НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**

Завідувач кафедри
комп'ютерних наук

_____/Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«_04_»__червня____2025 р

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

**«Автоматизована система проєктування графічних об'єктів»**

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
_____к.т.н., доцент_____/ Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :_____/ Панкратьєв В.О/
підпис ПІБ

Виконав:_____/ Клименко Б. Д./
підпис ПІБ

**КИЇВ-2023**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ЗАТВЕРДЖУЮ**

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

"16" грудня 2024 р.

# ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Клименко Борису Денисовичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Автоматизована система проєктування графічних об'єктів

Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи _____ / Панкратьєв В.О. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Клименко Б. Д. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
рік, місяць, число

# ЗМІСТ

# ВСТУП

У сучасному цифровому світі питання зручності та доступності інтерфейсів користувача набуває особливого значення. Із стрімким розвитком веб-технологій та широким впровадженням цифрових сервісів, що охоплюють майже всі сфери життєдіяльності людини — від онлайн-банкінгу до електронної комерції та урядових послуг — виникає потреба у створенні інтуїтивно зрозумілих, естетично привабливих і функціональних користувацьких інтерфейсів. Одним із ключових елементів таких інтерфейсів є кнопки, що слугують основними засобами взаємодії користувача з системою.

Незважаючи на їхню просту структуру з технічної точки зору, кнопки виконують критично важливу роль в управлінні потоками даних, активації функцій, навігації між компонентами веб-застосунків та відображенні зворотного зв’язку. Саме тому правильна побудова кнопок — як з візуальної, так і з функціональної точки зору — є важливою складовою сучасної веб-розробки. При цьому велика кількість дизайнерських рішень, стилів, кольорових схем та анімацій ставить перед розробниками завдання не лише технічного впровадження, а й забезпечення адаптивності та уніфікованості зовнішнього вигляду UI-елементів.

Зважаючи на це, актуальною є розробка програмного засобу, що дозволяє спростити створення користувацьких кнопок без необхідності глибокого знання HTML та CSS. Такий інструмент повинен забезпечити гнучке налаштування основних параметрів кнопки — тексту, кольору, шрифтів, фону, скруглення, тіні, градієнтів, іконок, а також інтерактивної поведінки у вигляді анімацій при наведенні або натисканні.

Метою даної дипломної роботи є створення веб-застосунку типу "UI Button Builder", який забезпечує візуальне конструювання HTML/CSS-кнопок у зручному інтерфейсі. У додатку користувач може в режимі реального часу

змінювати зовнішній вигляд кнопки, бачити live-прев’ю, а також копіювати готовий код для подальшого використання у власних проєктах.

У процесі розробки застосунку було використано мову розмітки HTML, таблиці стилів CSS, мову JavaScript для динамічної взаємодії, а також фреймворк Bootstrap для адаптивної верстки та підтримки сучасного дизайну. Було реалізовано можливість додавання іконок, створення градієнтного фону, налаштування hover-ефектів, а також візуальні анімації (типу "ripple" або "loading") для натискання. Особливу увагу приділено зручності користувача та гнучкості конфігурацій.

Хоча в рамках даного проєкту не передбачено збереження даних у реальній базі, для відповідності структурі дипломної роботи було змодельовано уявну базу даних, яка містить інформацію про користувачів та їх збережені кнопки. Це дозволило створити ER-діаграму, уявну структуру СУБД та SQL-запити, які демонструють можливість масштабування системи в майбутньому.

Таким чином, розроблений веб-застосунок дозволяє ефективно вирішити задачу швидкого та якісного створення інтерфейсних кнопок, що особливо актуально як для новачків у веб-розробці, так і для досвідчених спеціалістів, яким потрібен швидкий генератор стильних UI-рішень.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Постановка задачі

У сучасному світі стрімкого розвитку інформаційних технологій користувацький інтерфейс (UI — User Interface) стає не просто інструментом для взаємодії з цифровими системами, а ключовим чинником зручності, доступності та ефективності роботи з ними. Особливо це стосується веб-застосунків, які використовуються в різноманітних сферах — від освіти й бізнесу до розваг та управління проєктами. Якість і продуманість інтерфейсних елементів безпосередньо впливає на враження користувача, швидкість виконання завдань і навіть комерційний успіх продукту.

Одним із основних елементів UI у веб-середовищі є кнопки. Вони забезпечують взаємодію користувача із застосунком: ініціюють дії, навігацію, надсилання форм, підтвердження вибору тощо. При цьому сучасні вимоги до кнопок включають не лише функціональність, а й естетичну привабливість, відповідність загальному дизайну, адаптивність до різних пристроїв, наявність анімацій та інтерактивності. Розробка таких елементів вручну часто вимагає від розробника глибоких знань HTML, CSS, JavaScript, а також навичок у сфері дизайну інтерфейсів.

Таким чином, виникає потреба в інструменті, який дозволив би спростити процес створення UI-кнопок для широкого кола користувачів — від новачків до досвідчених фахівців. Ідеальним рішенням виступає веб-додаток-конструктор, що надає змогу налаштовувати кнопку в інтерактивному режимі, змінюючи її текст, колір, фон, шрифт, скруглення кутів, тіні, іконки, hover-ефекти, анімацію при кліку тощо. Отриманий результат повинен виводитися у вигляді прев’ю та відповідного HTML/CSS коду, який можна скопіювати для подальшого використання.

Основна мета дипломної роботи — розробити веб-застосунок, який реалізує функціонал візуального конструктора інтерфейсних кнопок.

Для досягнення цієї мети необхідно розв’язати наступні задачі:

- провести аналіз предметної області та визначити функціональні вимоги до системи;
- змоделювати структуру майбутньої системи з використанням UML-діаграм (прецедентів, діяльності, послідовності);
- побудувати умовну структуру бази даних для демонстрації інформаційного забезпечення системи;
- реалізувати прототип веб-застосунку з використанням HTML, CSS, JavaScript та фреймворку Bootstrap;
- забезпечити інтерактивність елементів, включаючи live-прев’ю, ripple-анімацію при кліку, "loading"-ефект та анімації при наведенні;
- реалізувати копіювання згенерованого коду у буфер обміну;
- здійснити тестування функціональності та перевірити коректність роботи ключових компонентів;
- описати можливі напрямки подальшого розвитку системи — наприклад, збереження шаблонів, авторизацію користувачів, експорт у форматах PNG/SVG, підключення до реальної БД тощо.

Таким чином, поставлена задача охоплює як розробку програмного забезпечення, так і моделювання його логічної структури, що дозволяє комплексно підійти до створення сучасного інтерфейсного інструменту з реальним прикладним значенням.

### 1.2 Моделювання предметної області

Візуальне проєктування інтерфейсних елементів, зокрема кнопок, є важливою складовою сучасної веб-розробки, адже саме через UI користувач взаємодіє із системою. Інтерфейс повинен бути не лише функціональним, а й естетично привабливим, доступним для сприйняття, адаптивним до різних пристроїв. У центрі цієї взаємодії перебувають елементи управління, головними серед яких є кнопки [1].

Згідно з принципами user-centered design, будь-який інтерфейс має бути орієнтованим на кінцевого користувача [2]. Саме тому дедалі більшої популярності набувають веб-додатки, що дозволяють створювати UI-елементи без програмування. Серед прикладів подібних сервісів — CSS Button Generator від Colorzilla, HTML Code Generator або Get Waves — інструменти, які дозволяють користувачеві візуально будувати окремі елементи інтерфейсу без потреби писати код вручну.

Однак більшість таких рішень є обмеженими в налаштуваннях або мають застарілий інтерфейс. Наприклад, CSS Button Generator дозволяє змінювати лише базові параметри — колір, тінь, градієнт — без live-прев'ю або анімацій при кліку. Крім того, значна частина з них не адаптована під сучасні фреймворки або підходи типу responsive UI.

З іншого боку, веб-розробники часто вдаються до фреймворків типу Bootstrap або Tailwind CSS, які надають готові стилі кнопок. Але ці рішення вимагають навичок роботи з класами, знань HTML/CSS, а не кожен користувач, особливо новачок, здатен швидко зрозуміти структуру таких стилів [3].

Саме тому виникає потреба в універсальному, інтуїтивному інструменті, що дозволяє:

- у візуальному форматі обирати параметри кнопки;
- бачити live-прев'ю;

- одразу копіювати готовий HTML/CSS-код;
- і навіть моделювати складні анімаційні ефекти — glow, ripple, loading.

У межах предметної області такого застосунку можна виділити кілька ключових концепцій:

- Користувач — особа, що взаємодіє із застосунком, обирає параметри кнопки.
- UI-компонент — кнопка, яка має властивості: колір фону, шрифт, розмір тексту, радіус скруглення, іконка, тінь, hover-ефект, ripple-ефект, світіння тощо.
- Генератор коду — внутрішній модуль, що формує HTML/CSS-код на основі введених параметрів.
- Симулятор збереження — умовна база даних (users, buttons), яка дозволяє уявити збереження шаблонів та подальше їхнє використання.

Ці компоненти тісно взаємодіють між собою. Наприклад, зміна кольору або вибір шрифту одразу впливає на візуальне прев'ю, а також оновлює HTML-код. Також кожен параметр може мати обмеження — наприклад, вибір іконки може комбінуватися лише з певними стилями (відповідно до best practices UI).

Предметну область можна розглядати і в контексті модульності: окремо функціонують блоки параметрів, блок прев'ю, блок генерації коду, блок експорту. Така побудова дає змогу легко масштабувати систему — наприклад, додати збереження у PDF або SVG, експортувати в React-компоненти, або підключити авторизацію користувачів.

Враховуючи те, що користувачі часто мають мінімальний технічний бекграунд, застосунок має підтримувати принципи гнучкого UX, мінімізуючи кількість обов'язкових дій, і водночас забезпечуючи повний контроль над параметрами компонента [4].

Таким чином, предметна область UI Button Builder охоплює як візуальні, так і логічні складові: інтерактивну взаємодію, параметризацію елементів, динамічне оновлення, генерацію коду та умовне збереження. Її структура логічно лягає в основу побудови UML-діаграм та наступного етапу — реалізації інтерфейсу.

### 1.3 Діаграма прецедентів

Діаграма прецедентів (англ. Use Case Diagram) — це графічне представлення функціональних вимог до програмної системи, яке використовується для моделювання взаємодії зовнішніх акторів із системою через набір цільових сценаріїв. Цей тип діаграм є одним з ключових елементів мови моделювання UML (Unified Modeling Language) [5]. Його застосування дозволяє формалізувати потреби користувача, визначити функціональне охоплення системи та встановити її межі ще до початку розробки.

У процесі створення веб-застосунку UI Button Builder діаграма прецедентів стала важливим етапом, що дав змогу відобразити усі ключові сценарії взаємодії користувача із системою. На відміну від інших типів UML-діаграм, прецедентна модель фокусується на функціях з точки зору кінцевого користувача, а не на внутрішній логіці реалізації, що робить її надзвичайно зручною для початкового етапу проєктування [6].

Система UI Button Builder є одноакторною, тобто передбачає взаємодію лише одного суб’єкта — користувача, який взаємодіє з інтерфейсом через набір доступних дій. Серед таких дій — введення тексту кнопки, вибір шрифту, зміна кольору фону, вибір градієнта та його кольорів, налаштування іконки, додавання тіней та світіння, вибір анімації при наведенні, натискання на прев’ю-кнопку, отримання ripple або loading ефекту, а також копіювання згенерованого HTML/CSS-коду.

Усі ці функції зображено на діаграмі прецедентів, що представлена нижче на рисунку 1.1. Діаграма побудована у стандартній UML-нотації. Вона містить одного актора та декілька прецедентів, що демонструють усі можливі сценарії взаємодії користувача із системою.


Рисунок 1.1 — Діаграма прецедентів

Згідно з діаграмою, користувач може послідовно налаштовувати всі параметри кнопки. Кожен прецедент є самостійною функцією, яка оновлює стан

прев’ю та код у реальному часі. Введення тексту кнопки є стартовою дією. Після цього користувач змінює шрифт, кольори, градієнт, додає іконку, анімацію, ефекти. Натискаючи на прев’ю, користувач бачить ripple-ефект або "Loading...", що імітує поведінку кнопки в реальних умовах. У кінці здійснюється копіювання коду до буфера обміну — тобто результат перенаправляється за межі системи.

У цілому, діаграма прецедентів забезпечує високий рівень абстракції, чітко відображає функціональність і дозволяє перейти до побудови діаграми діяльності на наступному етапі. Її використання є рекомендованим стандартом у сфері моделювання інтерактивних систем [7].

### 1.4 Діаграма діяльності

Діаграма діяльності (англ. Activity Diagram) — це один із базових засобів динамічного моделювання систем у мові UML, який дозволяє відобразити алгоритм виконання дій, що відбуваються в рамках конкретного процесу або сценарію. На відміну від діаграми прецедентів, що фокусується на зовнішніх взаємодіях, діаграма діяльності детально розкриває внутрішню логіку функціонування системи, її послідовність кроків, альтернативні гілки, паралельні процеси та точки прийняття рішень [8].

Діаграма діяльності тісно пов’язана з поняттям потоків управління, що дозволяє ефективно моделювати інтерфейсні взаємодії, особливо у веб-застосунках, де важливе значення має послідовність зміни станів залежно від дій користувача. У сучасному проєктуванні UI/UX такі діаграми дозволяють заздалегідь виявити логічні неузгодженості, визначити, які компоненти системи відповідають за зміну станів, і як саме реалізується логіка переходів між ними [9].

У рамках даного проєкту було розроблено діаграму діяльності для застосунку UI Button Builder. Вона відображає базовий сценарій взаємодії користувача з системою, починаючи з моменту завантаження сторінки і до моменту копіювання результату роботи.

Умовно весь процес можна поділити на кілька логічно пов’язаних блоків. На початку користувач відкриває веб-застосунок, після чого починається послідовне налаштування параметрів кнопки: вводиться текст, обирається шрифт, змінюється розмір, колір тексту та фон. Далі користувач може додати градієнт (у випадку, якщо обрано тип градієнта), вказати два кольори для плавного переходу, налаштувати іконку, а також вибрати ефекти — тінь, світіння, анімацію при наведенні.

Після завершення візуальної конфігурації користувач натискає на прев’ю-кнопку. У цей момент активується блок клікової анімації: ripple-ефект або

loading-текст із затримкою. У результаті дії оновлюється прев’ю, а згенерований HTML/CSS код відображається у текстовому полі. Завершальним етапом є натискання кнопки «Copy Code», після чого код кнопки копіюється у буфер обміну.

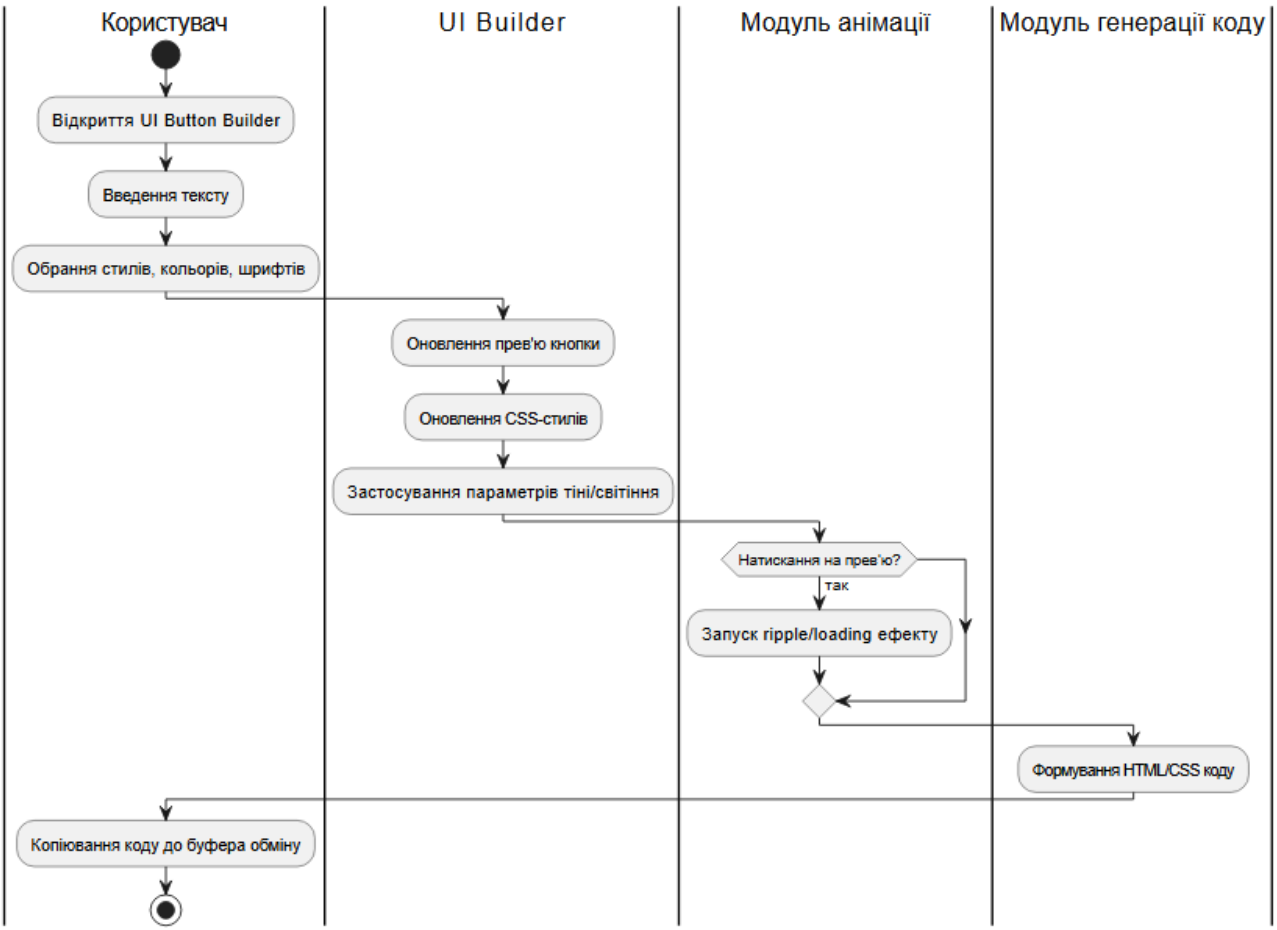



Рисунок 1.2 – Діаграма діяльності

Як видно з опису, система працює за принципом реактивного оновлення інтерфейсу, де кожна зміна відображається в режимі реального часу. Це дозволяє не лише покращити зручність взаємодії, а й скоротити час, необхідний для створення бажаного інтерфейсного елементу. Подібний підхід активно застосовується у сучасних фреймворках (React, Vue, Angular) і вважається одним із найефективніших з точки зору UX [10].

Діаграма діяльності, таким чином, слугує важливим артефактом, що формалізує внутрішню логіку застосунку та дозволяє повніше зрозуміти поведінку користувача й реакцію системи. Її побудова є невід’ємною частиною моделювання інтерфейсно-орієнтованих веб-застосунків.

### 1.5 Діаграма послідовності

Діаграма послідовності (англ. Sequence Diagram) — один із ключових інструментів динамічного моделювання в UML, який відображає послідовність взаємодії між об’єктами у часі. Її основною метою є показати, які саме компоненти системи беруть участь у виконанні конкретного сценарію, які повідомлення передаються між ними, і в якій черговості ці повідомлення відбуваються [11].

На відміну від діаграми діяльності, яка акцентує увагу на логічному потоці дій, діаграма послідовності дозволяє візуалізувати взаємодію між окремими об’єктами, модулями або підсистемами, причому з точним урахуванням часової послідовності викликів. Цей підхід особливо корисний у випадках, коли система має поділену відповідальність між різними модулями — інтерфейсом, візуальним рендерингом, анімацією, генератором коду тощо.

У контексті веб-додатку UI Button Builder діаграма послідовності описує типовий сценарій взаємодії користувача з системою: від налаштування параметрів до натискання на прев’ю та копіювання коду. Для цього в межах системи логічно виділено чотири основні об’єкти: Користувач, UI Controller, Code Generator, Animation Engine.

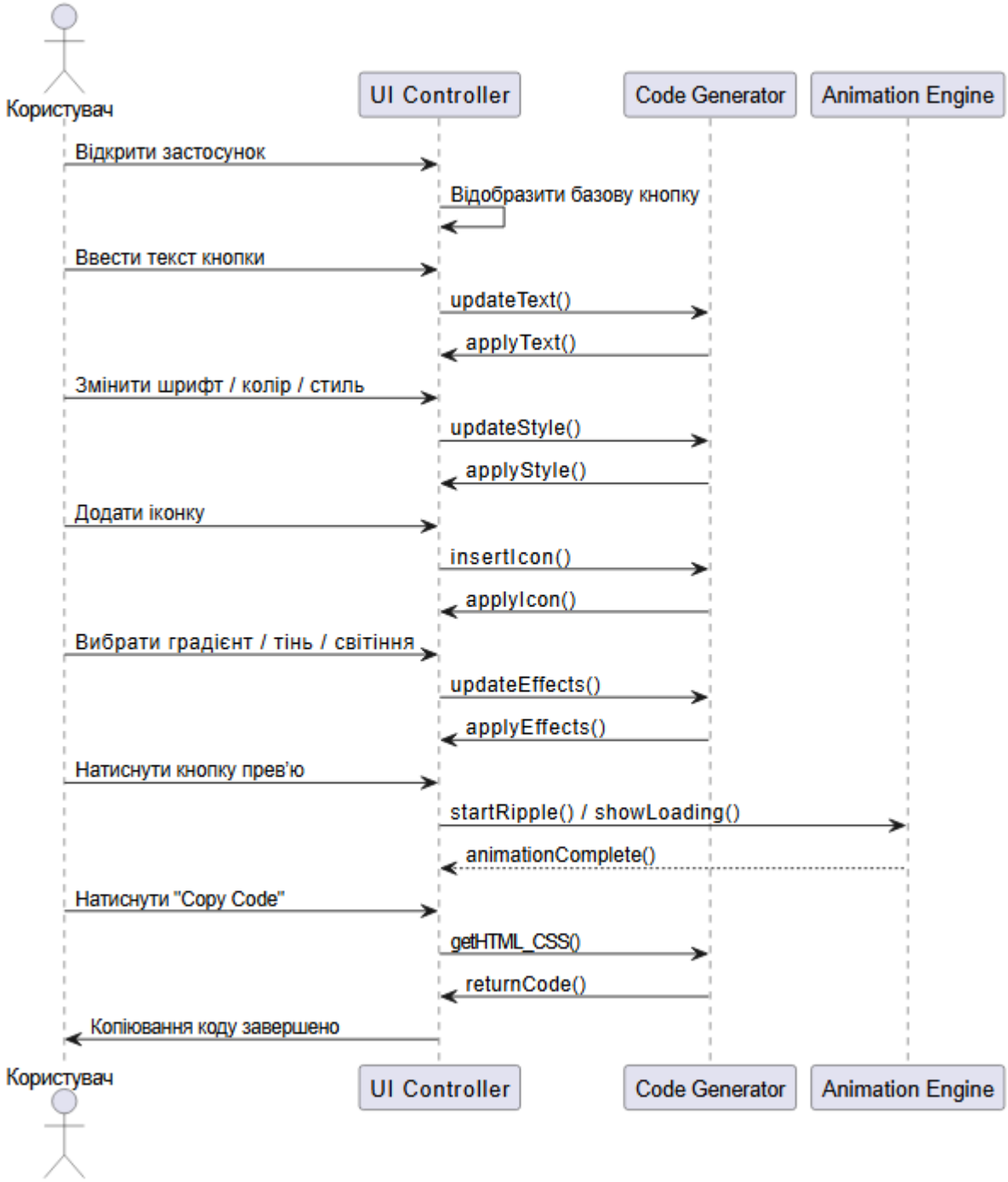



Рисунок 1.3 – Діаграма послідовності

Сценарій починається з того, що користувач ініціює відкриття застосунку. UI Controller (тобто інтерфейсна частина системи) відображає базову кнопку та параметри для налаштування. Далі відбувається серія подій, що повторюються:

користувач вводить текст або обирає стиль, а система фіксує зміни та надсилає запит до UI Controller.

UI Controller обробляє введені дані і викликає функції генератора CSS-коду, який, у свою чергу, оновлює стилі кнопки в реальному часі. Наприклад, зміна шрифту чи кольору викликає метод updateButtonStyle(), який оновлює DOM-елемент та CSS-властивості.

Якщо користувач активує ripple-ефект або натискає кнопку в прев'ю, UI Controller передає запит до Animation Engine, який запускає відповідну анімацію. Після завершення ефекту інтерфейс повертає візуальний стан до початкового.

На завершальному етапі користувач натискає на кнопку «Copy Code», яка ініціює метод copyToClipboard(). Цей метод викликається з боку UI Controller, однак використовує внутрішню функцію navigator.clipboard.writeText() — стандартний API браузера, що здійснює копіювання сформованого HTML-коду з модуля Code Generator.

Таким чином, кожен компонент системи — UI Controller, Code Generator, Animation Engine — відіграє окрему роль у реалізації повного сценарію взаємодії, і їхні виклики чітко синхронізуються між собою.

Діаграма послідовності дозволяє виявити потенційні точки розриву логіки, дублювання функцій або надмірну залежність між модулями. Її побудова є обов'язковою практикою у професійній розробці вебзастосунків, де складність логіки часто розподілена між фронтендом, анімаційними модулями та генераторами вмісту [12].

# 2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

## 2.1 Загальні відомості про ER-діаграму

ER-моделювання (Entity–Relationship modeling) є фундаментальною методологією для проєктування структури баз даних, яка отримала широке застосування в інформаційних системах ще з 1976 року, коли її вперше запропонував Пітер Чен [14]. Модель «сутність–зв’язок» дозволяє формалізувати у вигляді графічної схеми логіку предметної області: визначити, які об’єкти в ній існують, якими властивостями вони володіють, та як саме пов’язані між собою.

На відміну від інших моделей даних, зокрема ієрархічної чи мережевої, ER-модель є незалежною від конкретної реалізації і зручна для побудови логічної структури, яку пізніше можна перенести у фізичну модель реляційної бази даних [15]. Її основними конструктивними елементами є сутності (entities), атрибути (attributes) та зв’язки (relationships). Сутність описує абстрактний або реальний об’єкт системи (наприклад, користувач, замовлення, кнопка), атрибути — це характеристики сутності, а зв’язки визначають взаємозалежності між об’єктами.

ER-діаграма є візуальним відображенням цих конструкцій. Сутності зображуються у вигляді прямокутників, атрибути — як овали, а зв’язки — як ромби з лініями, що з’єднують відповідні сутності. Зв’язки можуть мати типи кардинальності (один до одного, один до багатьох, багато до багатьох), що дозволяє точно описати логіку взаємодії між елементами [16].

Побудова ER-діаграми зазвичай передбачає кілька етапів: аналіз предметної області та виділення основних сутностей; визначення атрибутів кожної сутності; встановлення зв’язків і їхньої кратності; виявлення первинних та зовнішніх ключів (на етапі нормалізації); трансформація в схему таблиць для подальшої реалізації в СУБД [17].

ER-модель є універсальним інструментом не лише в академічному середовищі, а й у реальних проєктах. Вона використовується при проєктуванні

ERP-систем, e-commerce платформ, CRM-сервісів, медичних систем, освітніх платформ тощо. Саме завдяки своїй зрозумілості, наочності та відповідності логіці реляційної моделі даних ER-діаграма залишається основним засобом проєктування баз навіть у великих комерційних системах [18].

У сфері веб-розробки, зокрема у SPA (Single Page Applications) та low-code-платформах, ER-діаграми дозволяють спроєктувати внутрішні структури даних, що передаються через API або зберігаються локально. Навіть у випадках, коли система не використовує повноцінну БД (наприклад, застосунок працює лише в браузері), ER-модель допомагає краще зрозуміти логіку збереження даних і потенційні можливості масштабування [19].

У контексті проєкту UI Button Builder — веб-додатку, що дозволяє створювати кастомізовані HTML/CSS кнопки — ER-моделювання виконує роль формальної документації до логіки структурування параметрів кнопки. Моделювання сутностей, таких як User і UIButton, дозволяє описати, як могли б зберігатися шаблони, як пов’язані вони з користувачами, і що система потенційно може масштабуватись для реалізації особистих кабінетів, історії, каталогів шаблонів тощо. Навіть за відсутності фізичної реалізації БД, ER-діаграма дає змогу продемонструвати структуроване бачення системи та забезпечує відповідність вимогам до повноцінного інформаційного забезпечення програмного продукту [20].

Таким чином, побудова ER-діаграми — це не лише академічна формальність, а цілком прикладна практика, яка дозволяє чітко змоделювати предметну область, задати логіку даних і підготувати систему до можливої майбутньої інтеграції з реляційними СУБД. Саме тому ER-моделі є обов’язковим компонентом технічної документації в рамках будь-якої серйозної розробки програмного забезпечення [21].

### 2.2 Побудова ER-діаграми

У межах проєктування інформаційного забезпечення веб-застосунку UI Button Builder, було сформовано модель логічної структури даних у вигляді ER-діаграми. Це дозволяє не лише формалізувати об’єкти предметної області, а й змоделювати їхню потенційну реалізацію у вигляді реляційної бази даних на основі SQLite — легкої, вбудованої СУБД, що часто використовується у невеликих або автономних веб/десктоп-застосунках.

У системі було виокремлено дві основні сутності:

- User — репрезентує користувача, який взаємодіє із застосунком. Має унікальний ідентифікатор, логін, email та пароль (збережений у вигляді хешу). У майбутньому можливе доповнення цієї сутності ролями або правами доступу.
- UIButton — уособлює збережений шаблон кнопки, створений користувачем. Вона містить набір параметрів: текст, шрифт, розмір, колір тексту та фону, градієнт, іконку, ефекти наведення, тіні та світіння, а також дату створення. Усі ці поля можуть бути використані для динамічного відтворення кнопки в інтерфейсі або при генерації коду.

Між сутностями User та UIButton встановлено зв’язок типу "один до багатьох" — один користувач може створити кілька кнопок, кожна з яких пов’язана з ним через зовнішній ключ user_id.

На рисунку 2.1 представлено ER-діаграму системи. Вона відображає сутності, їхні атрибути та зв’язки між ними відповідно до стандартів побудови логічних моделей даних.

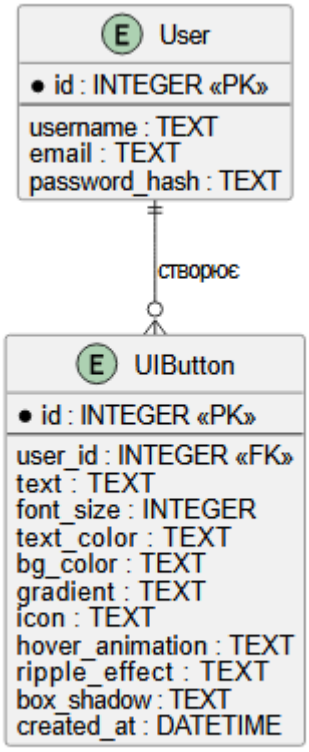



Рисунок 2.1 ER-діаграма структури даних

У контексті реалізації цієї структури на SQLite, таблиця users виступає головною (master-таблицею), а buttons — підлеглою (detail-таблицею), зв’язаною через зовнішній ключ. Такий підхід дозволяє легко виконувати запити типу SELECT * FROM buttons WHERE user_id = ..., що дає змогу відображати лише ті кнопки, які були створені конкретним користувачем.

Дана модель є гнучкою, простою у реалізації та зручною для розширення. При необхідності можуть бути додані нові атрибути або навіть нові сутності (наприклад, "категорії кнопок", "збережені кольорові палітри" або "глобальні шаблони").

Таким чином, побудована ER-діаграма не лише формалізує логіку збереження даних у системі, а й демонструє можливість її реального втілення в

### 2.3 Вибір та обґрунтування СУБД

Для реалізації інформаційного забезпечення веб-застосунку UI Button Builder було обрано систему управління базами даних SQLite. Це рішення базується на особливостях самої системи, її архітектурі та обсягах даних, які потенційно обробляються у рамках проєкту.

SQLite — це легка, вбудована, серверless-СУБД, яка зберігає всі дані у звичайному файлі на диску. На відміну від повноцінних клієнт-серверних рішень (таких як PostgreSQL або MySQL), SQLite не потребує розгортання окремого серверного процесу, має мінімальний об’єм та чудово підходить для застосунків з невеликим навантаженням або локальною архітектурою.

Однією з головних причин вибору SQLite є її простота інтеграції у вебзастосунки, які не передбачають великої кількості одночасних підключень або складної системи доступу. У випадку UI Button Builder, кожен користувач створює власні шаблони кнопок у межах окремого сеансу — тобто обробка даних відбувається на рівні локальної сесії, без необхідності постійної синхронізації або транзакцій з декількома користувачами одночасно.

Крім того, SQLite чудово працює в середовищах, де не передбачено повноцінної серверної логіки. Наприклад, при розробці desktop-версії конструктора кнопок або автономного Electron-додатку, вона може зберігати дані прямо в локальному файлі, не створюючи додаткових залежностей і конфігурацій. Це суттєво полегшує розгортання системи та знижує вимоги до обслуговування.

Важливою перевагою також є низький поріг входу — навіть розробники без глибоких знань у сфері баз даних можуть легко створювати таблиці, виконувати запити та інтегрувати SQLite у свої проєкти. А за рахунок підтримки стандартного синтаксису SQL, ця СУБД може слугувати відмінним середовищем для початкової реалізації, з можливістю подальшої міграції на інші рішення у разі масштабування.

З технічної точки зору, SQLite підтримує всі ключові можливості, необхідні для реалізації логіки UI Button Builder:

- створення таблиць із первинними та зовнішніми ключами;
- виконання простих запитів до окремих сутностей (users, buttons);
- фільтрацію, сортування, об’єднання;
- збереження даних у структурованому вигляді з можливістю бекапу.

Таким чином, обрання SQLite як основної СУБД для даного застосунку є оптимальним варіантом, який відповідає масштабу, архітектурі та функціональному призначенню системи. У майбутньому, за необхідності розширення функціоналу (наприклад, додавання багатокористувацького доступу, спільної роботи або онлайн-синхронізації), структура моделі даних дозволяє безболісно мігрувати до більш потужних рішень — таких як PostgreSQL або MySQL.

### 2.4 Створення бази даних

Після побудови ER-діаграми та вибору системи управління базами даних необхідним етапом є формування фізичної структури БД. У межах даного проєкту, який реалізовано з використанням SQLite, база даних створюється у вигляді локального .db-файлу, що містить усі таблиці, індекси та збережені записи. Формування цієї структури виконується шляхом написання SQL-запитів типу CREATE TABLE.

Згідно з побудованою логічною моделлю, у системі передбачено дві основні таблиці: users — для зберігання облікових даних користувачів, та buttons — для збереження створених кнопок. Обидві таблиці реалізуються із застосуванням первинних ключів (PRIMARY KEY), а також зовнішнього ключа (FOREIGN KEY), що формує зв’язок між користувачем і його кнопками. Такий підхід дозволяє зберігати цілісність даних і забезпечує коректну фільтрацію шаблонів відповідно до конкретного користувача.

Для зберігання облікової інформації в таблиці users передбачено поля: id (унікальний ідентифікатор користувача), username (логін), email та password_hash. Поле id є автоінкрементним цілим числом і виступає первинним ключем. Поля username та email можуть бути проіндексовані при потребі для забезпечення швидкого пошуку.

У таблиці buttons зберігаються всі параметри кнопки: текст, шрифт, розмір, кольори, іконка, анімації, ефекти тощо. Зв’язок із користувачем реалізовано через поле user_id, яке є зовнішнім ключем, що посилається на users(id). Це дозволяє зберігати для кожного користувача довільну кількість кнопок, прив’язаних до його облікового запису.

Нижче наведено SQL-інструкції для створення зазначених таблиць у СУБД SQLite:

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
    username TEXT NOT NULL,
    email TEXT NOT NULL,
    password_hash TEXT NOT NULL
);

CREATE TABLE buttons (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    text TEXT NOT NULL,
    font_size INTEGER,
    text_color TEXT,
    bg_color TEXT,
    gradient TEXT,
    icon TEXT,
    hover_animation TEXT,
    ripple_effect TEXT,
    box_shadow TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Ключовою особливістю структури є використання вбудованого часу створення (created_at), який автоматично фіксується за допомогою DEFAULT CURRENT_TIMESTAMP. Це дозволяє у подальшому реалізувати сортування шаблонів за датою, реалізувати фільтрацію або відображення історії створення.

Також важливою є директива ON DELETE CASCADE у зовнішньому ключі: у випадку видалення користувача автоматично видаляються всі кнопки, що були з ним пов’язані. Це забезпечує підтримку цілісності на рівні БД і виключає необхідність додаткової перевірки вручну.

Створена структура легко масштабується: до неї можна додати додаткові поля (наприклад, теги, статус публікації, обкладинки) або створити нові таблиці (наприклад, для збереження загальних шаблонів або улюблених елементів).

Таким чином, розроблена структура бази даних повністю відповідає логіці системи UI Button Builder, забезпечує надійне зберігання даних, можливість фільтрації, прив’язки до користувача та майбутнього розширення функціоналу.

### 2.5 Додавання користувачів до бази даних

Після створення структури таблиць у СУБД наступним етапом є заповнення бази початковими даними. У межах даної системи первинне наповнення виконується через SQL-запити типу INSERT INTO, які дозволяють додати нових користувачів до таблиці users, а також пов’язані з ними шаблони кнопок у таблицю buttons.

З огляду на безпекові вимоги, під час додавання користувачів доцільно зберігати не сам пароль, а його хеш, що генерується на стороні бекенду з використанням криптографічних алгоритмів, таких як SHA-256 або bcrypt. У прикладі нижче використовується вже готове хешоване значення:

```
INSERT INTO users (username, email, password_hash)
VALUES ('button_god', 'godmode@ui.com', '5f4dcc3b5aa765d61d8327deb882cf99');
```

У цьому прикладі користувач button_god буде збережений з поштою godmode@ui.com та хешем (що відповідає паролю password, як класика для демонстрацій). Ідентифікатор id буде присвоєно автоматично завдяки AUTOINCREMENT.

Після цього можна додати пов’язані кнопки, що належать даному користувачу. Наприклад:

```
INSERT INTO buttons (user_id, text, font_size, text_color, bg_color, gradient, icon, hover_animation, ripple_effect, box_shadow)
VALUES (
    1,
    'Натисни мене!',
    18,
    '#ffffff',
    '#007BFF',
    'linear-gradient(to right, #ff8a00, #e52e71)',
```

```
    ' ',
    'hover-glow',
    'ripple',
    '0px 10px 15px rgba(0,0,0,0.3)'
);
```

У цьому запиті передається набір параметрів, що відповідає структурі кнопки: текст, шрифт, кольори, анімації та візуальні ефекти. Значення user_id = 1 означає, що дана кнопка буде пов’язана з користувачем button_god.

За аналогією можна створити набір шаблонів для демонстрації або тестування, а також реалізувати інтерфейс, який буде автоматично додавати користувачів та кнопки з форми реєстрації.

У разі використання SQLite в браузерному середовищі або у форматі локального застосунку (наприклад, Electron), ці дані можуть зберігатися в окремому .db-файлі, який оновлюється при кожному новому записі. У більш складних реалізаціях можливе також використання API для відправлення SQL-команд із фронтенду або бекенду.

Таким чином, додавання користувачів до бази є важливим етапом перевірки структури даних, тестування зв’язків між таблицями та підготовки системи до інтеграції з інтерфейсом авторизації чи збереження шаблонів у реальному часі.

# 3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

## 3.1 Вибір інструментарію для розробки програмного забезпечення

Для реалізації веб-застосунку UI Button Builder було обрано класичний технологічний стек фронтенд-розробки, який поєднує в собі гнучкість, простоту, швидкість розробки та широкі можливості кастомізації інтерфейсів. До складу обраного інструментарію входять: HTML5, CSS3, JavaScript, бібліотека Bootstrap, а також різноманітні допоміжні вебтехнології, що дозволяють реалізовувати інтерактивні ефекти, анімації та роботу з буфером обміну.

Базовою мовою розмітки виступає HTML5 — стандартна мова створення структури вебсторінок. У контексті даного проєкту HTML використовується для формування інтерфейсних елементів: полів введення, селекторів, слайдерів, чекбоксів, кнопок та області прев'ю. Семантична структура HTML дозволяє легко масштабувати інтерфейс та забезпечує сумісність з усіма сучасними браузерами.

Для стилізації елементів інтерфейсу застосовується CSS3. Саме за допомогою каскадних таблиць стилів реалізовано адаптивну верстку, зміни шрифтів, кольорів, тіней, радіусів скруглення, плавні переходи та візуальні ефекти. Окрему увагу приділено підтримці псевдокласів :hover, :focus, а також сучасних засобів роботи з анімацією та фільтрами. За допомогою CSS користувач отримує можливість динамічно формувати вигляд кнопки без використання складних фреймворків.

Щоб уникнути надмірного дублювання коду та прискорити розробку, було використано Bootstrap — популярну CSS-бібліотеку, яка забезпечує готові компоненти та класи для реалізації адаптивного макету. Bootstrap значно спрощує створення сіток, розташування блоків, стилізацію форм, кнопок і модалів. Його використання дозволяє отримати сучасний, акуратний інтерфейс із мобільною адаптацією без необхідності писати додаткові медіазапити вручну.

Ключовим інструментом для забезпечення логіки взаємодії є JavaScript. У застосунку він відповідає за обробку подій (наприклад, натискання кнопки, зміна кольору, введення тексту), оновлення прев'ю-кнопки в реальному часі, генерацію HTML/CSS коду, вставку ripple-ефектів та копіювання результату в буфер обміну через API navigator.clipboard. Для зручності код розбито на функціональні блоки: робота з параметрами, ефекти, анімації, копіювання.

Також використано набір допоміжних інструментів — шрифти Google Fonts, emoji-іконки, CSS-анімаційні бібліотеки для hover-ефектів, а також локальні CSS-класи для ripple-анімації, стилізованих тіней, glow-ефектів.

Усі вибрані інструменти мають відкриту ліцензію, що дозволяє використовувати їх без обмежень у навчальних та комерційних проєктах. Крім того, вони добре документовані, мають активну спільноту та підтримуються всіма популярними браузерами.

Таким чином, комбінація HTML5, CSS3, Bootstrap та JavaScript забезпечує швидку, стабільну та зручну реалізацію конструктора UI-кнопок з високим рівнем кастомізації, інтерактивності та адаптивності.

### 3.2 Реалізація внутрішньої логіки застосунку

На відміну від типових сайтів з фіксованим вмістом, застосунок UI Button Builder функціонує як динамічний генератор інтерфейсних елементів, у якому всі взаємодії відбуваються у режимі реального часу. Користувач не просто переглядає сторінку — він безпосередньо формує та трансформує HTML/CSS-код через інтерфейсні контролери, що взаємодіють із візуальним прев'ю та кодовим блоком. Вся логіка побудована без використання сторонніх фреймворків — тільки JavaScript, DOM і здоровий глузд.

Основний принцип роботи застосунку полягає в тому, що кожен інтерфейсний параметр (колір, розмір шрифту, текст кнопки тощо) пов'язаний із подією (event listener). Коли користувач змінює значення в полі (наприклад, обирає новий колір або редагує текст), запускається відповідна функція, яка оновлює:

1. властивості DOM-елемента кнопки в прев'ю;
2. рядок HTML/CSS у виводі коду для копіювання.

Це означає, що застосунок працює як двонаправлений трансформатор: дії користувача → зміна параметрів → оновлення UI → оновлення коду. Наприклад, при виборі тіней система одразу застосовує CSS-властивість box-shadow до прев'ю, а також додає відповідний рядок у CSS-код. Те саме відбувається з градієнтами, іконками, шрифтами тощо.

Для генерації стилів застосовується об'єктна модель параметрів, у якій кожен параметр записується як окреме поле (buttonConfig.text, buttonConfig.fontSize, buttonConfig.shadow тощо). При кожній зміні викликається глобальна функція updateButton(), яка перебудовує і HTML, і CSS на основі поточного стану цього об'єкта.

Окремо реалізовано модулі:

- анімованої реакції на клік (ripple-ефект), що вбудовується у кнопку через псевдоелементи та стилі;
- loading-стану, який тимчасово замінює текст кнопки на "Loading..." з анімацією;
- копіювання коду через API navigator.clipboard.writeText(), що дозволяє миттєво передати результат користувачу.

Система не перезавантажує сторінку і не використовує бекенд: усе обчислюється на клієнтській стороні. Це дозволяє уникнути затримок і робить інтерфейс максимально інтерактивним.

Крім того, реалізовано обробку ситуацій за замовчуванням: якщо користувач нічого не вибрав (наприклад, не ввів текст або не обрав колір), кнопка відображається з базовими параметрами, а код генерується відповідно до "чистого" шаблону.

Окремий блок логіки відповідає за перевірку комбінацій: наприклад, якщо користувач вмикає градієнт, то фон-кольори підміняються відповідними gradient-рядками, і звичайне background-color ігнорується. Це дозволяє уникнути CSS-конфліктів і забезпечує однозначну поведінку при копіюванні.

На момент генерації коду система динамічно формує:

- <button>-елемент із усіма класами, стилями та іконками;
- тег <style>, в якому прописані кастомні правила для конкретної кнопки;
- блок, що доступний для копіювання користувачем одним кліком.

Таким чином, застосунок реалізовано як фронтендова low-code система, яка перетворює вхідні параметри на реальний візуальний результат та готовий код, не вимагаючи від користувача знань програмування. Уся логіка підтримується локально в браузері, що дозволяє використовувати UI Button Builder навіть без доступу до інтернету або бекенд-сервера.

# 4 ВПРОВАДЖЕННЯ СИСТЕМИ

## 4.1 Тестування системи

Тестування програмного забезпечення є критично важливим етапом розробки, що дозволяє перевірити коректність реалізації функціоналу, стабільність поведінки інтерфейсу, відповідність очікуваним результатам та готовність системи до практичного використання. У межах розробки UI Button Builder проводилось функціональне тестування, яке охоплювало всі основні елементи застосунку: від динамічного оновлення прев'ю до генерації коду та взаємодії з буфером обміну.

Для перевірки роботи були складені тест-кейси, що моделювали різні сценарії взаємодії з інтерфейсом, зокрема: зміна тексту кнопки, вибір кольору фону, додавання градієнту, встановлення іконки, зміна розміру шрифту, додавання тіні, копіювання коду, перевірка ripple-ефекту, ввімкнення loading-анімації та інші.

Кожен тест містив вхідні дані, очікувану поведінку, фактичний результат та статус виконання. Усі тести проводилися вручну у середовищі браузера Google Chrome. Таблиця 3.1 містить узагальнені результати тестування.

Таблиця 3.1 – Результати тестування функціональності UI Button Builder

| № | Назва тесту | Вхідні дані | Очікуваний результат | Фактичний результат | Статус |
|---|---|---|---|---|---|
| 1 | Зміна тексту кнопки | Введено "Натисни мене" | Кнопка оновилась, у коді з’явився новий текст | Відповідає очікуванню | Пройдено |
| 2 | Зміна кольору фону | Вибрано #FF5733 | Змінився колір кнопки, код містить background-color: #FF5733 | Відповідає | Пройдено |
| 3 | Додавання градієнту | Вибрано linear-gradient(to right, #ff8a00, #e52e71) | У коді з’явився правильний градієнт | Успішно відображено | Пройдено |
| 4 | Додавання іконки | Введено | Іконка з’явилась перед текстом, у коді відображена | Відповідає | Пройдено |
| 5 | Зміна кольору тексту | Вибрано #FFFFFF | Текст став білим, у стилях є color: #FFFFFF | Повністю збігається | Пройдено |
| 6 | Зміна розміру шрифту | Встановлено 20px | Текст кнопки збільшено, у коді font-size: 20px | Зміни застосовані | Пройдено |
| 7 | Додавання тіні | Вибрано "0px 10px 15px rgba(0,0,0,0.3)" | У стилях кнопки є box-shadow, ефект візуально присутній | Застосовано | Пройдено |

| | | | | | |
|---|---|---|---|---|---|
| 8 | Застосування ripple-ефекту | Натиснуто на кнопку | Після натискання з’являється ripple-анімація | Ефект працює стабільно | Пройдено |
| 9 | Loading-анімація при кліку | Увімкнено режим "loading" | Текст змінюється на "Loading...", додається анімація | Працює, кнопка змінюється | Пройдено |
| 10 | Копіювання HTML/CSS коду | Натиснуто "Copy" | Код скопійовано до буфера обміну, без помилок | Код вставляється у блокнот | Пройдено |
| 11 | Збереження стану між змінами | Послідовна зміна кольору, шрифту, іконки | Кнопка реагує на всі параметри, зміни не конфліктують | Система зберігає комбінації | Пройдено |
| 12 | Очистка стилів при скиданні | Встановлено reset | Стилі повертаються до дефолтних, код очищено | Функція працює, все скидається | Пройдено |
| 13 | Порожні поля | Нічого не введено | Кнопка зберігає базовий стиль, код створюється з дефолтами | Помилок немає | Пройдено |
| 14 | Несумісні комбінації (фон + градієнт) | Вибрано і фон, і градієнт одночасно | Пріоритет має градієнт, фон ігнорується | Поведінка передбачувана | Пройдено |
| 15 | Робота в мобільному перегляді | Зменшено розмір вікна браузера | Інтерфейс адаптується, елементи не виходять за межі екрану | Адаптація успішна | Пройдено |

У результаті тестування система UI Button Builder продемонструвала стабільну роботу при взаємодії з основними параметрами та динамічному оновленні інтерфейсу. Усі ключові функції — зміна властивостей, застосування ефектів, генерація коду, інтерактивність — працюють коректно. Система реагує швидко, візуальні зміни відображаються миттєво, помилки або зависання під час тестування не зафіксовані.

Тестування підтвердило готовність застосунку до практичного використання в режимі одного користувача. Подальший розвиток може включати автоматизоване тестування, інтеграцію unit-тестів та підтримку логування подій для діагностики.

### 4.2 Апаратні та технічні засоби

Для забезпечення стабільної роботи веб-застосунку UI Button Builder необхідно враховувати як апаратні, так і програмні вимоги до середовища виконання. Оскільки проєкт реалізовано як frontend-рішення, яке повністю працює у браузері без серверної частини, основні вимоги зосереджуються на потужності клієнтського пристрою, сумісності з браузерами та підтримці сучасних вебтехнологій.

Умовно технічні засоби можна поділити на три категорії: апаратне забезпечення користувача, програмне середовище, а також параметри браузера та середовища відображення. У таблицях нижче наведено рекомендовані та мінімально допустимі характеристики, необхідні для повноцінної роботи застосунку.

Таблиця 4.2 – Апаратне забезпечення клієнтського пристрою

| Параметр | Мінімальні вимоги | Рекомендовані вимоги |
|---|---|---|
| Процесор (CPU) | 2 ядра, 1.5 GHz | 4 ядра, 2.5+ GHz |
| Оперативна пам’ять (RAM) | 2 ГБ | 4–8 ГБ |
| Відеоадаптер | Вбудований (Intel HD) | Будь-який сучасний GPU |
| Місце на диску | 100 МБ (для кешу браузера) | 500+ МБ (для роботи з іншими вкладками) |
| Дисплей | 1024×768 | 1920×1080 або вище |

Таблиця 4.3 – Програмне забезпечення середовища виконання

| Компонент | Мінімальна версія | Рекомендована версія |
|---|---|---|
| Операційна система | Windows 7 / macOS 10 | Windows 10+, macOS 12+, Linux Ubuntu 20+ |
| Браузер | Google Chrome 90+ | Chrome 100+, Firefox 110+, Edge 100+, Safari 14+ |
| JavaScript | Увімкнений | Увімкнений (ECMAScript 6+) |
| Плагіни | Не обов’язково | Clipboard API, Web Animations API |
| Інтернет-з’єднання | Необов’язкове | Для хостингової версії — 512 Кб/с+ |

Таблиця 4.4 – Середовище відображення та UX-параметри

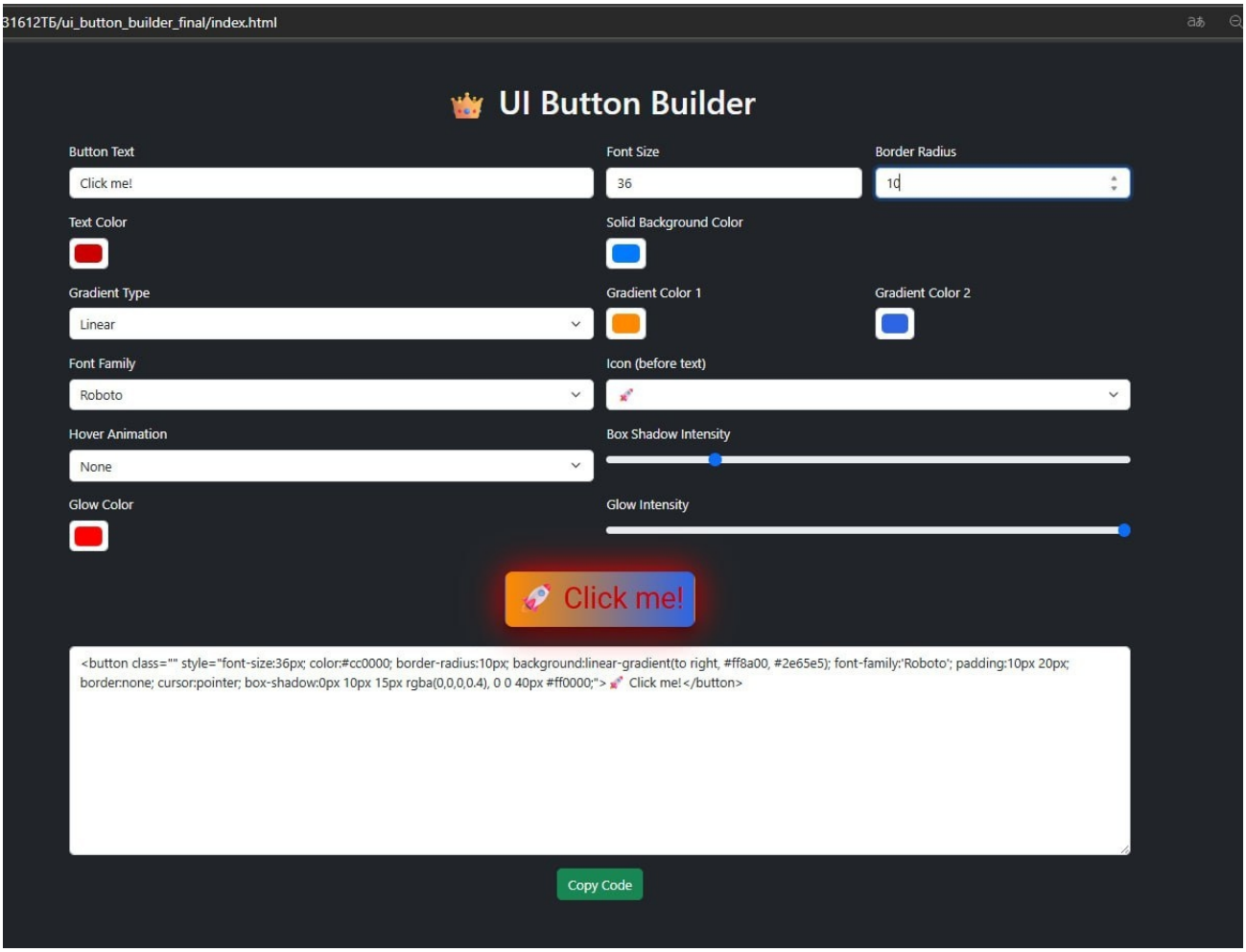
| Параметр | Значення |
|---|---|
| Підтримка роздільної здатності | Від 1024×768 до 4K |
| Тип інтерфейсу | Адаптивний, mobile-first |
| Анімації | CSS-анімації, псевдоелементи, JavaScript ripple |
| Підтримка темної теми | Часткова (на рівні браузера) |
| Підтримка буфера обміну | Через navigator.clipboard.writeText() |
| Прев’ю результату | Live-рендерінг через innerHTML + style |
| Генерація коду | Динамічна, вбудована у DOM |

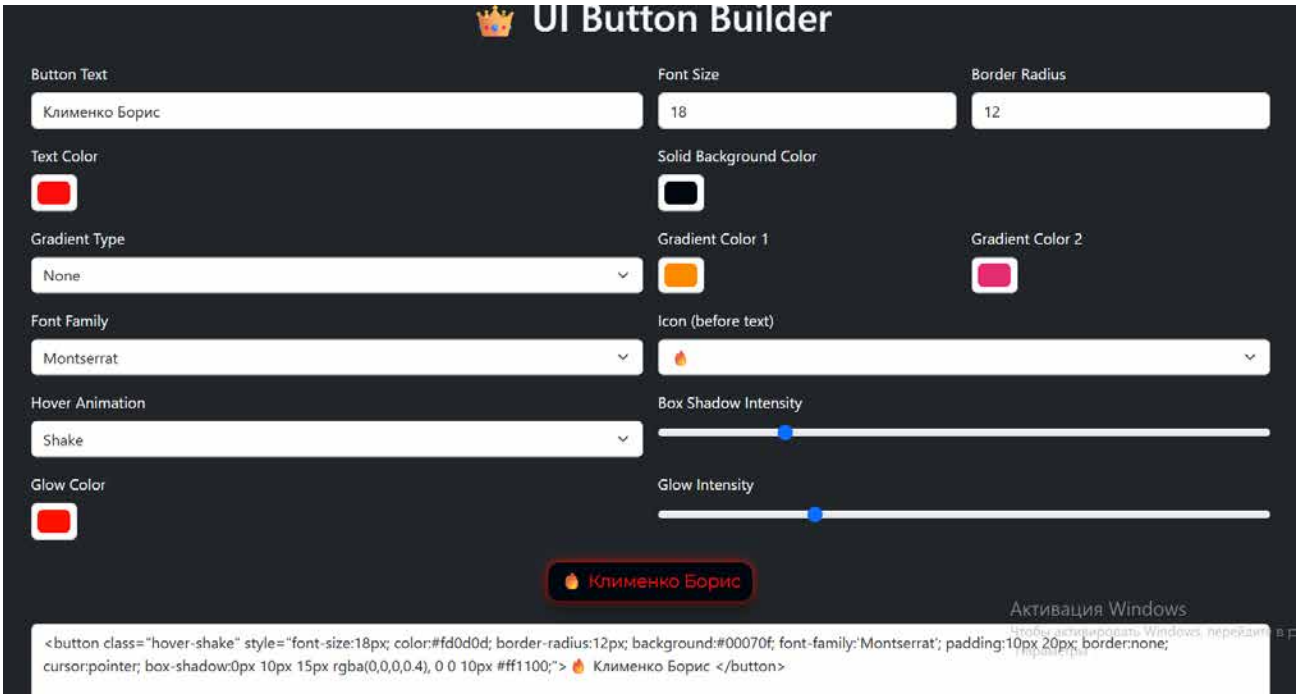
Таким чином, застосунок не вимагає встановлення додаткового ПЗ, не навантажує систему і повністю функціонує у звичайному сучасному браузері. Завдяки мінімальним вимогам до обчислювальних ресурсів, UI Button Builder може використовуватися як на сучасних комп’ютерах, так і на ноутбуках середнього рівня, планшетах та навіть мобільних пристроях з достатнім розміром екрану.

Проєкт також адаптований до широкого спектру середовищ: він може бути розгорнутий як на локальному хостингу (через файл index.html), так і на публічних платформах (GitHub Pages, Netlify, Replit). Немає необхідності в

бекенді або базі даних — усі зміни зберігаються в рамках поточної сесії, що додатково спрощує інтеграцію та обслуговування.

## 4.3 Опис роботи програми



Веб-застосунок UI Button Builder реалізовано у вигляді односторінкового інтерфейсу, що дозволяє користувачеві створювати кастомні HTML/CSS кнопки у реальному часі. Увесь процес побудови елементу управління відбувається через інтуїтивно зрозумілий візуальний інтерфейс, що складається з панелі параметрів, області попереднього перегляду (preview) та блоку згенерованого коду.

Після завантаження сторінки користувач бачить стартову конфігурацію: кнопка з базовим текстом, шрифтом і фоном. Зліва або зверху (залежно від адаптації) розташована панель керування параметрами. Вона містить:

- текстове поле для введення напису на кнопці;
- селектор шрифтів і слайдер розміру шрифту;
- колірні піктограми для кольору тексту і фону;
- опцію ввімкнення градієнту і вибору двох кольорів;
- поле для вставки emoji-іконки або спеціального символу;
- вибір hover-анімації з ефектами типу scale, glow, wobble тощо;
- тумблери для ripple-ефекту та loading-анімації;
- контроль тіні та glow-ефекту з можливістю вибору кольору і інтенсивності.

Кожна взаємодія користувача одразу тригерить відповідні функції JavaScript, які оновлюють вигляд кнопки в області preview та блок зі згенерованим кодом. Це дозволяє не лише візуально оцінити результат, а й отримати готовий HTML/CSS-код для вставки у свій проєкт.

При кожній зміні параметрів викликається внутрішня функція updateButton(), яка перебудовує:

- стилі DOM-елемента в реальному часі;
- CSS-рядки у коді (включно з fallback'ами);
- HTML-структуру кнопки з врахуванням іконок та тексту.

У нижній частині інтерфейсу розташовано блок коду, де динамічно оновлюється результат. Користувач може натиснути кнопку «Copy», яка активує navigator.clipboard.writeText() і дозволяє швидко скопіювати згенерований HTML/CSS до буфера обміну.

Крім базових функцій, застосунок реалізує:

- ripple-ефект — при кліку на прев'ю, з'являється розтікання хвилі від центра натискання;
- loading-ефект — замінює текст кнопки на «Loading…» з анімацією на кілька секунд;
- збереження стану кнопки в змінній об'єкта, яка може бути розширена під localStorage.

У разі не введення певних параметрів застосунок підставляє значення за замовчуванням. Наприклад, якщо не обрано колір фону — використовується стандартний Bootstrap-синій. Це дозволяє працювати навіть при частковому налаштуванні.

Також передбачено відображення конфліктів стилів: наприклад, при активації градієнту фон-кольори ігноруються, про що користувач отримує візуальне підтвердження — наприклад, дизактивація поля або змінена підсвітка.

Загалом логіка побудована за принципом low-code UI builder, де кожна дія має візуальний і технічний відгук. Користувач створює кнопку повністю через інтерфейс — без написання коду — а на виході отримує легкий для інтеграції результат. Усі дії здійснюються локально в браузері без звернень до сервера, що гарантує швидкість, стабільність і повну автономність.

# ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено повноцінний веб-застосунок UI Button Builder, призначений для візуального проєктування HTML/CSS кнопок з широкими можливостями кастомізації. Система реалізована як односторінковий застосунок, що функціонує повністю на стороні клієнта і не потребує встановлення додаткових компонентів чи серверного забезпечення.

У межах реалізації проєкту виконано такі ключові завдання:

- проведено аналіз предметної області та визначено основні функціональні вимоги до застосунку;
- побудовано діаграми прецедентів, діяльності та послідовності для формалізації логіки системи;
- змодельовано інформаційне забезпечення у вигляді ER-діаграми з реалізацією логічної структури бази даних у середовищі SQLite;
- реалізовано інтерфейс, що підтримує динамічне налаштування параметрів кнопки — тексту, шрифтів, кольорів, градієнтів, іконок, тіней, анімацій, ripple-ефектів тощо;
- забезпечено генерацію HTML/CSS-коду в реальному часі з можливістю копіювання результату до буфера обміну;
- виконано тестування функціональності, що підтвердило стабільну роботу всіх компонентів системи.

Розроблений застосунок відзначається високою інтерактивністю, простотою у використанні та гнучкістю в налаштуванні інтерфейсних елементів. Завдяки використанню тільки фронтенд-технологій (HTML, CSS, JavaScript, Bootstrap), його можливо легко масштабувати, інтегрувати у більші вебсистеми або розгорнути як окремий desktop-додаток через Electron.

Подальший розвиток проєкту може передбачати:

- додавання реєстрації користувачів та збереження шаблонів у базі даних;
- експорт кнопок у форматах SVG або React-компонентів;
- реалізацію колективної бібліотеки кнопок з оцінками та тегами;
- підтримку локалізації, темної теми та drag-and-drop UI.

Таким чином, UI Button Builder — це не просто інструмент для створення кнопок, а база для побудови повноцінної low-code платформи інтерфейсного дизайну. Проєкт повністю відповідає поставленим вимогам, демонструє високу якість реалізації та може бути використаний як у навчальних, так і в практичних цілях.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Krug, S. (2014). Don’t Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders.
2. ISO 9241-210:2010. Human-centred design for interactive systems.
3. W3C. Design Principles for Web UI Elements. Available at: https://www.w3.org/TR/wai-aria-practices-1.2/
4. Nielsen, J. (1993). Usability Engineering. Academic Press.
5. Буч, Г., Рамбо, Дж., Джекобсон, І. UML. Користувацький посібник. — К.: Діалектика, 2005. — 496 с.
6. Шаховська, Н. Б., Рощенко, А. М. Моделювання програмного забезпечення: навч. посібник. — Львів: Львівська політехніка, 2020. — 256 с.
7. Object Management Group. UML Superstructure Specification, v2.5 [Електронний ресурс]. — Режим доступу: https://www.omg.org/spec/UML
8. Месарович, М. Системний аналіз: принципи, методи, застосування. — Київ: Либідь, 2012. — 432 с.
9. Хацкевич, Л. М., Пащенко, В. П. UML у проєктуванні інформаційних систем: навчальний посібник. — Харків: ХНУРЕ, 2020. — 188 с.
10. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.
11. Фаулер, М. UML. Основи та практики. — Київ: Діалектика, 2021. — 384 с.
12. Літвінов, І. М. Розробка інтерфейсних систем з використанням UML: навч. посіб. — Харків: ХНУРЕ, 2020. — 212 с.
13. Чумаченко, І. В. Проєктування баз даних: навч. посіб. — Харків: ХНУРЕ, 2018. — 174 с.

14. Chen, P. The Entity-Relationship Model — Toward a Unified View of Data // ACM Transactions on Database Systems. — 1976. — Vol. 1, No. 1. — P. 9–36.
15. Сидоренко М. В. Проєктування баз даних: навч. посібник. — Київ: КНЕУ, 2020. — 214 с.
16. Halpin T. Information Modeling and Relational Databases. 2nd ed. — Morgan Kaufmann, 2010. — 976 p.
17. Мартін Дж. Управління базами даних. — 2001. — 992 с.
18. Coronel C., Morris S. Database Systems: Design, Implementation, and Management. — Boston: Cengage Learning, 2018. — 672 p.
19. Карпінський Б. Системи баз даних: моделювання та проєктування. — Львів: Видавництво ЛНУ імені І. Франка, 2021. — 168 с.
20. Harrington J. L. Relational Database Design and Implementation. — Morgan Kaufmann, 2016. — 712 p.
21. Teorey T. J., Lightstone S., Nadeau T. Database Modeling and Design: Logical Design. 5th ed. — Morgan Kaufmann, 2011. — 352 p.

# ДОДАТОК А
# ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>UI Button Builder</title>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link rel="stylesheet" href="css/style.css">
<link
href="https://fonts.googleapis.com/css2?family=Roboto&family=Montserrat&family
=Comic+Neue&family=Arial&display=swap" rel="stylesheet">
</head>
<body class="bg-dark text-light">
<div class="container py-5">
    <h1 class="text-center mb-4">  UI Button Builder</h1>
    <div class="row g-3">
        <div class="col-md-6">
            <label class="form-label">Button Text</label>
            <input type="text" id="btnText" class="form-control" value="Click me!">
        </div>
        <div class="col-md-3">
            <label class="form-label">Font Size</label>
            <input type="number" id="fontSize" class="form-control" value="16">
        </div>
```

```
            <div class="col-md-3">
                <label class="form-label">Border Radius</label>
                <input type="number" id="radius" class="form-control" value="8">
            </div>
            <div class="col-md-6">
                <label class="form-label">Text Color</label>
        <input type="color" id="textColor" class="form-control form-control-color" value="#ffffff">
            </div>
            <div class="col-md-6">
                <label class="form-label">Solid Background Color</label>
        <input type="color" id="solidBgColor" class="form-control form-control-color" value="#007BFF">
            </div>
            <div class="col-md-6">
                <label class="form-label">Gradient Type</label>
                <select id="gradientType" class="form-select">
                    <option value="">None</option>
                    <option value="linear">Linear</option>
                    <option value="radial">Radial</option>
                </select>
            </div>
            <div class="col-md-3">
                <label class="form-label">Gradient Color 1</label>
        <input type="color" id="gradientColor1" class="form-control form-control-color" value="#ff8a00">
            </div>
            <div class="col-md-3">
```

```
                <label class="form-label">Gradient Color 2</label>
            <input type="color" id="gradientColor2" class="form-control form-control-color" value="#e52e71">
            </div>
            <div class="col-md-6">
                <label class="form-label">Font Family</label>
                <select id="fontFamily" class="form-select">
                    <option value="Arial">Arial</option>
                    <option value="Roboto">Roboto</option>
                    <option value="Montserrat">Montserrat</option>
                    <option value="Comic Neue">Comic Neue</option>
                </select>
            </div>
            <div class="col-md-6">
                <label class="form-label">Icon (before text)</label>
                <select id="icon" class="form-select">
                    <option value="">None</option>
                    <option value=" "> </option>
                    <option value=" "> </option>
                    <option value="    ">    </option>
                    <option value=" "> </option>
                    <option value=" "> </option>
                    <option value="    ">    </option>
                    <option value="    ">    </option>
                    <option value=" "> </option>
                    <option value=" "> </option>
                    <option value=" "> </option>
```

```
                <option value=" "> </option>
            </select>
        </div>
        <div class="col-md-6">
            <label class="form-label">Hover Animation</label>
            <select id="hoverEffect" class="form-select">
                <option value="">None</option>
                <option value="hover-scale">Scale</option>
                <option value="hover-glow">Glow</option>
                <option value="hover-pulse">Pulse</option>
                <option value="hover-shake">Shake</option>
            </select>
        </div>
        <div class="col-md-6">
            <label class="form-label">Box Shadow Intensity</label>
    <input type="range" id="shadowIntensity" class="form-range" min="0" max="50" value="10">
        </div>
        <div class="col-md-6">
            <label class="form-label">Glow Color</label>
    <input type="color" id="glowColor" class="form-control form-control-color" value="#00ffff">
        </div>
        <div class="col-md-6">
            <label class="form-label">Glow Intensity</label>
    <input type="range" id="glowIntensity" class="form-range" min="0" max="40" value="10">
        </div>
```

```
            </div>
            <div class="my-4 text-center position-relative">
                <div id="previewWrap" class="position-relative d-inline-block">
        <button id="previewBtn" class="btn overflow-hidden position-relative">Click me!</button>
                </div>
            </div>
        <textarea id="codeOutput" class="form-control mb-3" rows="10" readonly></textarea>
            <div class="text-center">
        <button onclick="copyCode()" class="btn btn-success">Copy Code</button>
            </div>
        </div>
        <script src="js/script.js"></script>
        </body>
        </html>

        .preview-button::before,
        .preview-button::after {
            position: absolute;
            pointer-events: none;
            color: inherit;
            font-size: inherit;
        }

        .preview-button::before {
            content: var(--before-content);
```

```
    left: -1.5em;
    top: 50%;
    transform: translateY(-50%);
}

.preview-button::after {
    content: var(--after-content);
    right: -1.5em;
    top: 50%;
    transform: translateY(-50%);
}

.hover-scale:hover {
    transform: scale(1.1);
    transition: all 0.3s ease-in-out;
}
.hover-glow:hover {
    box-shadow: 0 0 20px currentColor;
    transition: all 0.3s ease-in-out;
}
.hover-pulse:hover {
    animation: pulse 1s infinite;
}
.hover-shake:hover {
    animation: shake 0.5s;
}

@keyframes pulse {
```

```
    0% { transform: scale(1); }
    50% { transform: scale(1.05); }
    100% { transform: scale(1); }
}

@keyframes shake {
    0% { transform: translateX(0); }
    25% { transform: translateX(-3px); }
    50% { transform: translateX(3px); }
    75% { transform: translateX(-3px); }
    100% { transform: translateX(0); }
}

.hover-scale:hover {
    transform: scale(1.1);
    transition: all 0.3s ease-in-out;
}
.hover-glow:hover {
    box-shadow: 0 0 20px currentColor;
    transition: all 0.3s ease-in-out;
}
.hover-pulse:hover {
    animation: pulse 1s infinite;
}
.hover-shake:hover {
    animation: shake 0.5s;
}
@keyframes pulse {
```

```
    0% { transform: scale(1); }
    50% { transform: scale(1.05); }
    100% { transform: scale(1); }
}
@keyframes shake {
    0% { transform: translateX(0); }
    25% { transform: translateX(-3px); }
    50% { transform: translateX(3px); }
    75% { transform: translateX(-3px); }
    100% { transform: translateX(0); }
}
.ripple-effect {
    position: absolute;
    width: 100%;
    height: 100%;
    animation: ripple 0.3s linear;
    border-radius: inherit;
    background: rgba(255,255,255,0.3);
    top: 0;
    left: 0;
    pointer-events: none;
}
@keyframes ripple {
    0% { opacity: 1; }
    100% { opacity: 0; }
}

.ripple-effect {
```

```
    position: absolute;
    width: 150px;
    height: 150px;
    background: rgba(255,255,255,0.4);
    border-radius: 50%;
    transform: scale(0);
    animation: ripple-grow 0.6s linear;
    pointer-events: none;
}

@keyframes ripple-grow {
    to {
        transform: scale(2.5);
        opacity: 0;
    }
}

const get = id => document.getElementById(id);

const elements = [
  "btnText", "fontSize", "textColor", "radius", "solidBgColor", "gradientType",
  "gradientColor1", "gradientColor2", "fontFamily", "icon", "hoverEffect",
  "shadowIntensity", "glowColor", "glowIntensity"
];

const previewBtn = get("previewBtn");
const codeOutput = get("codeOutput");
```

```
function updateButton() {
    const text = get("btnText").value;
    const size = get("fontSize").value + "px";
    const color = get("textColor").value;
    const rad = get("radius").value + "px";
    const solidBg = get("solidBgColor").value;
    const gradType = get("gradientType").value;
    const grad1 = get("gradientColor1").value;
    const grad2 = get("gradientColor2").value;
    const font = get("fontFamily").value;
    const icn = get("icon").value;
    const hover = get("hoverEffect").value;
    const shadow = get("shadowIntensity").value;
    const glowColor = get("glowColor").value;
    const glowIntensity = get("glowIntensity").value;

    const background = gradType
      ? `${gradType}-gradient(to right, ${grad1}, ${grad2})`
      : solidBg;

const boxShadow = `0px ${shadow}px ${shadow * 1.5}px rgba(0,0,0,0.4), 0
0 ${glowIntensity}px ${glowColor}`;

    previewBtn.className = `btn ${hover} overflow-hidden position-relative`;
    previewBtn.innerHTML = `${icn ? icn + ' ' : ''}${text}`;
    previewBtn.style.fontSize = size;
    previewBtn.style.color = color;
    previewBtn.style.borderRadius = rad;
```

```
        previewBtn.style.background = background;
        previewBtn.style.fontFamily = font;
        previewBtn.style.boxShadow = boxShadow;
        previewBtn.style.position = "relative";

    const html = `<button class="${hover}" style="font-size:${size}; color:${color}; border-radius:${rad}; background:${background}; font-family:'${font}'; padding:10px 20px; border:none; cursor:pointer; box-shadow:${boxShadow};">${icn ? icn + ' ' : ''}${text}</button>`;
        codeOutput.value = html;
    }

    function copyCode() {
        codeOutput.select();
        document.execCommand("copy");
    }

    previewBtn.addEventListener("click", function (e) {
        const ripple = document.createElement("span");
        ripple.className = "ripple-effect";
        ripple.style.left = e.offsetX + "px";
        ripple.style.top = e.offsetY + "px";
        previewBtn.appendChild(ripple);
        setTimeout(() => ripple.remove(), 600);
        const original = previewBtn.innerHTML;
        previewBtn.innerHTML = "Loading...";
        previewBtn.disabled = true;
        setTimeout(() => {
```

```
        previewBtn.innerHTML = original;
        previewBtn.disabled = false;
    }, 1200);
});

elements.forEach(id => get(id).addEventListener("input", updateButton));
updateButton();
```