча сталь	Нержавіюча сталь	–
Тип нагріву	Електрика	Електрика	Електрика
Керування басейном	Ручне	Ручне	Ручне
Освітлення	Світлодіодне	Світлодіодне	Світлодіодне
Дезинфекція	На основі хлору	На основі хлору	На основі хлору
Форма виконання чаші	–	Прямокутна	–
Кольорове покриття	–	Класичне	–
Вартість (в гривнях)	459 397,2	454 660,8	441 454,2

Таким чином, можна зробити висновок, що при однакових властивостях найдешевше обійдеться будівництво басейна з нержавіючої сталі, а найдорожче – з бетону.

ВИСНОВКИ

У результаті виконаної роботи було розроблено веборієнтовану інформаційну систему компанії будівництва басейнів, яка надає повну інформацію про компанію, її вид діяльності, послуги тощо. Додатково була розроблена функціональність «Розрахунок вартості проекту», яка дає змогу клієнтам дізнатись ціну будівництва басейну на їхній території. Доступ до системи можна отримати з різних пристроїв та браузерів, адже вебсайт є кросплатформним та адаптивним.

У процесі виконання роботи було розглянуто сучасні засоби та техніки розробки, відзначені їхні позитивні та негативні сторони, проведено порівняльний аналіз між технологіями. За основу було взято мову програмування JavaScript, яка найчастіше використовується у веброзробці. Для реалізації інформаційної системи використовувався фреймворк Next.js та багато інших сумісних з ним бібліотек. За допомогою цього фреймворку, інформаційна система отримує хорошу SEO-оптимізацію, і, як результат, високу індексацію та ранжирування сайту, що значно сприяє залученню нових клієнтів.

Всі текстові дані зберігаються у файлах формату json під кожну локаль і сторінку інформаційної системи окремо. Такий спосіб полегшує ймовірну модифікацію чи розширення даних.

Завдяки тому, що програмний продукт написаний за допомогою сучасної мови програмування та її провідних технологій, гарантується з часом підтримка всіх компонентів та їх подальша модернізація.

Реалізований продукт виконано у повному обсязі і повністю задовольняє поставлену задачу. Інформаційна система працює ефективно та надійно і покриває всі необхідні потреби компанії будівництва басейнів.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Client-server application [Електронний ресурс]. – 2019. – Режим доступу: https://madooei.github.io/cs421_sp20_homepage/client-server-app/ (дата звернення 16.10.2021). – Назва з екрана.
2. What is JavaScript? [Електронний ресурс]. – 2020. – Режим доступу: https://www.javascripttutorial.net/what-is-javascript/ (дата звернення 18.10.2021). – Назва з екрана.
3. JavaScript overview [Електронний ресурс]. – 2020. – Режим доступу: https://www.tutorialspoint.com/javascript/javascript_overview.htm (дата звернення 18.10.2021). – Назва з екрана.
4. Angular vs React vs Vue [Електронний ресурс]. – 2020. – Режим доступу: https://merehead.com/ru/blog/angular-vs-react-vs-vue-2021/ (дата звернення 20.10.2021). – Назва з екрана.
5. Next.js Overview [Електронний ресурс]. – 2021. – Режим доступу: https://www.tutorialspoint.com/nextjs/nextjs_overview.htm (дата звернення 26.10.2021). – Назва з екрана.
6. WebStorm [Електронний ресурс]. – 2019. – Режим доступу: https://uk.wikipedia.org/wiki/WebStorm (дата звернення 29.10.2021). – Назва з екрана.
7. Visual Studio Code [Електронний ресурс]. – 2020. – Режим доступу: https://ru.wikipedia.org/wiki/Visual_Studio_Code (дата звернення 29.10.2021). – Назва з екрана.