МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет

«Харківський авіаційний інститут»

Кафедра комп'ютерних наук та інформаційних технологій

**КУРСОВИЙ ПРОЕКТ**

**(РОБОТА)**

з дисципліни

Проектування інформаційних систем

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва дисципліни)

на тему:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студента (ки) 4 курсу 336ст групи

галузі знань

12 «Інформаційні технології»

(шифр і назва)

спеціальності\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

122 «Комп'ютерні науки» »

(шифр і назва)

освітня програма «Комп’ютеризація

обробки інформації та управління»\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали студента)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Харків – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет

«Харківський авіаційний інститут»

Кафедра комп'ютерних наук та інформаційних технологій

“ЗАТВЕРДЖУЮ”

Завідувач кафедри

\_\_\_\_\_\_\_\_\_ Олег ФЕДОРОВИЧ

( підпис) (ім’я та прізвище)

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

Завдання до курсової роботи з курсу **«Проектно-орієнтоване управління створення комп’ютерних систем»**

Студенту \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Група\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Початкові дані:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата видачі: 10.02.2025

Термін захисту роботи: 03.05.2025

| Зміст курсової роботи: | % | Термін |
| --- | --- | --- |
| 1. Систематизація матеріалу накопиченого під час підготовки до написання роботи. | 16.6 | 12.02.2025 |
| 2. Написання реферату та вступу, теоретичної частини (основні розділи пояснювальної записки). | 16.6 | 15.03.2025 |
| 3. Проектування системи. Побудова діаграм. Розробка додатків. Написання коду. | 16.6 | 06.04.2025 |
| 4. Вибір методів тестування. Написання тесткейсів, їх реалізація. Тестування інформаційної системи. | 16.6 | 18.04.2025 |
| 5. Формулювання висновків. Дооформлення роботи. | 16.6 | 23.04.2025 |
| 6. Підготовка до захисту. Розробка презентації. | 16.6 | 24.04.2025 |

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 дата підпис прізвище

**ЗМІСТ**

[ВСТУП 5](#_heading=h.7g54r7j0f4jb)

[1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ЗАВДАННЯ 8](#_heading=h.izbg9n17qlel)

[1.1. Види подій та сертифікатів 10](#_heading=h.txyhvnj2vl1i)

[1.2. Проблема керування сертифікатами 13](#_heading=h.3kf8b0dkhcp3)

[1.3. Види веб-застосунків та їх різновиди 13](#_heading=h.tqp863ch06e7)

[2. ОГЛЯД ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА   
КОНКУРУЮЧИХ РІШЕНЬ 16](#_heading=h.6lsuy1ky30mx)

[2.1. Free Certificate Maker for Custom Certificates 16](#_heading=h.u60259dugbdi)

[2.2. Free Online Certificate Maker 17](#_heading=h.ec9hb3efeclr)

[2.3. Online Certificate Maker 18](#_heading=h.5b7pifbokhtd)

[3. ВИЯВЛЕННЯ ВИМОГ ДО СИСТЕМИ. ОПИС   
ЇЇ ФУНКЦІОНАЛЬНОСТІ 22](#_heading=h.goiyk2sp8v1a)

[3.1. Види вимог до програмного продукту 22](#_heading=h.4du0r487a8jd)

[3.2. Аналіз прецедентів 25](#_heading=h.qqwz96y8sz46)

[3.3. Опис функціональності 28](#_heading=h.tl6w0y2ymm75)

[4. АРХІТЕКТУРА СИСТЕМИ 31](#_heading=h.e1amfrisb4th)

[4.1. Поняття архітектури програмного продукту 31](#_heading=h.qqbkec9u7o0d)

[4.2. Архітектури веб-застосунків 32](#_heading=h.kfx9dre9jb6d)

[4.3. Опис структури створеного продукту. Опис класів. 35](#_heading=h.kigvrc462ydn)

[4.4. Опис бази даних 38](#_heading=h.us6p48jpbepp)

[4.5. Особливості розгортання програмного продукту 40](#_heading=h.ov9l6cu0qaoe)

[5. ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА 43](#_heading=h.6n5m4e94btqg)

[5.1. Поняття інтерфейсу 43](#_heading=h.j90e04xocf6m)

[5.2. Проєктування інтерфейсу системи 44](#_heading=h.llj8usqp5anr)

[5.3. Технологічні аспекти проєктування інтерфейсу 49](#_heading=h.fkjt4cy75olc)

[6. ОПИС ПРОЕКТНИХ РІШЕНЬ, ІНСТРУМЕНТІВ   
ТА ПІДХОДІВ ДО РОЗРОБКИ 51](#_heading=h.ygplj7cyom1t)

[6.1. Опис мови програмування 51](#_heading=h.6i1fekgps4f8)

[6.2. Використані технології та бібліотеки 52](#_heading=h.1j1rm2qly4if)

[6.3. Середовище розробки 59](#_heading=h.naawvo1db6hq)

[6.4. Методологія розробки 60](#_heading=h.gwjim25qr7bt)

[7. МЕТОДИ ТА РЕЗУЛЬТАТИ ТЕСТУВАННЯ 63](#_heading=h.6g46z3mzn9k7)

[7.1. Поняття про тестування, види тестів 63](#_heading=h.5o2zquhgotcw)

[7.2. Обрані види тестів та результати тестування 64](#_heading=h.e7l0vyfpwjzh)

[ВИСНОВКИ 67](#_heading=h.90t5varis9td)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 68](#_heading=h.n7ve33262kq)

[ДОДАТОК А. ВИХІДНІ КОДИ (ФРАГМЕНТ) 72](#_heading=h.wirio417dpv)

[ДОДАТОК Б. ЗНІМКИ ЕКРАНУ ГОТОВОГО ПРОДУКТУ 78](#_heading=h.sj4bq3x4wvk2)

# 

# ВСТУП

*Актуальність кваліфікаційної роботи*: у сучасному світі, коли онлайн-освіта, вебінари та онлайн-курси стають все більш популярними, потреба в зручному та швидкому способі генерування сертифікатів для учасників стає надзвичайно актуальною. Ручне створення сертифікатів для кожного учасника забирає значну кількість часу та може призвести до помилок. Веб-застосунок для генерації сертифікатів в наш час є необхідним інструментом для освітніх установ, конференцій та інших подій, що надають учасникам сертифікати про участь.

Розробка такого веб-застосунку не тільки спростила б процес генерації сертифікатів, але й дозволила б автоматизувати цей процес, зменшити ймовірність помилок та зберегти час організаторів подій. Крім того, веб-застосунок забезпечить зручний доступ до сертифікатів для кожного учасника, що є додатковою перевагою для користувачів.

Такий веб-застосунок також може забезпечити збереження даних про учасників та їхні сертифікати в електронному форматі, що дозволить легко відстежувати участь у різних подіях та швидко пред'являти сертифікати на запит. Це сприятиме покращенню процесів адміністрування та архівування даних, що є важливим аспектом в організації будь-яких освітніх заходів. Такий веб-застосунок може також враховувати різні дизайни сертифікатів, щоб задовольнити потреби різних організацій та подій.

З огляду на швидкий розвиток онлайн-освіти та необхідність ефективного керування даними про учасників подій, розробка подібного веб-застосунку є дійсно актуальною і перспективною темою. Вона не тільки відповідає потребам сучасного освітнього середовища, але й сприяє оптимізації процесів управління та забезпечує зручний доступ до важливої інформації для всіх зацікавлених сторін.

*Метою роботи* є створення ефективного інструменту для автоматизації процесу створення сертифікатів для різноманітних подій, семінарів, курсів чи конференцій. Цей застосунок дозволить користувачам швидко та легко створювати персоналізовані сертифікати з необхідною інформацією, такою як ім'я отримувача, назва події, дата та іншими деталями.

Головною метою розробки є спрощення процесу створення сертифікатів, щоб зменшити час та зусилля, які зазвичай витрачаються на цей процес. Крім того, застосунок може забезпечити професійний вигляд та стандартизацію сертифікатів, що видаватимуться на різних заходах, забезпечуючи однаковий стиль та якість.

Такий інструмент буде корисним для організаторів подій, тренерів, викладачів та будь-яких інших осіб, які регулярно видають сертифікати. Розробка такого веб-застосунку дозволить автоматизувати та оптимізувати цей процес, що призведе до збільшення продуктивності та якості в роботі з оформленням сертифікатів.

Для того щоб досягти мети, необхідно виконати ряд *завдань*:

* зробити детальний огляд основних об'єктів предметної області;
* виконати аналіз програмного забезпечення аналогічного призначення та провести порівняння за обраними критеріями;
* провести детальний аналіз вимог та визначити й візуалізувати на UML-діаграмі основні прецеденти;
* використовуючи спеціалізоване програмне забезпечення виконати проєктування інтерфейсу веб-застосунку та побудувати необхідні для успішної реалізації UML-діаграми;
* розробити програмний продукт та провести його тестування;
* підготувати скрипт для розгортання веб-застосунку на сервер;.
* описати проектні рішення, інструменти створення, використані технології

та підходи до розробки;

* розрахувати економічні показники створеного продукту.

*Об’єктом дослідження* в роботі є інструменти, процеси та технології створення веб-застосунків.

*Предметом дослідження* є розробка веб-застосунку для генерації сертифікатів з використанням технологій Node.js та Vue.js.

*Методи дослідження*: в процесі виконання кваліфікаційної роботи були використані наступні методи:

Системний аналіз – для деталізації і розчленування на окремі важливі складові частини, що надає можливості з аналізу, прогнозування, проектування прийняття рішень в складних системах різної природи на основі системної методології.

Аналіз і синтез – для визначення особливостей процесу візуального оформлення програмного продукту.

Порівняння – для зіставлення даних у динаміці., використовуються з метою встановлення логічних закономірностей які впливають на досліджувані об'єкти або явища, і пошуку переваг та вразливостей які можуть виявлятися під впливом факторів.

*За результатами дослідження* сформульовані вимоги до системи для створення сертифікатів та розроблено відповідний програмний продукт.

*Практичне значення результатів.* Одержані результати можуть бути використані в діяльності освітніх закладів, навчальних центрів та інших установ.

# 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ЗАВДАННЯ

Предметна область сама є самостійним об’єктом або може бути об’єктом у складі іншої предметної області.

Предметна область (ПрО) – це те, що аналізується з метою виділення специфічної множини понять (сутностей, об’єктів) і зв’язків між ними. На множині цих понять визначається простір проблем (problem space) і простір рішень (solution space).

Найбільше поширення серед методів аналізу ПрО одержав метод OOAS Шлеєра і Меллора [1], призначений для подання ПрО за допомогою таких моделей:

* інформаційна модель системи;
* модель станів об’єктів, що може будуватися для будь-якого з об’єктів

інформаційної моделі;

* модель процесів, що відображає процеси і дії, які відбуваються в системі при проходженні моделей станів через життєві цикли – одержання, породження і завершення подій у системі.

Згідно з цим методом ПрО аналізується в три етапи: інформаційне моделювання, моделювання станів, моделювання процесів. Як результат виконання цих процесів створюються, відповідно, вищезазначені три моделі.

Зв’язки об’єктів визначаються в процесі інформаційного моделювання, а поведінка – в процесі моделювання станів. Модель станів відображає динамічні стани об’єктів системи і їхню поведінку. На третьому етапі визначаються дії і процеси, що породжують події. Дії мають функціональну природу. Ціль моделювання процесів полягає в тому, щоб розділити процеси на дії, які разом визначають функціональний зміст системи.

Під інформаційною моделлю розуміється сукупність об’єктів (сутностей) ПрО, їхніх характеристик (атрибутів) і зв’язків між ними. Вона створюється за реляційним принципом: подання зв’язків між об’єктами і їхніми атрибутами у вигляді відношень.

Аналіз ПрО полягає у виявленні об’єктів, наданні їм унікальних імен, що відповідають важливим поняттям цієї предметної області. Об’єктами можуть бути:

* абстракції реально існуючих об’єктів ПрО;
* ролі як абстракції цілей або призначення людини в системі;
* взаємодії об’єктів, отримані шляхом встановлення зв’язків між ними і

частинами системи;

* специфікації для подання правил, критеріїв і обмежень на застосування

об’єктів у системі.

Таким чином, елементами інформаційної моделі можуть бути об’єкти, їх атрибути й ідентифікатори, а також зв’язки між об’єктами.

Між об’єктами предметної області можуть існувати семантичні зв’язки. Наприклад, у певному розумінні студент пов’язаний з професором, який викладає в його групі, а сертифікат - з курсом, за проходження якого він виданий. Зв’язок – це абстракція певної змістовної залежності між об’єктами. Як правило, зв’язки встановлюються між об’єктами одного або різних класів і характеризуються кількістю екземплярів об’єктів, що одночасно можуть брати участь у цих зв’язках.

Зв’язки між об’єктами класифікуються за множинністю. Відповідно до цієї класифікації виділяють три різновиди зв’язків:

* один до одного (1:1) – існує тоді, коли у зв’язку беруть участь по одному екземпляру від цих об’єктів (наприклад, курс викладається професором, професор веде один курс);
* один до багатьох (1:N), існує тоді, коли один екземпляр об’єкта деякого класу може бути зв’язаний одночасно більш ніж з одним екземпляром іншого або того самого класу (наприклад, за проходження курсу видаються сертифікати, а сертифікати свідчать про проходження курсу);
* багато до багатьох (M:N) – існує тоді, коли у зв’язку можуть брати участь по декілька екземплярів об’єктів з кожного класу, тобто один або більше екземплярів одного класу зв’язані з одним або декількома екземплярами іншого або того самого класу (наприклад, на курсі навчаються студенти, студенти проходять одночасно кілька курсів).

Модель станів призначена для відображення динамічної поведінки, зміни станів об’єктів інформаційної моделі і життєвого циклу поведінки об’єктів. Стан моделі залежить від ситуації, обумовленої правилами і лінією поведінки об’єкта. Подія – це інцидент, що змушує об’єкт переходити з одного стану в інший. Усі екземпляри одного класу мають однакову поведінку, яка визначається:

* станом, залежним від поточних значень окремих його атрибутів;
* станом, що змінюється внаслідок виконаних над об’єктами дій;
* станом ПрО, залежним від сукупності станів її об’єктів;
* деякими процесами і діями, що змінюють стан об’єкта протягом його життєвого циклу.

Побудова моделі станів починається з виділення в інформаційній моделі об’єктів, що мають динамічну поведінку (наприклад, зміна стану з часом), визначення часу створення екземпляра об’єкта або його видалення (наприклад, електрична лампочка перегоріла, закінчився її життєвий цикл).

Модель процесів базується на вимогах до поведінки майбутньої системи. Поведінка визначається діями процесів, які пов’язані зі змінами у моделі станів. Дія – це реакція на подію, що ініціює виконання певних функцій системи. Кожна дія входить до складу процесу і визначається в термінах цього процесу і архівних даних об’єктів. Тобто в даній моделі процес – це сукупність дій (операцій), а архів даних – це атрибути об’єктів інформаційної моделі. Для кожної дії, вказаної на моделі станів, утворюється діаграма процесу, на якій відображається виникнення подій при виконанні функцій системи.

Як джерело даних процесів можуть розглядатись:

* атрибути об’єктів, що продовжують існувати після завершення роботи;
* системи;
* системний годинник;
* таймер;
* дані про події, що відбуваються;
* повідомлення від зовнішніх об’єктів.

Послідовність виконуваних процесів утворює потік керування, а кожен процес утворює потік даних. Потоки даних зображаються діаграмами діяльностей [1].

## 1.1. Види подій та сертифікатів

Подія – це те, що відбувалося або відбулося, сталося; явище, факт суспільного або особистого життя. Сукупність пов'язаних між собою в якомусь відношенні явищ, фактів суспільного життя (звичайно важливих, значних), які становлять ніби щось єдине ціле [2]. Існує безліч різних видів подій, для яких можна видавати сертифікати:

* семінари;
* онлайн-курси;
* тренінги з підвищення кваліфікації.

Розглянемо кожен з них окремо.

Семінар – це одна з форм навчання, при якій учасники дискутують, обговорюють представлену проблему. Семінар прийнято відносити до навчальної форми у вищих навчальних закладах. Хоча, на даний момент, семінари стали поширеними також для особистісного навчання, навчання бізнесу та багатьох інших сфер життя людини.

Семінар як форма навчання з’явилася в Стародавній Греції і Стародавньому Римі. На семінарах студенти виступали зі своєю доповіддю, після чого обмінювалися думками, проводили дискусії і приходили до спільного консенсусу. Саме знаходження єдиного правильного рішення було основною метою і функцією семінарів. Форми проведення семінарів.

Всі семінари прийнято ділити на 3 категорії:

* навчальні;
* науково-практичні;
* бізнес-семінари.

Перший вид семінарів відноситься до навчальних занять в університеті.

Науково-практичні – збирають аудиторію науковців, які вже мають досвід і знання в обговорюваній темі і знайомляться з роботами своїх колег.

Бізнес-семінари – це обмін досвідом серед представників однієї спеціальності, удосконалення знань щодо нових методик роботи. Саме бізнес-семінари вважаються найкращим методом підвищення кваліфікації.

Семінари, які можуть організовуватися в рамках однієї компанії називаються корпоративними. Вони можуть бути відкритими – вільними для відвідування усіма зацікавленими учасниками. У будь-якому випадку, корпоративний або відкритий, семінар допомагає ознайомитися з новими тенденціями, методиками роботи і бути на крок попереду своїх конкурентів. В окрему категорію можна винести також вебінари.

Сучасні технології покликані полегшувати життя своїм користувачам. Тепер зовсім не обов’язково долати величезні відстані, щоб бути присутнім на тому чи іншому семінарі. Багато компаній і окремих спікерів організовують семінари в режимі «онлайн». Структура таких вебінарів не відрізняється від звичайних офлайн зустрічей: спікер представляє свою тему, в ході доповіді учасники вебінару можуть писати свої питання в коментарях. Виступаючий читає питання і відразу ж дає на них відповіді [3].

Онлайн-курс – це програма навчання, яка організована відповідно до навчального плану (зазвичай за модулями) та проходить у віртуальному просторі. Онлайн-курси можуть бути неформальними та зосередженими на одній навичці або формальними, що ведуть до отримання сертифікату чи ступеня.

Різні онлайн-курси мають різні функції: деякі курси можуть бути суворо прив’язані до навчальної програми, інші можуть дозволити вам взяти перерву та зробити перерву у класі, коли дозволяє ваш розклад. Деякі можуть мати «живі» дискусійні групи, заплановані для відеоконференцій у певний час, інші можуть просто записувати лекції та надавати вам доступ до їх відео в будь-який час.

Онлайн-курси для самостійного навчання – це онлайн-курси, на яких ви можете вивчати розділи, проходити тести, відвідувати лекції та виконувати будь-яку курсову роботу у зручний для вас час – немає фіксованих термінів для проходження програми або її частин. Це дає більше автономії щодо того, де і коли виконувати курсову роботу, але декому важко залишатися в курсі через відсутність структури.

У онлайн-курсів багато переваг. Однією з основних переваг є гнучкість: онлайн-курси доступні у будь-який час та в будь-якому місці. Онлайн-курси з самостійним вивченням, де навчання здійснюється самостійно і немає жодних тимчасових зобов’язань, крім того, що студент мотивований витратити, часто ідеально підходять для тих, хто має щільний графік або хто намагається поєднувати сім’ю або кілька робіт [4].

Тренінг – це перш за все, навчання, яке базується на досвіді людини, розвиває здібності особистості до пошуку нових знань, творчості, допомагає здійснити інтелектуальний прорив, формує новий світогляд.

Важливою особливістю цієї форми підвищення кваліфікації є те, що його учасники не лише отримують нову інформацію, але відпрацьовують одночасно нові навички, використовуючи їх на практиці.

Тренінги умовно поділяються на дві групи: одні мають на меті формування професійних, ділових навичок і умінь працівника та/або групи працівників – це бізнес-тренінги, а інші торкаються психологічних аспектів спілкування в колективі та у виробничій діяльності і формують навички поведінки – це психологічні або соціально-психологічні тренінги

Під час кожного тренінгу пропонується виконати ті чи інші вправи, орієнтовані на розвиток чи демонстрацію певних психологічних якостей або навичок. Для цього застосовуються рольові, ділові ігри та інші методи [5].

## 1.2. Проблема керування сертифікатами

Сертифікати відіграють критичну роль у перевірці автентичності, конфіденційності та цілісності даних під час їх передачі через мережу. Однак існує низка проблем, пов'язаних з керуванням сертифікатами, які можуть впливати на їх ефективність та надійність.

Перш за все, проблема керування сертифікатами пов'язана з технічними аспектами їх використання та розповсюдження. Це включає в себе:

* як сертифікати створюються та підписуються;
* яким чином сертифікати поширюються серед користувачів та систем;
* як швидко та ефективно можна відкликати сертифікати у випадку компрометації або зміни статусу.

Не менш важливим аспектом є безпека. Вона включає в себе:

* стійкість до атак;
* достатній рівень захисту конфіденційних даних.

Одна з ключових проблем полягає в керуванні життєвим циклом сертифікатів, включаючи їхнє вчасне оновлення, відкликання та видалення після закінчення строку дії.

Проблема керування сертифікатами є складною та багатогранною. Вона охоплює технічні, безпекові та організаційні аспекти, які потребують уваги для забезпечення ефективності та безпеки систем, що використовують цифрові сертифікати. Для подолання цих проблем необхідні комплексні підходи та постійне вдосконалення процесів керування сертифікатами.

## 1.3. Види веб-застосунків та їх різновиди

Веб-застосунок – це програмне забезпечення або програма, яку можна відкрити за допомогою будь-якого браузера. Зовнішній інтерфейс веб програми розробляється за допомогою таких мов програмування: HTML, CSS, Javascript, які підтримуються на будь-якому браузері (Opera, Chrome, Mozilla). У той час як для написання серверної частини (Back-end) може використовуватися будь-яка інша мова програмування або фреймворк, Python, PhP, Ruby, Java.

Основні переваги веб-застосунків:

* веб-застосунки можуть застосовуватися на будь-якій операційній системі (Linux, Mac, Windows), оскільки всі вони підтримують сучасні браузери;
* у зв’язку з тим, що у веб-застосунку використовується той самий код порівняно з desktop додатками їх набагато легше підтримувати;
* застосунок простіше програмувати оскільки він не передбачає значного використання ресурсів ПК (пам’яті, процесора відеокарти);
* на відміну від мобільних застосунків, для веб-застосунків не потрібно схвалення вендорів жодних платформ, щоб випустити свою програму;
* веб-застосунки – це економний варіант для будь-якого підприємства, оскільки веб-програми не вимагають підписки або покупки ліцензій, а можуть використовуватися як SaaS-сервіс, що значно дешевше.

Існує три види веб-додатків:

* Single Page Application (SPA).
* Multi Page Application (MPA).
* Progressive Web App (PWA).

Single Page Application (SPA) – односторінковий інтерактивний застосунок. Важливо, що він не тільки знаходиться на одній сторінці, а й, подібно до повноцінної програми, є інтерактивним. Так інформаційний веб-сайт може складатися з однієї сторінки, але, по суті, не бути SPA. В односторінковому веб-застосунку користувач, перемикаючись між вкладками, залишається на одній сторінці. Причому підвантажуються та оновлюються лише необхідні частини контенту, що грає на користь швидкості SPA.

Приклад односторінкової програми – Gmail. Варто звернути увагу – коли відбувається перемикання між списками повідомлень, адреса сторінки не змінюється. Це відмінна ознака SPA.

Основна мова створення SPA – JavaScript. Невеликий односторінковий застосунок можна створити за допомогою бібліотеки jQuery. Однак цей варіант не найоптимальніший для великих проектів. Найкраще використовувати фреймворки Vue, React або Angular.

MPA (Multi Page Application) – традиційні багатосторінкові веб-програми. Коли користувач взаємодіє з веб-сайтом, завантажуються нові HTTP-сторінки. Тому обмін даними відбувається повільніше, ніж у SPA. Особливо, якщо є проблеми з інтернет-з’єднанням або хостингом веб сайту. Приклади MPA – інтернет-магазини, такі як Rozetka та Amazon.

PWA (Progressive Web App) – прогресивна програма близька за своїми можливостями, функціями та якістю користувацького досвіду до нативних комп’ютерних та мобільних застосунків. Чіткого кордону між не-PWA та PWA застосунком немає. Але можна виділити низку характеристик.

Зокрема PWA має містити проксі-шар (Service Worker) та Web App маніфест. По суті, браузер виступає віртуальною машиною для запуску веб-додатків, подібно до того, як Windows запускає exe-файли, а Android – apk.

Service Worker – це проксі-шар між серверною та клієнтською частиною. Він знаходиться у браузері, через нього проходять усі запити. Таким чином, є два фронтенди шару – в одному прописується інтерфейс, в іншому – логіка. Це дозволяє виконувати повноцінні програми для Інтернету. Service Worker зазвичай пишеться на чистому JS [6].

**Висновки за розділом**

Отже, проаналізувавши предметну область завдання ми виділили основні поняття (сутності), такі як події, сертифікати, веб-застосунки, проаналізували їх характерні риси, різновиди, та взаємозв’язки між ними.

# 2. ОГЛЯД ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА КОНКУРУЮЧИХ РІШЕНЬ

Веб-застосунок, розробляється як новий продукт, і не має на меті конкурувати з іншими. У цьому розділі ми розглянемо кілька альтернативних програмних продуктів, які можуть конкурувати з нашим, проаналізуємо їхні переваги та недоліки, щоб визначити найкращий вибір для реалізації подібних сценаріїв. Також треба звернути увагу на те, яка функціональність існуючих рішень вплинула на проектні рішення під час розробки Генератора сертифікатів.

В цьому розділі ми розглянемо кожну з альтернатив уважно, щоб зробити обґрунтований вибір стосовно функціональності нашого продукту.

## 2.1. Free Certificate Maker for Custom Certificates

"Free Certificate Maker for Custom Certificates" [7] є потужним онлайн-інструментом для створення індивідуальних сертифікатів, що надає користувачам можливість легко та швидко створювати професійні та привабливі сертифікати для різних потреб.

Сайт пропонує широкий вибір попередньо розроблених шаблонів сертифікатів, які користувачі можуть налаштовувати відповідно до своїх вимог. Ці шаблони містять усі необхідні елементи для створення сертифікатів, що економить час і зусилля. Інструмент підтримує додавання брендових кольорів і шрифтів, що дозволяє створювати сертифікати, які відповідають фірмовому стилю компанії чи установи.

Користувачі можуть почати створення сертифіката, обравши один з доступних шаблонів. Після цього вони можуть налаштувати текст, кольори, графіку та інші елементи за допомогою зручного інтерфейсу. Інструмент підтримує інтеграцію з іншими сервісами, що дозволяє додавати до сертифікатів логотипи та інші графічні елементи.

Після завершення дизайну сертифіката, користувачі можуть завантажити його у форматі PDF з мітками для друку або як зображення для електронної розсилки. Також є можливість генерувати посилання для онлайн-поширення сертифікатів.

Цей сайт є ідеальним рішенням для тих, хто шукає зручний та ефективний спосіб створення сертифікатів без необхідності мати глибокі знання у графічному дизайні. Завдяки своєму простому інтерфейсу та багатому набору функцій, цей інструмент підходить для різноманітних користувачів, від освітян до бізнесменів (Рисунок 2.1 та Рисунок 2.2).

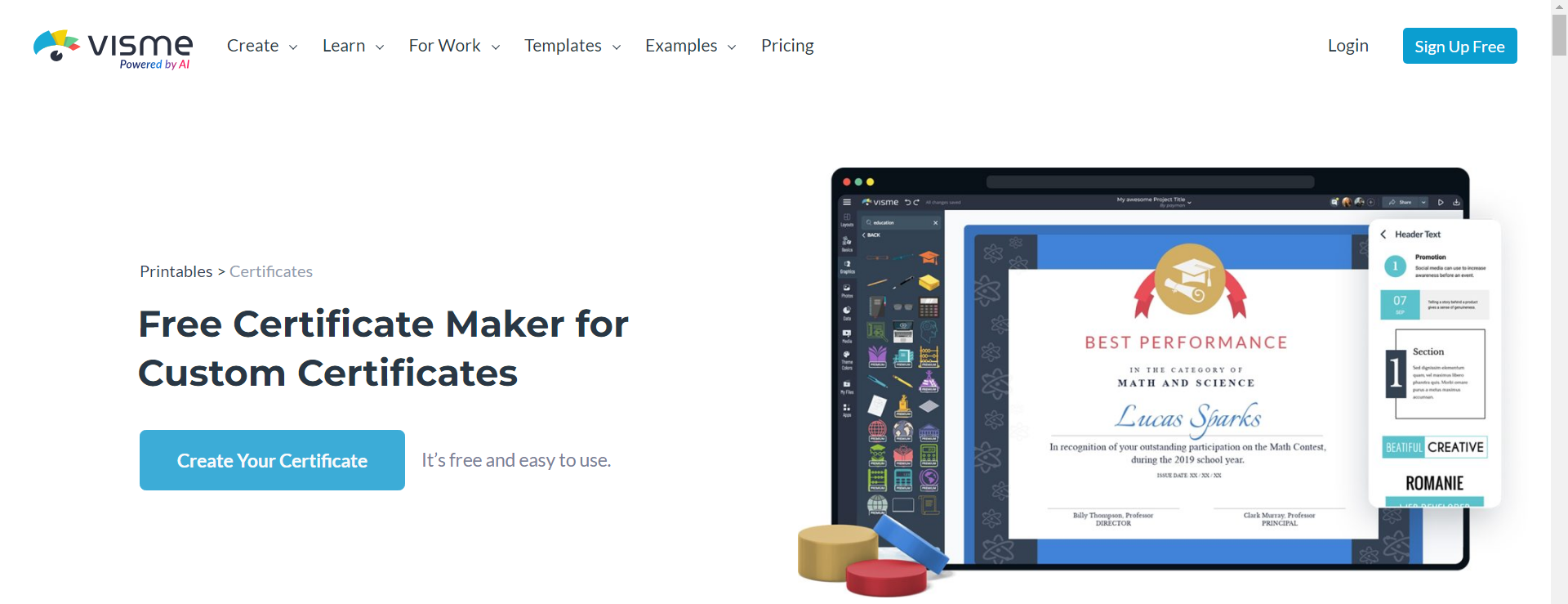


Рисунок 2.1 – Загальний вигляд інтерфейсу програми Free Certificate Maker for Custom Certificates

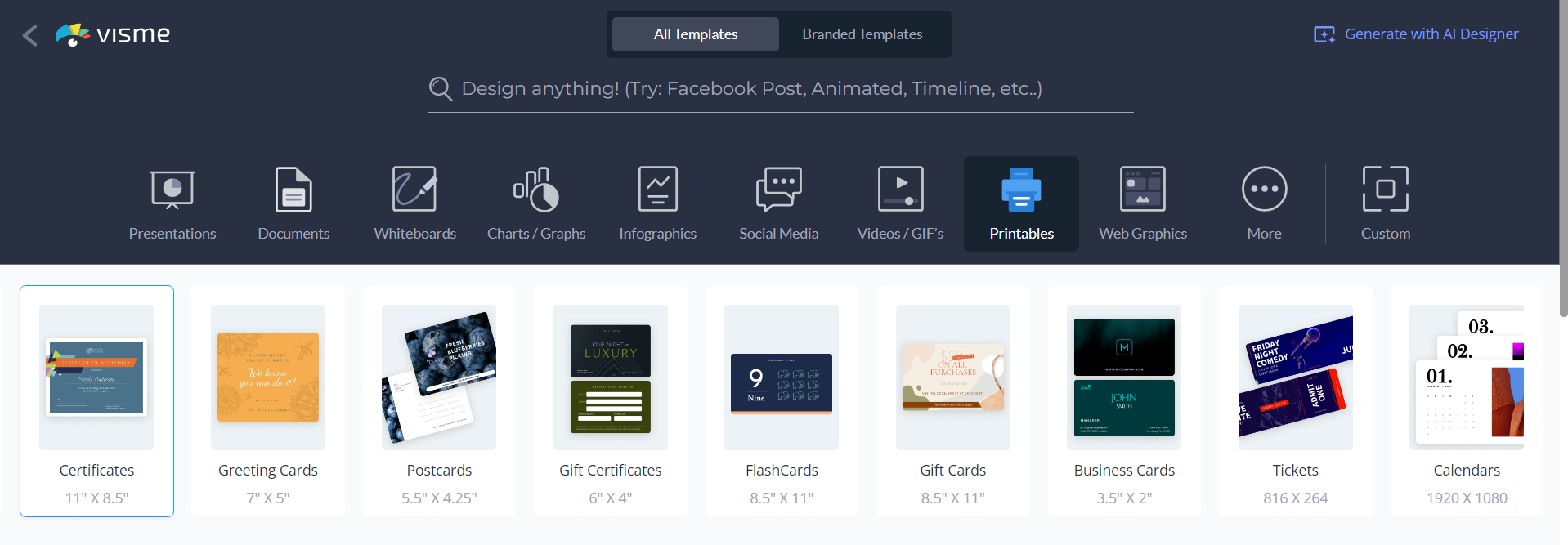


Рисунок 2.2 – Загальний вигляд створення сертифікату за допомогою програми Free Certificate Maker for Custom Certificates

## 2.2. Free Online Certificate Maker

Free Online Certificate Maker (Canva) [8] – це онлайн-інструмент для створення сертифікатів, який пропонує широкий вибір шаблонів та стилів, що дозволяють створювати професійно виглядаючі сертифікати для будь-яких нагод. Платформа забезпечує простий інтерфейс drag-and-drop, який не вимагає спеціальних навичок дизайну, що дозволяє користувачам швидко налаштувати текст, кольори, шрифти та фони. Canva також пропонує можливість додавання високоякісних зображень та ілюстрацій, а також функцію спільної роботи над дизайном в реальному часі. Готові сертифікати можна зберегти у форматах PDF, JPG або PNG, або замовити їх друк через Canva Print.

Canva має багатий набір шаблонів для різних типів сертифікатів, включаючи сертифікати визнання, участі, завершення курсів, досягнень та дипломи. Користувачі можуть легко персоналізувати дизайн, додаючи власні елементи, тексти та зображення, і замовляти високоякісні друки зі швидкою доставкою.

Canva робить процес створення сертифікатів доступним для всіх, навіть без досвіду в дизайні, і надає інструменти для створення привабливих та професійних сертифікатів, які можна використовувати для різних подій (Рисунок 2.3 та Рисунок 2.4).

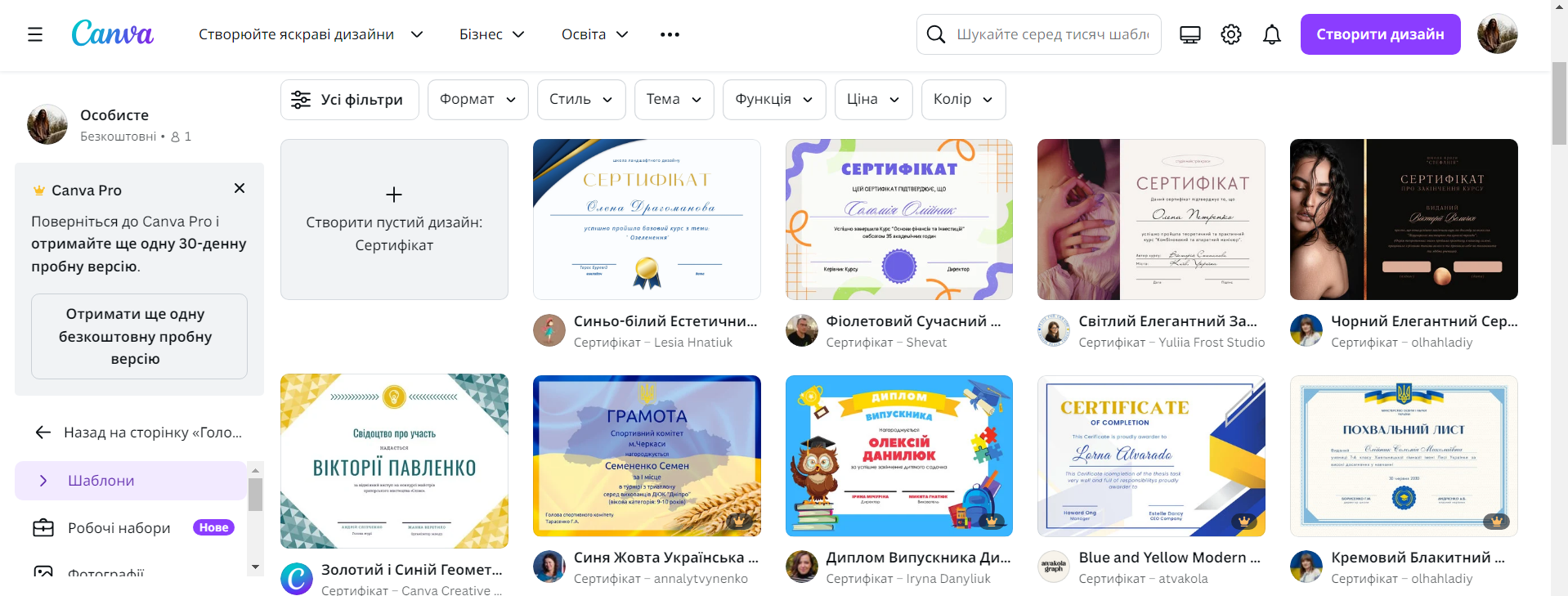


Рисунок 2.3 – Загальний вигляд інтерфейсу програми Canva

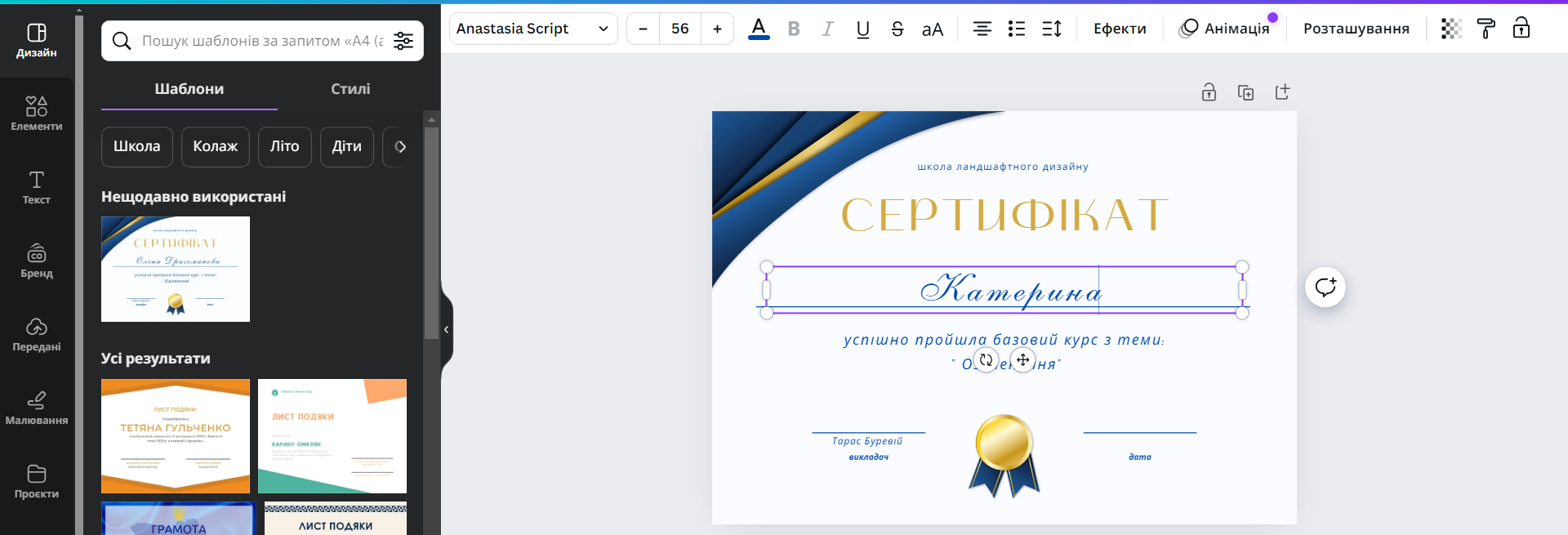
****

Рисунок 2.4 – Загальний вигляд створення сертифікату в Canva

## 2.3. Online Certificate Maker

Online Certificate Maker [9] пропонує безкоштовний інструмент для створення сертифікатів в онлайн режимі. Користувачі можуть легко генерувати сертифікати для різних подій, таких як досягнення студентів або співробітників. Сервіс підтримує автоматичне створення сертифікатів за допомогою API, що дозволяє згенерувати велику кількість документів.

Ключові особливості:

* високоякісні шаблони векторного формату;
* автоматичне налаштування розміру шрифта для довгих імен;
* підтримка різних мов, включаючи англійську, китайську, японську, арабську та інші;
* налаштування кольорів та шрифтів;
* можливість завантаження сертифікатів у форматах PDF та JPG.

Цей інструмент ідеально підходить для професіоналів, які хочуть швидко створити красиві та якісні сертифікати без необхідності дизайнерських навичок. Користувачі можуть створювати і завантажувати до п'яти сертифікатів щодня безкоштовно.

Сайт також пропонує інші безкоштовні інструменти, такі як генератор запрошень на весілля, генератор ID-карток для заходів та інші (Рисунок 2.4).



Рисунок 2.4 – Головна сторінка програмного продукту Online Certificate Maker

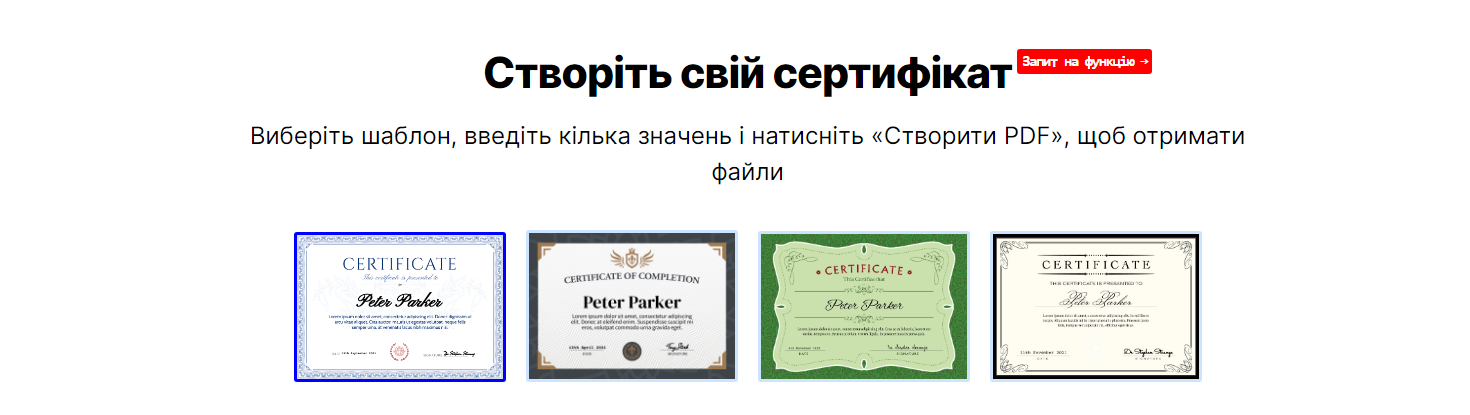


Рисунок 2.5 – Створення сертифікату (Частина 1)



Рисунок 2.5 – Створення сертифікату (Частина 2)

В розділі було проаналізовано ряд продуктів аналогічного призначення, а саме Free Certificate Maker for Custom Certificates, Free Online Certificate Maker (Canva) та Online Certificate Maker. Порівняльний аналіз було виконано на основі таких критеріїв, як адаптивність, багатокористувацький режим, інтуїтивно-зрозумілий інтерфейс користувача, наявність української мови, можливість імпорту/експорту даних, крос-платформність (наявність версій для Linux/MacOS), автентифікація з використанням соціальних облікових записів (Google/Facebook/Twitter тощо), наявність довідкових матеріалів та документації, можливості з налагодження та розширення функціональності (Rest API/SDK), вартість ліцензії.

Провівши порівняльний аналіз програмних продуктів аналогічного призначення результати заносимо в таблицю – Порівняльна характеристика програмних продуктів аналогічного призначення (Таблиця 1) в якій «+» позначаємо наявність, а «–» це відсутність реалізації того чи іншого критерія обраного програмного забезпечення.

**Висновки за розділом**

Виконаний нами порівняльний аналіз дав наступні результати:

* усі розглянуті програмні продукти отримали позитивну оцінку за адаптивність, що робить їх придатними для використання на різних пристроях та різних розмірах екрану;
* так само всі розглянуті продукти отримали позитивну оцінку за інтуїтивно-зрозумілий інтерфейс, що сприяє зручності та ефективності використання;
* всі розглянуті продукти підтримують українську мову, що може бути конкурентною перевагою на ринку україномовних користувачів;
* у всіх програмних продуктах є можливість імпорту та експорту даних, що є важливим для обміну інформацією з іншими системами або збереженням даних у зручному форматі;
* у всіх розглянутих продуктах є крос-платформна підтримка, що дозволяє їх використання на різних операційних системах.

Таблиця 1. Порівняльна характеристика програмних продуктів аналогічного призначення

|  | Free Certificate Maker for Custom Certificates | Free Online Certificate Maker | Online Certificate Maker | Створений програмний продукт |
| --- | --- | --- | --- | --- |
| Адаптивність | + | + | + | + |
| Багатокористувацький режим | – | + | – | + |
| Інтуїтивно-зрозумілий інтерфейс користувача | + | + | + | + |
| Наявність української мови | + | + | + | + |
| Можливість імпорту/експорту даних | + | + | + | + |
| Крос-платформність (наявність версій для Linux/MacOS) | + | + | + | + |
| Автентифікація з використанням соціальних облікових записів (Google/Facebook/Twitter тощо) | + | + | + | – |
| Наявність довідкових матеріалів та документації | + | + | + | + |
| Можливості з налагодження та розширення функціональності (Rest API/SDK) | + | + | + | + |
| Вартість ліцензії | Pro 1004.70 грн /місяць  Starter 497.27 грн/місяць | Преміум 608.91 грн /місяць | Enterprise  12137.56 грн /місяць | – |

# 

# 3. ВИЯВЛЕННЯ ВИМОГ ДО СИСТЕМИ. ОПИС ЇЇ ФУНКЦІОНАЛЬНОСТІ

В цьому розділі буде зосереджено увагу на описі функціональності створеного веб-застосунку. Цей веб-застосунок розроблений для зручного та ефективного створення, експорту та друку сертифікатів, що підтверджують участь у різних подіях, таких як тренінги, вебінари, конференції тощо. Основними функціями веб-застосунку є надання можливості користувачам обирати шаблони сертифікатів, введення необхідних даних для персоналізації сертифікату, автоматична генерація унікального номера для кожного сертифікату та збереження відповідних записів у базі даних. Також передбачається розмежування прав доступу для різних користувачів системи залежно від їх ролей. Крім того, веб-застосунок має привабливий та адаптивний інтерфейс користувача для зручного використання на різних пристроях. Готовий продукт буде опублікований в Мережі, щоб забезпечити його доступність для використання широким колом користувачів.

## 3.1. Види вимог до програмного продукту

У ряді публікацій формування вимог до програмних систем (ПС) розглядається як певна ділова гра, під час якої виявляються інтереси зацікавлених у розробці ПС осіб, правила реалізації цих інтересів у конкретному продукті. При цьому висловлюються різного роду претензії й обмеження на властивості і способи забезпечення вимог для отримання кінцевого програмного продукту. Отже, зрозуміло що нема формалізованих методів збирання й опису вимог, а також відсутнє загальноприйняте визначення самого поняття вимога. Наведемо тлумачення цього слова з джерел [1–3].

Вимоги – це твердження про функції й обмеження системи.

Вимоги – це властивості, які повинен мати продукт, щоб являти бути собою цінним для свого користувачів.

Вимоги – це специфікація того, що і як має бути реалізовано.

Відповідно до міжнародного глосарію з термінології комп’ютерної науки [10] вимога містить у собі опис:

* умов або можливостей, необхідних користувачеві для вирішення поставлених проблем або для досягнення цілей;
* функцій і обмежень, які повинна мати система або системні компоненти, щоб виконати контракт замовника на розробку;
* положень і регламенту, встановлених використаними стандартами і відображених у специфікаціях або інших формальних документах на систему;
* умов, можливостей і обмежень середовища, необхідних для проектування і виконання системи.

Вимоги до продукту охоплюють умови користувачів щодо зовнішнього поводження системи і погляди розробників на деякі параметри системи. Термін користувач стосується осіб, зацікавлених у створенні системи.

Вимоги до програмного забезпечення (ПЗ) можуть бути системними, функціональними і нефункціональними. Вимоги користувачів (user requirements) задаються умовами досягнення цілей і задач, віддзеркалюють вимоги споживачів до спектра розв’язуваних майбутньою системою задач. Вони подаються як текстовий опис або сценарії, прецеденти, таблиці «подія-відгук» тощо.

Системні вимоги (system requirements) визначають зовнішні умови виконання системних функцій і обмежень на створення продукту, а також вимоги до опису програмно-апаратних підсистем. Системні вимоги накладають обмеження на архітектуру системи, засоби її візуального подання і функціонування. Для опису системних вимог використовують спеціальні шаблони і форми, що допомагають уявити вхідні і вихідні дані й автоматизувати ці вимоги.

Вимоги до атрибутів якості (quality attributes) – це деякі обмеження на властивості функцій або системи, важливі для користувачів або розробників. Наприклад, переносність, цілісність, стійкість системи до збоїв.

Функціональні вимоги – це перелік функцій або сервісів, які повинна надавати система, а також обмежень на дані і поводження системи при їхньому виконанні.

Специфікація функціональних вимог (software requirements specification) – опис функцій та їхніх властивостей, які не містять у собі протиріч і виключень.

Нефункціональні вимоги визначають умови виконання функцій (наприклад, захист інформації у базі даних (БД), аутентифікація доступу до програмної системи (ПС) тощо) у середовищі, що безпосередньо не пов'язані з функціями, а відбивають потреби користувачів щодо їх виконання. Ці вимоги характеризують принципи взаємодії із середовищами або іншими системами, а також визначають показники часу роботи, захисту даних і досягнення якості з урахуванням рекомендацій використовуваного стандарту. Вони можуть встановлюватися як числові значення (наприклад, час чекання відповіді, кількість клієнтів, що обслуговуються і ін.) у різних одиницях виміру, включаючи, наприклад, ймовірність (значення ймовірності безвідмовної роботи системи – показника її надійності). Для більшості сучасних ПС вимоги складаються з умов і обмежень типу:

* конфіденційність, безпека і захист даних;
* відмовостійкість;
* одночасність доступу користувачів до системи;
* час чекання відповіді при звертанні до системи (продуктивність);
* склад виконуваних функцій системи (запуск, швидкість реакції й ін.);
* положення стандартів з виконання сформульованих вимог.

Дані вимоги визначаються і формалізуються аналітиками на процесі аналізу і проектування структури системи. Так, у випадку вимог з безпеки функціонування системи, в системі виділяються категорії користувачів, що мають право доступу до тих або інших функцій (програмних компонентів) або даних, та передбачаються додаткові функції системи з перевірки доступу (санкціонований доступ до них чи ні). Якщо потрібно обмежити доступ до конкретних даних (наприклад, до окремих записів, полів у таблиці), то в системі може передбачатися, наприклад, мандатний захист.

Для захисту всієї системи від несанкціонованого доступу користувачі реєструються і проходять аутентифікацію для роботи із системою.

Для відновлення і збереження резервних копій БД, архівів баз даних аналізуються можливості системи управління базами даних (СУБД) і способи забезпечення необхідного рівня безперебійної роботи системи, правил доступу авторизованих користувачів і заходів боротьби з різними загрозами, що надходять ззовні від користувачів, які не мають прав доступу до деяких або до всіх даних системи.

До вихідного продукту пред'являються нефункціональні вимоги до:

* застосування (якість інтерфейсу, продукту й ін.);
* продуктивності (пропускна здатність, час реакції й ін.);
* надійності виконання (без помилок і відмов);
* зовнішніх інтерфейсів, за якими виконується взаємодія з іншими компонентами або підсистемами.

Опис усіх видів вимог проводиться з урахуванням стандартів, наприклад, стандарту з якості ISO/IEC ДСТУ 9126 і стандартизованого понятійного і термінологічного довідника, що містить у собі загальноприйняті терміни щодо структури ПрО і призначення функцій системи. Специфікація вимог відображає принципи взаємодії проектованої системи з іншими середовищами, платформами і загальносистемним забезпеченням (БД, СКБД, мережі та ін.).

Формування документа зі специфікаціями вимог завершується на процесі проектування архітектури, після чого він узгоджується з замовником системи і використовується як керування дій при виконанні задач розробки програмного продукту на процесах життєвого циклу (ЖЦ) і отриманні готового продукту. При виявленні на них неузгоджених вимог, проводиться їхнє уточнення і, відповідно, вносяться зміни у деякі задачі процесу розроблення системи або характеристики продукту.

Остаточно сформульовані вимоги – основа для підписання контракту між замовником і розробником системи [1].

## 3.2. Аналіз прецедентів

Діаграма прецедентів – це узагальнене представлення функціонального призначення системи, яке має відповідати на головне питання моделювання: що робить система у зовнішньому світі? На діаграмі прецедентів застосовують два типи базових сутностей: варіанти використання (прецеденти) і діючі особи (актори), між якими встановлюють такі типи відношень:

* асоціація між актором та прецедентом;
* узагальнення між акторами;
* узагальнення між прецедентами;
* залежності чи асоціації (різних типів) між прецедентами.

Під час задання прецедентів акцентують увагу на поводженні системи стосовно заданого актора. Прецеденти допомагають визначити можливості та межі системи. Нотація для прецеденту досить скромна – це текст (наказова дієслівна форма), який, залежно від Case-засобу, розташований в овалі або під овалом.

Прецеденти визначають за специфікацією вимог до системи. Актором може бути будь-який взаємодіючий з системою зовнішній суб’єкт: фізична особа (наприклад, торговий агент), зовнішня програмна система (скажімо, програма підготовки і друку рахунків) чи пристрій (наприклад, датчик температури). Усі процеси взаємодії між діючими суб’єктами і системою розглядають як прецеденти. Діаграми прецедентів відображають статичні аспекти системи з позиції користувачів. Ця позиція охоплює, здебільшого, поведінку системи, тобто видимі ззовні сервіси, які надає система.

UML дає змогу моделювати контекст за допомогою діаграм прецедентів, у яких увагу акцентують на зовнішніх акторах. Важливо правильно визначити акторів, оскільки це даватиме змогу описати класи сутностей, що взаємодіють із системою. Ще важливіше визначити, хто не є актором, адже при цьому обмежується оточення системи (у ньому залишаються тільки ті елементи, що беруть участь у її роботі) [11].

ПРЕЦЕДЕНТ: "УВІЙТИ ДО ВЕБ-ЗАСТОСУНКУ"

Ектор: Користувач

Передумова: Користувач має доступ до Інтернету та URL адресу веб-застосунку.

Післяумова: Користувач успішно увійшов до веб-застосунку та отримав доступ до його функціональності.

Сценарій:

1. Користувач відкриває веб-браузер на своєму пристрої.
2. Користувач вводить URL адресу веб-застосунку у адресному рядку браузера.
3. Система переспрямовує користувача на сторінку входу.
4. Користувач вводить свої облікові дані (логін та пароль) у відповідні поля на сторінці входу.
5. Система перевіряє правильність введених даних.
6. Якщо введені дані правильні, система авторизує користувача і перенаправляє його на головну сторінку веб-застосунку.
7. Користувач отримує доступ до функціональності веб-застосунку, залежно від його ролі та прав доступу.
8. Якщо введені дані неправильні, система відображає повідомлення про помилку та пропонує спробу увійти ще раз.

ПРЕЦЕДЕНТ: ЗАПОВНИТИ РЕЄСТРАЦІЙНУ ФОРМУ

Ектор: Користувач

Передумова: Користувач має доступ до системи і бажає зареєструвати нову подію.

Післяумова: Інформація про подію та учасників успішно збережена у системі.

Сценарій:

1. Користувач увійшов до системи.
2. Користувач вводить назву події у відповідне поле.
3. Користувач обирає потрібний шаблон для події із доступних варіантів або обирає власний.
4. Користувач вводить тривалість події у відповідне поле.
5. Користувач вводить ПІБ усіх учасників події у відповідні поля або додає їх як окремі записи.
6. Користувач вводить ПІБ того, хто проводив захід, у відповідне поле.
7. Система зберігає введену інформацію та підтверджує успішну реєстрацію події.

ПРЕЦЕДЕНТ: ЗГЕНЕРУВАТИ СЕРТИФІКАТ

Ектор: Користувач

Передумова: Користувач вже заповнив всі необхідні дані.

Післяумова: Сертифікат успішно згенеровано, і користувач має можливість завантажити його або роздрукувати.

Сценарій виконання:

1. Після заповнення форми, користувач підтверджує створення сертифікату.
2. Користувач має можливість завантажити згенерований сертифікат у форматі PDF або роздрукувати його прямо з веб-застосунку.

ПРЕЦЕДЕНТ: ЗБЕРЕЖЕННЯ КОРИСТУВАЧА В БАЗІ ДАНИХ

Передумова: Користувач зайшов на веб-застосунок та зареєструвався в ньому.

Післяумова: Користувач збережений в базі даних з усіма необхідними даними.

Сценарій:

1. Система отримує дані, що були введені користувачем під час реєстрації.
2. Система перевіряє ці дані на коректність та відповідність вимогам до формату.
3. Якщо дані введені коректно, система зберігає їх в базі даних разом з індивідуальним ідентифікатором користувача.
4. Система надсилає підтвердження про успішну реєстрацію користувачеві.
5. Якщо дані не коректні, система повідомляє користувача про помилку та просить відредагувати їх.

ПРЕЦЕДЕНТ: ГЕНЕРАЦІЯ УНІКАЛЬНОГО НОМЕРУ СЕРТИФІКАТУ ТА ЗБЕРЕЖЕННЯ ЗАПИСУ ПРО НЬОГО У БАЗІ ДАНИХ

Передумова: Система отримує запит на створення нового сертифікату.

Післяумова: Система успішно генерує унікальний номер сертифікату і зберігає запис про нього у базі даних.

Сценарій:

1. Система отримує запит на створення нового сертифікату.
2. Система генерує унікальний номер сертифікату.
3. Система зберігає запис про сертифікат разом з його унікальним номером у базі даних.

Діаграма прецедентів наочно демонструє можливості системи, а також взаємодію користувача з нею. Діаграму для нашого продукту показано на Рисунку 3.1.

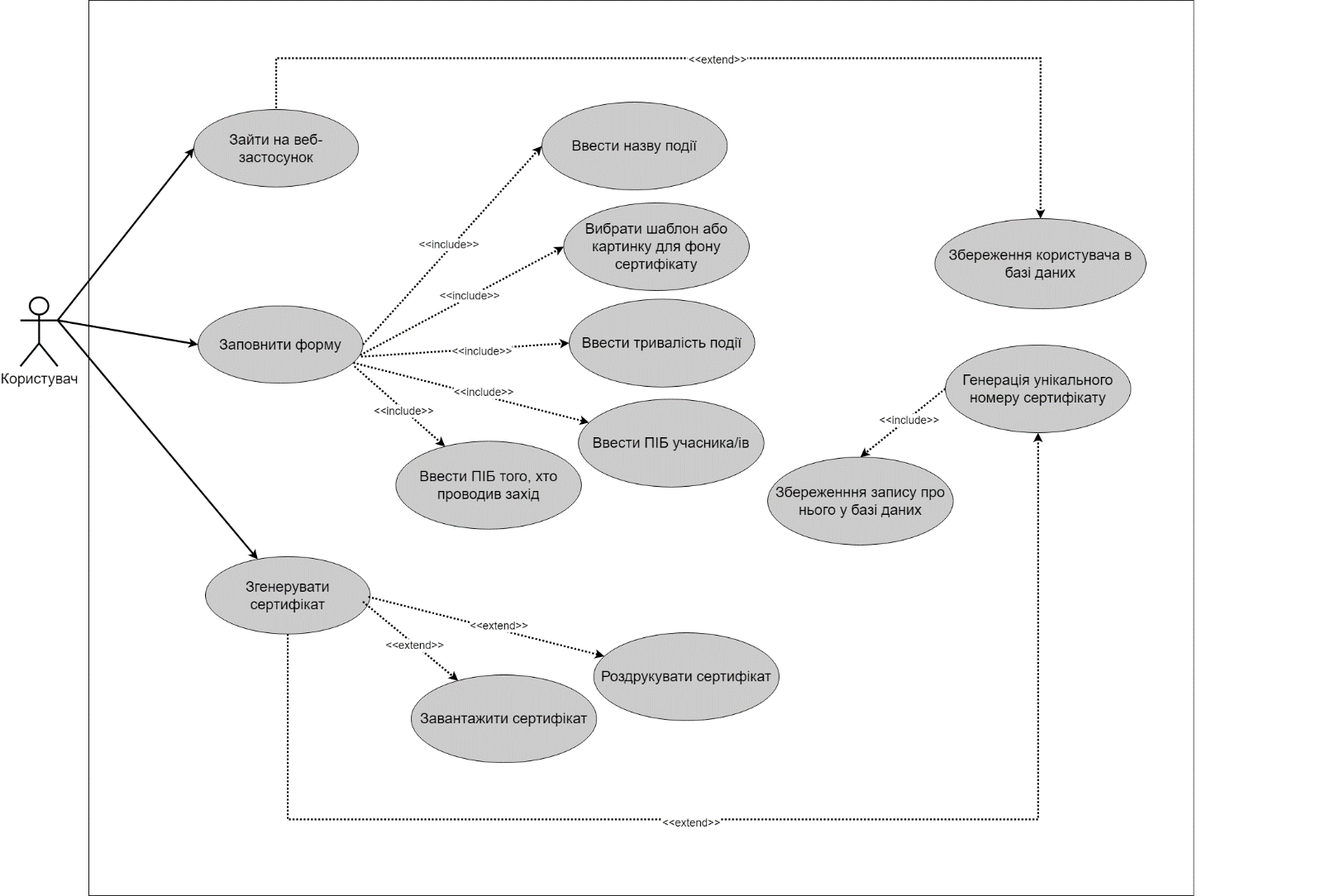


Рисунок 3.1 – Діаграма прецедентів створюваного веб-застосунку

## 3.3. Опис функціональності

Функціональність програмного забезпечення – це здатність програмного продукту виконувати набір функцій, визначених у його зовнішньому описі, і які відповідають заданим або імовірною потребам користувачів [12].

ФУНКЦІОНАЛЬНІ ВИМОГИ:

* реєстрація та авторизація користувачів;
* можливість створення нового сертифікату;
* вибір шаблону для сертифікату;
* введення або вибір необхідних даних для сертифікату;
* автоматична генерація унікального номера сертифікату;
* можливість експорту сертифікату у форматі PDF;
* можливість друку сертифікату;
* збереження запису про кожен створений сертифікат у базі даних;
* розмежування прав доступу користувачів;

НЕФУНКЦІОНАЛЬНІ ВИМОГИ:

* привабливий та зручний інтерфейс користувача;
* адаптивний дизайн для відображення на різних пристроях;
* висока швидкість роботи веб-застосунку навіть при великій кількості одночасних користувачів;
* захист персональних даних користувачів та конфіденційності інформації про сертифікати;
* стійкість до збоїв та відновлення роботи після відмови;
* масштабованість системи для можливості розширення функціональності та обробки більшої кількості даних у майбутньому;
* відповідність стандартам безпеки Інтернету;
* підтримка міжнародних стандартів форматування дат та часу для сертифікатів;
* простота встановлення та підтримки.

У розділі були визначені вимоги до програмного продукту, що є веб-застосунком для генерації, експорту та друку сертифікатів про проходження тренінгів, участь у вебінарах та заходах. Були визначені користувачі системи, такі як адміністратори та звичайні користувачі, які можуть створювати та редагувати сертифікати.

Основними прецедентами є:

* створення сертифікату;
* експорт сертифікату;
* друк сертифікату;
* автоматична генерація унікального номера сертифікату та збереження запису про нього у базі даних;
* збереження записів про сертифікати у базі даних;
* збереження записів про користувачів у базі даних.

**Висновки за розділом**

В результаті аналізу, виконаного під час написання цього розділу, виявлені вимоги ми візуалізували за допомогою діаграми прецедентів UML, де кожен прецедент пов’язаний з відповідним ектором і відображає взаємодію між користувачами та системою.

# 4. АРХІТЕКТУРА СИСТЕМИ

## 4.1. Поняття архітектури програмного продукту

Архітектура програмного продукту (ПП) – це будова програмного продукту, тобто представлення його як системи, що складається з деякої сукупності взаємодіючих підсистем. У ролі підсистем можуть виступати окремі програмні компоненти, сервіси чи застосунки.

Розробка архітектури є першим етапом боротьби зі складністю ПП, на якому реалізується принцип виділення незалежних компонент.

Основні завдання розробки архітектури ПП:

* виділення програмних підсистем і відображення в них зовнішніх функцій;
* встановлення способів взаємодії між виділеними програмними підсистемами.

Архітектура ПП розробляється на основі функціональної специфікації (ФС). Тобто, проводиться подальша конкретизація функціональної специфікації, яка призводить до формування архітектури.

В більшості випадків для проектування і візуалізації архітектури програмного продукту використовують діаграми. Дуже часто це діаграми UML, зокрема, діаграми класів, компонентів, пакетів, розгортання. Слід відзначити також, що наявність діаграм не звільнює команду від необхідності формування традиційної текстової документації

Основні питання для розробника архітектури ПП:

* як користувач буде використовувати ПП?
* як ПП буде розгортатися і обслуговуватися при експлуатації?
* які висуваються вимоги до таких атрибутів ПП, як якість, безпека, продуктивність, можливість паралельної обробки, інтернаціоналізація і конфігурація?
* як спроектувати ПП, щоб він залишався гнучким і зручним в обслуговуванні протягом усього періоду експлуатації?
* які існують основні архітектурні елементи?

Необхідно пам'ятати, що архітектура має:

* формувати структуру системи, але приховувати її;
* розкривати всі можливі варіанти використання і сценарії ПП;
* відповідати вимогам усіх зацікавлених сторін;
* виконувати вимоги як функціональні, так і вимоги стосовно якості продукту.

ПП має бути розділений на окремі компоненти з мінімальним перекриттям функціональності. Важливим чинником є граничне зменшення кількості точок зіткнення, що забезпечить високу зв'язність (high cohesion) і слабке зчеплення (low coupling). Неправильне розмежування функціональності може призвести до високої зв'язаності й ускладнити взаємодію між компонентами, навіть не дивлячись на слабке перекриття функціональності окремих компонентів.

Проектувати потрібно тільки те, що є необхідним. Лише в окремих випадках, коли вартість розробки (або втрат у разі невдалого проектування) дуже високі, допускається повне попереднє проектування і тестування. У інших випадках, особливо при гнучкій розробці, можна уникнути масштабного попереднього проектування. Особливо – якщо вимоги до ПП розпливчасті і чітко не визначені, або існує вірогідність зміни дизайну з часом.

Архітектура програмної системи практично ніколи не обмежена лише одним архітектурним стилем, часто вона є поєднанням архітектурних стилів, утворюючих повну систему. Наприклад, може існувати SOA (архітектура, орієнтована на сервіси) – дизайн, що складається з сервісів, при розробці яких використовувалася багатошарова архітектура і об'єктно-орієнтований архітектурний стиль.

Поєднання архітектурних стилів також корисно при побудові веб-застосунків, де можна досягти ефективного розділення функціональності за рахунок застосування багатошарового архітектурного стилю. Таким чином можна відокремити логіку уявлення від бізнес-логіки і логіки доступу до даним. Рівень уявлення може розгортатися в демілітаризованій зоні (DMZ), розташованій між внутрішньою мережею організації і зовнішньою мережею [13].

## 4.2. Архітектури веб-застосунків

Архітектура веб-застосунків описує взаємозв’язок між компонентами веб-застосунку, їхню структуру та спосіб взаємодії. Вона визначає, як розподіляються функції та обов’язки між різними частинами застосунку, щоб забезпечити його ефективну роботу та легкість розширення.

Одним з найпоширеніших підходів до архітектури веб-застосунків є клієнт-серверна модель. У цій моделі, веб-застосунок складається з двох основних компонентів: клієнта та сервера. Клієнт (зазвичай веб-браузер) відповідає за відображення інтерфейсу користувача та збирання введених даних, тоді як сервер обробляє запити клієнта, виконує бізнес-логіку та надає необхідні дані.

Веб-застосунки складаються з різних компонентів, які співпрацюють між собою для забезпечення функціональності додатку. Основні компоненти включають:

Клієнтська частина (Front-end): Це інтерфейс, який відображається користувачеві у веб-браузері. Вона включає HTML, CSS та JavaScript, які використовуються для створення інтерактивних елементів та забезпечення користувацького досвіду.

Серверна частина (Back-end): Це сервер, який обробляє запити клієнта, зберігає та обробляє дані, виконує бізнес-логіку та повертає відповіді клієнту. Вона може включати різні технології, такі як веб-сервери, бази даних тощо.

У архітектурі веб-застосунків використовуються різні шаблони, які допомагають організувати та структурувати компоненти додатку. Деякі з найпоширеніших архітектурних шаблонів включають монолітну архітектуру, клієнт-серверну архітектуру та мікросервісну архітектуру [14].

Монолітна архітектура – традиційний підхід до побудови програмних продуктів, за якого вся функціональність зосереджена у кодовій базі та виконується в єдиному процесі. Весь стек застосунку, включно з призначеним для користувача інтерфейсом, бізнес-логікою і доступом до даних, інтегрований в одну програму.

Одна з ключових особливостей «моноліту» – наявність єдиного кодового базису. Це означає, що весь вихідний код застосунку знаходиться в одному місці, що полегшує його розуміння і розробку. Запуск і оновлення продукту з монолітною архітектурою вимагає мінімальних зусиль, оскільки всі компоненти знаходяться разом. Це робить процес розгортання відносно простим і зрозумілим для розробників. Завдяки централізованій структурі, процеси моніторингу та налагодження також стають простішими. Розробники можуть використовувати стандартні інструменти для відстеження проблем і поліпшення продуктивності.

Однак у цієї архітектури є й недоліки. У міру того, як монолітний застосунок збільшується в обсязі, можуть виникати обмеження в його гнучкості та масштабованості. Також при внесенні правок може знадобитися зміна всього кодового базису, а це тривалий і трудомісткий процес. У разі збою навіть одного компонента можливі серйозні проблеми у функціонуванні всього застосунку.

Монолітна архітектура – проста і зрозуміла модель розробки, яка особливо підходить для невеликих і середніх проєктів. Однак зі зростанням складності та масштабу можливі обмеження і збільшення ризиків.

Мікросервісна архітектура – підхід до розробки програмного забезпечення, у відповідності до якого застосунок розбивається на невеликі автономні сервіси, кожен з яких виконує обмежений обсяг функціональності. Усі складові системи взаємодіють між собою через API (програмний інтерфейс застосунку), що дає змогу їм функціонувати незалежно один від одного.

З визначення зрозуміло, що кожен елемент «мікросервісу» відповідає за певну функціональність, що забезпечує легкість у підтримці та модифікації IT-продуктів. Припустимо, у нас є інтернет-магазин. Замість монолітного застосунку, ми можемо розділити його на мікросервіси: для управління продуктами на складі, для обробки замовлень, для обліку користувачів тощо. Кожен сервіс відповідає за свій аспект бізнесу, що забезпечує гнучкість і масштабованість.

Мікросервісна архітектура – рішення для складніших і масштабніших проєктів, де гнучкість, незалежність розроблення та стійкість до відмов зазначені в ключових вимогах. Однак її успішна реалізація вимагає уважного управління комунікацією між сервісами [15].

Архітектура клієнт-сервер – це концепція інформаційної мережі, в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів. Розглянута архітектура визначає два типи компонентів: сервери і клієнти.

Сервер – це об’єкт, що надає сервіс (послуги) іншим об’єктам мережі за їх запитами. Сервіс – це процес обслуговування клієнтів.

Сервер працює за завданнями клієнтів і керує виконанням їх завдань. Після виконання кожного завдання сервер надсилає отримані результати клієнту, від якого він отримав це завдання.

Сервісна функція в архітектурі клієнт-сервер описується комплексом прикладних програм, відповідно до якого виконуються різноманітні прикладні процеси.

Процес, який викликає сервісну функцію за допомогою певних операцій, називається клієнтом. Ним може бути програма або користувач.

Клієнти – це робочі станції, які використовують ресурси сервера і надають зручні інтерфейси користувача. Інтерфейси користувача – це процедури взаємодії користувача з системою або мережею.

Клієнт є ініціатором і використовує сервіси сервера. У цьому процесі клієнт запитує вид обслуговування, встановлює сеанс, отримує потрібні йому результати і повідомляє про закінчення роботи.

У мережах з виділеним файловим сервером на виділеному автономному ПК встановлюється серверна мережева операційна система. Цей ПК стає сервером. Програмне забезпечення (ПЗ), встановлене на робочій станції, дозволяє їй обмінюватися даними з сервером.

Крім мережевої операційної системи необхідні мережні прикладні програми, що реалізують переваги, надані мережею.

Мережі на базі серверів мають кращі характеристики і підвищену надійність. Сервер володіє головними ресурсами мережі, до яких звертаються інші робочі станції.

У сучасній клієнт – серверній архітектурі виділяється чотири групи об’єктів: клієнти, сервери, дані і мережеві служби.

Клієнти розташовуються в системах на робочих місцях користувачів. Дані в основному зберігаються на серверах. Мережеві служби є спільно використовуваними серверами і даними. Крім того служби керують процедурами обробки даних [16].

## 4.3. Опис структури створеного продукту. Опис класів.

Програмний продукт складається з кількох основних компонентів, рознесених між клієнтською (Frontend) та серверною (Backend) частинами.

Спочатку про Frontend частину даного проекту. Перш за все метою створення цього компоненту є розробка привабливого та інтуїтивно зрозумілого інтерфейсу для користувача. Використано сучасні технології та практики дизайну для забезпечення адаптивності до різних пристроїв та розмірів екранів. Не менш важливою є реалізація функцій для вводу необхідних даних, вибору шаблону сертифікату, попереднього перегляду, а також експорту або друку сертифікату. Також він відповідає за реалізацію механізмів аутентифікації користувачів і контроль доступу до функціональності за різними ролями (викладач, учень).

Backend частина цього проекту була розроблена для реалізації бази даних для зберігання інформації про користувачів, шаблони сертифікатів та створені сертифікати. Також її використання було необхідним для реалізації логіки для автоматичної генерації унікальних номерів сертифікатів на основі введених даних та обраного шаблону. І не менш важливим аспектом була реалізація API для взаємодії між Frontend та Backend частинами. Це дозволило розширити функціональність застосунку та інтегрувати його з іншими системами, якщо потрібно.

Також не слід забувати що для цього програмного продукту було важливим обрання хостинг-провайдера для розміщення веб-застосунку та його публікації в Інтернеті, а також забезпечення захисту, конфіденційності та цілісності даних, а також захист від потенційних кібератак.

Далі про класи. Але перш ніж описати класи, що наявні в проекті, хотілось би нагадати, що таке взагалі класи.

В ООП клас – це шаблон, за яким створюються об’єкти. Він допомагає групувати разом пов’язані змінні та функції. Коли створюється екземпляр класу, то це якраз і є той об’єкт, який був створений на його основі. А клас може використовуватися для створення безлічі об’єктів, які відрізнятимуться один від одного своїм станом (тобто значеннями полів).

Клас дає змогу створювати об’єкти за своїм унікальним шаблоном. Коли створюється клас, то йому задається набір властивостей і методів, а потім його можна використовувати для створення об’єктів, які матимуть ті самі властивості та методи. Це як Lego: створюється свій унікальний блок, а потім використовується для створення різних конструкцій. У класі можна задати властивості, що визначають стан об’єкта (колір, розмір, ім’я) і методи, що визначають, як об’єкт взаємодіятиме з іншими об’єктами та з навколишнім середовищем [17].

Тепер про класи в проекті. Програмний продукт містить наступні класи:

user (Користувач) – відповідає за представлення та обробку даних про користувачів, включаючи їх автентифікацію та авторизацію;

certificate (Сертифікат) – містить інформацію про створені сертифікати, таку як унікальний номер, дані користувача, вибраний шаблон сертифікату тощо. Відповідає за генерацію та збереження даних про сертифікати;

template (Шаблон сертифікату) – представляє різні шаблони для сертифікатів, включаючи їх дизайн та вміст. Діаграму класів для нашого застосунку показано на Рисунку 4.1.

## PlantUML diagram

Рисунок 4.1 – Діаграма класів створюваного веб-застосунку

## 4.4. Опис бази даних

В проекті було використано базу даних MongoDB. MongoDB – це NoSQL СУБД, що створена для розробки сучасних додатків. Вона гнучка, з нею просто працювати та вирішувати нестандартні завдання.

Поки реляційні (SQL) бази розкладають інформацію за жорстко пов'язаним та структурованим таблицями, MongoDB вносить дані в документи, що мають структуру, подібну до JSON. У них теж буде структура та ієрархія, але менш жорстка. Простий приклад: зберігання даних користувача у якого не один номер телефону та адреса, а декілька, без підключення розширень та інших доробок. Робота з такими об'єктами інтуїтивно зрозуміла розробникам сайтів та додатків, а сама модель зберігання добре підходить для документів з унікальними даними, які змінюються з часом, наприклад, профілів користувачів та каталогів [18].

Кожна база даних MongoDB містить колекції, які, у свою чергу, містять документи. Кожен документ може мати різну кількість полів. Розмір і зміст кожного документа можуть відрізнятися один від одного.

Структура документа більше відповідає тому, як розробники будують свої класи та об’єкти на відповідними мовами програмування. Розробники часто кажуть, що їхні класи не є рядками та стовпцями, а мають чітку структуру з парами ключ-значення.

Рядки (або документи, як викликано в MongoDB) не потребують попередньої визначеної схеми. Натомість поля можна створювати на льоту.

MongoDB-середовища дуже масштабовані. Компанії в усьому світі визначили що деякі з них працюють із понад 100 вузлами з мільйонами документів у базі даних [19].

Порівняння структури реляційних БД з MongoDB наочно продемонстровано на Рисунку 4.2.

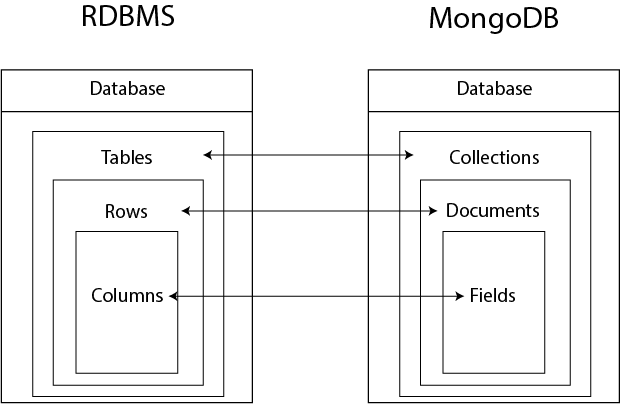


Рисунок 4.2 – Структура реляційних БД з MongoDB

Оскільки як сказано вище, кожна БД в MongoDB містить колекції, в моєму проекті їх буде три. Перша колекція «users» в якій будуть збережені всі дані про користувачів такі як: name, email, password і role. Наступна колекція містить інформацію про сертифікати, а саме «certificates», в ній буде присутня така інформація як унікальний номер сертифікату, кому виданий, ким виданий, дата видачі, тривалість та назва заходу за який видано сертифікат. І остання колекція це «certificate-templates», в якій будуть збережені базові шаблони сертифікату, а також якщо певний користувач (викладач) завантажить свій шаблон, то на нього він автоматично збережеться у базі даних. Також у базі даних наявний такий зв’язок, що при створені викладачем сертифікату для певного користувача системи (учня), він автоматично буде збережений у колекції з сертифікатами на ім’я даного учня і учень матиме змогу переглядати його в своєму кабінеті. Структуру бази даних та зв’язки між колекціями зображено на ER-діаграмі (Рисунок 4.3).

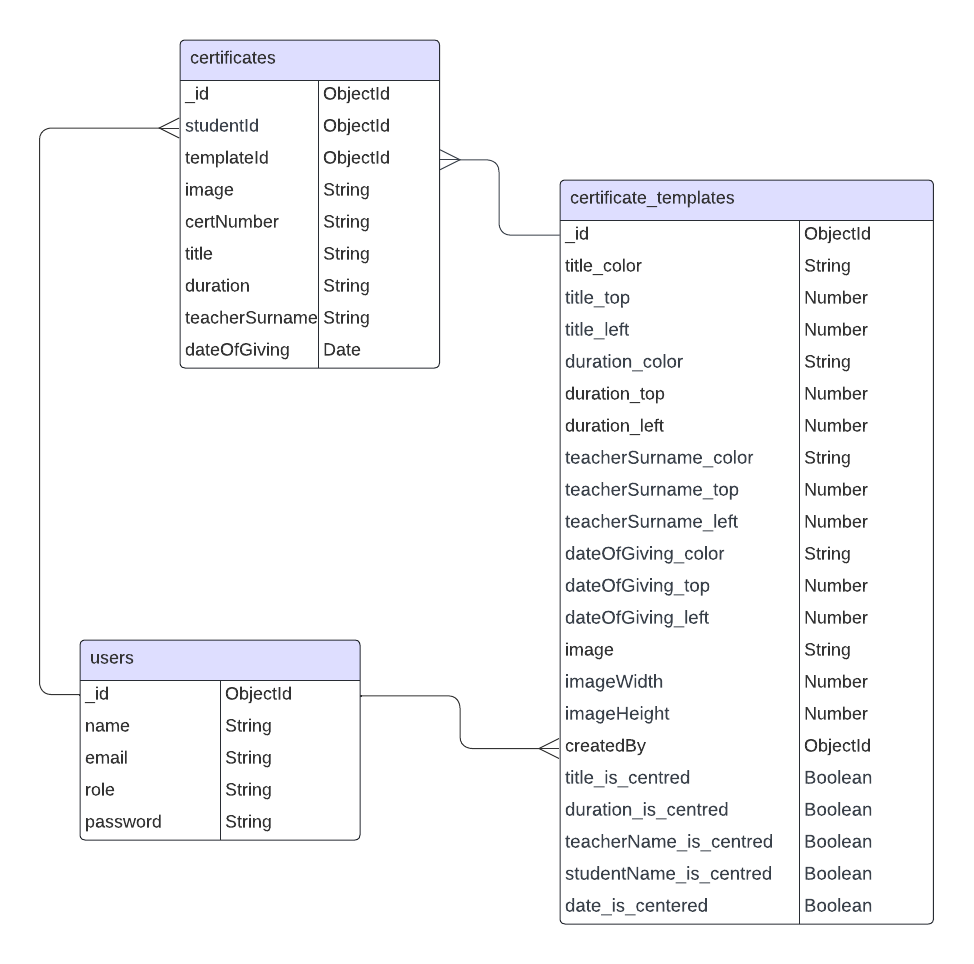


Рисунок 4.3 – ER-діаграма бази даних створюваного веб-застосунку

## 4.5. Особливості розгортання програмного продукту

Розгортання програмного продукту базується на використанні технологій Node.js та Vue.js.

Розгортання Node.js-сервера потрібне для обробки запитів до веб-застосунку. Це може включати встановлення необхідних пакетів, конфігурування середовища виконання, налаштування бази даних та інші необхідні для середовища налаштування.

Vue.js-додаток має бути розгорнутий на веб-сервері, аби бути доступним для користувачів через веб-браузер. Це може бути здійснене за допомогою різних методів, включаючи використання CDN для статичних файлів або розгортання на сервері з використанням веб-сервера.

Програмний продукт потребує бази даних для зберігання інформації про створені сертифікати, події та інші деталі. Налаштування та розгортання бази даних MongoDB забезпечить надійне зберігання даних.

Важливо враховувати аспекти безпеки під час розгортання для забезпечення захищеної комунікації між клієнтом і сервером, а також використання механізмів автентифікації та авторизації для захисту конфіденційності та цілісності даних.

При розгортанні слід врахувати можливість масштабування програмного продукту в майбутньому, якщо кількість користувачів зростатиме.

Після розгортання важливо встановити механізми моніторингу, які дозволять вам відстежувати продуктивність програмного продукту, виявляти проблеми та вчасно реагувати на них.

Розгортання програмного продукту "Генератор сертифікатів" вимагає уважного планування, налаштування та тестування для забезпечення стабільної та ефективної роботи системи.

Для розгортання програмного продукту було використано сервіс Render [20].

Render – це сучасна платформа для хостингу, яка пропонує простий спосіб розгортання та керування веб-додатками та сервісами. Вона дозволяє розгортати як статичні, так і динамічні сайти, бекенд-сервіси і бази даних.

Автоматичні оновлення відбуваються кожного разу, коли ви вносите зміни в репозиторій на GitHub або GitLab. Механізм візуалізації підтримує кілька мов програмування, включаючи Node.js, Python, Ruby, Go та інші. Метод render забезпечує повністю інтегровану базу даних PostgreSQL і Redis, резервна копія якої автоматично створюється та збільшується. Сертифікати SSL є автоматичними та безкоштовними, що забезпечує безпеку зв’язку між програмами. Масштабування ваших додатків легко виконати вгору або вниз залежно від навантаження.

Платформа підтримує фонові процеси, що дозволяє виконувати періодичні завдання без необхідності додаткових серверів. Крім того, візуалізація дозволяє розгортати статичні веб-сайти, які швидко завантажуються та прості в обслуговуванні.

Переваги Render включають в себе легкість у використанні та швидке розблокування, а також високу надійність і автоматичний моніторинг.

Діаграму розгортання нашого продукту показано на Рисунку 4.4.

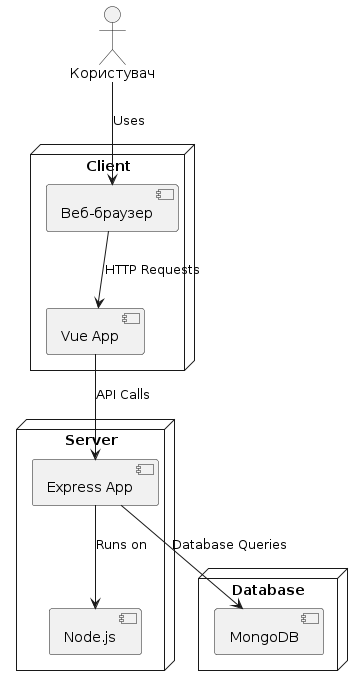


Рисунок 4.4 – Діаграма розгортання створюваного веб-застосунку

**Висновки за розділом**

В цьому розділі було зроблено огляд архітектури системи, опис структури створеного продукту та основних класів, бази даних та розгортання застосунку.

# 5. ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

Проектування інтерфейсу користувача – це ключовий етап у розробці будь-якого програмного продукту або веб-сайту. Його мета полягає в створенні зручного, ефективного та привабливого інтерфейсу, який відповідає потребам та очікуванням користувачів. У цьому розділі будуть розглянуті основні принципи та методи проектування інтерфейсу, інструменти, що використовуються для його реалізації, а також кращі практики, які допоможуть створити продукт, з яким користувачі будуть задоволені.

## 5.1. Поняття інтерфейсу

Дизайн графічного інтерфейсу користувача (GUI) вже давно став невід'ємною частиною нашої цифрової реальності. Від комп'ютерів до смартфонів і планшетів, GUI впроваджується в різноманітні пристрої та програми, щоб зробити їхню взаємодію з людьми легкою та інтуїтивно зрозумілою.

Дизайн графічного інтерфейсу користувача (GUI) – це процес створення та вдосконалення візуального способу взаємодії між людиною та комп'ютером. Він використовує графічні елементи, такі як іконки, кнопки, вікна та панелі керування, щоб дозволити користувачам легко комунікувати з цифровими пристроями, програмами та системами.

Основна мета дизайну GUI – забезпечити інтуїтивну та просту взаємодію, не вимагаючи від користувачів попереднього досвіду або знання конкретних комп'ютерних мов. Він дає можливість користувачам виконувати різноманітні завдання шляхом маніпулювання графічними об'єктами за допомогою миші, тачпаду або екрана смартфона.

Графічний інтерфейс користувача розробляється з урахуванням принципів ергономіки, дотримуючись таких аспектів, як розташування елементів на екрані, використання кольорів, розмірів та форм об'єктів, логічність послідовності дій та зручність способу навігації.

Графічний інтерфейс користувача (GUI) працює шляхом створення взаємодії між користувачем та комп'ютером за допомогою графічних елементів та інструментів. Основна ідея полягає в тому, щоб зробити використання комп'ютера легким та інтуїтивно зрозумілим для користувача, незалежно від його рівня технічних навичок.

Основна парадигма GUI – це «вікна, іконки, меню та вказівник» (Windows, Icons, Menus, Pointer – WIMP). Цей підхід передбачає використання віртуальних елементів керування, таких як вікна, кнопки, панелі меню та іконки, якими користувач може маніпулювати за допомогою фізичних пристроїв введення, наприклад, миші або тачпаду.

Основний компонент GUI – це вікно. Вікно являє собою прямокутну область на екрані, в якій відображаються інформація та графічні об'єкти. Користувач може переміщувати вікна, змінювати їхні розміри, закривати або відкривати нові.

Іконки є маленькими графічними символами, які можуть бути програмами, файлами або функціями. Вони зазвичай розміщені на робочому столі або в меню, та користувач може клікнути на них, щоб запустити відповідну програму або виконати певну дію.

Меню – це список опцій або команд, які користувач може вибрати для виконання певних дій. Меню може бути розташоване на панелі інструментів, у верхній частині вікна або на контекстному меню.

Покажчик, як правило, являє собою стрілку на екрані, якою користувач може переміщати та вибирати об'єкти на екрані. Він служить інструментом навігації та взаємодії з графічними елементами.

Коли користувач взаємодіє з графічним інтерфейсом, він може використовувати різні дії, такі як клікання, перетягування, натискання кнопок або введення тексту за допомогою клавіатури. Ці дії передаються до комп'ютера, який обробляє їх та виконує відповідні команди або відображає результати відповідно до взаємодії користувача [21].

## 5.2. Проєктування інтерфейсу системи

Графічний інтерфейс користувача (GUI) має свої переваги та недоліки, які варто враховувати при розробці та використанні.

Переваги графічного інтерфейсу користувача:

Інтуїтивність: GUI заснований на візуальних елементах, таких як іконки, кнопки та меню, що робить його доступним та легким для користувачів з різним рівнем технічних знань. Візуальна репрезентація допомагає користувачам швидко розуміти, які функції та опції доступні.

Зручність: GUI надає користувачам зручність взаємодії з комп'ютером. Вони можуть виконувати дії шляхом простого клацання мишею, перетягування об'єктів та використання клавіатури для швидкого введення даних.

Візуалізація: GUI дозволяє відображати інформацію за допомогою графічних елементів, що дозволяє користувачам більш ефективно сприймати та аналізувати дані. Графіки, діаграми, таблиці та інші візуальні елементи допомагають зрозуміти складну інформацію швидше та ефективніше.

Мультимедіа: GUI підтримує інтеграцію мультимедійних елементів, таких як зображення, відео та звук. Це дозволяє створювати більш привабливі та інтерактивні інтерфейси, що сприяє покращенню вражень користувачів.

Недоліки графічного інтерфейсу користувача:

Вимоги до апаратного забезпечення: GUI може вимагати відносно потужних комп'ютерних ресурсів для ефективної роботи. Велика кількість графічних ефектів та опцій може призводити до повільної роботи системи на застарілих або менш потужних комп'ютерах.

Навантаження на пам'ять: GUI використовує графічні елементи, які можуть займати більше місця в оперативній пам'яті порівняно з текстовим інтерфейсом. Це може призводити до збільшеного споживання пам'яті та зниження продуктивності системи.

Складність: Деякі GUI можуть бути вкрай складними та заплутаними через велику кількість функцій та опцій. Для користувачів з обмеженим досвідом або незнайомих з конкретною програмою це може бути перешкодою для ефективного використання інтерфейсу.

Залежність від обладнання: GUI зазвичай розробляється для конкретної платформи або операційної системи, що може призводити до обмежень у використанні на інших пристроях або платформах.

Усі ці фактори варто враховувати при проектуванні та використанні графічного інтерфейсу користувача, забезпечуючи його оптимальну ефективність та зручність для користувачів [21].

Для створення прототипу інтерфейсу було використано сервіс Figma.

Figma – це хмарний інструмент для проектування та створення прототипів, який дозволяє виконувати проекти в реальному часі.

Він поєднує в собі можливості інтерфейсу користувача (UI) і взаємодії з користувачем (UX), що дозволяє дизайнерам, розробникам та іншим членам команди співпрацювати в режимі реального часу.

Замість того, щоб зберігати файли локально, усі проекти зберігаються в хмарі, що робить їх легкими для спільного використання та доступними з будь-якого місця, де є підключення до Інтернету.

Figma підтримує багатокористувацький режим, тобто кілька людей можуть редагувати один і той самий файл одночасно, переглядаючи зміни в режимі реального часу, сприяючи кращій співпраці та спілкуванню між членами команди.

Крім того, Figma інтегрована з іншими інструментами та платформами, такими як Slack і Jira, що дозволяє легко інтегрувати процеси проектування в існуючі робочі процеси.

Цей інструмент підтримує прототипування з інтерактивними елементами, дозволяючи перевіряти ідеї та взаємодії перед їх реалізацією.

Інструмент пропонує багато плагінів і додаткових функцій, які розширюють його можливості та дозволяють налаштувати його відповідно до конкретних потреб вашого проекту чи команди.

Створений нами прототип інтерфейсу програмного продукту Показано на рисунках 5.1-5.6.

# 

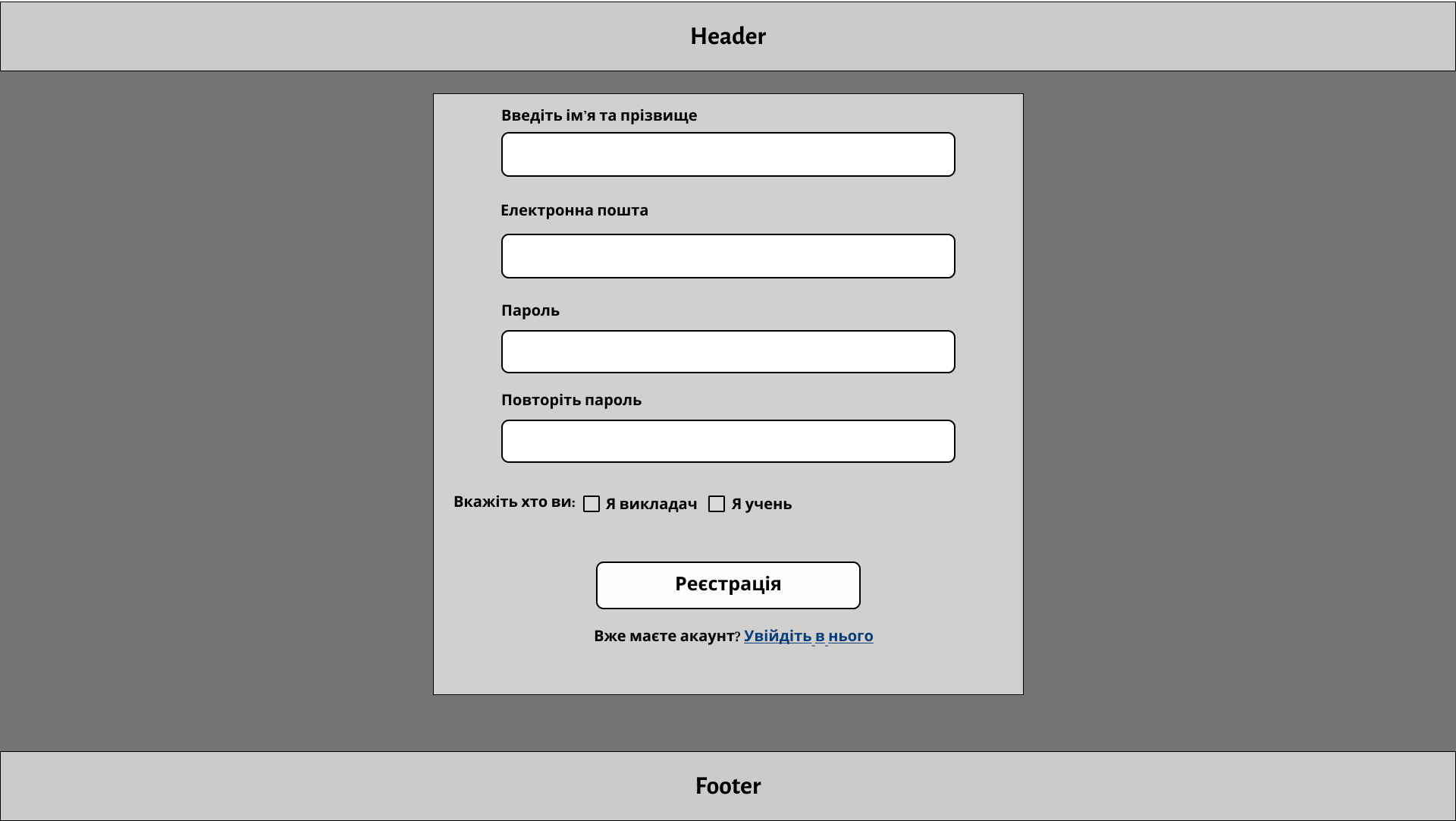


Рисунок 5.1 – Прототип інтерфейсу сторінки реєстрації

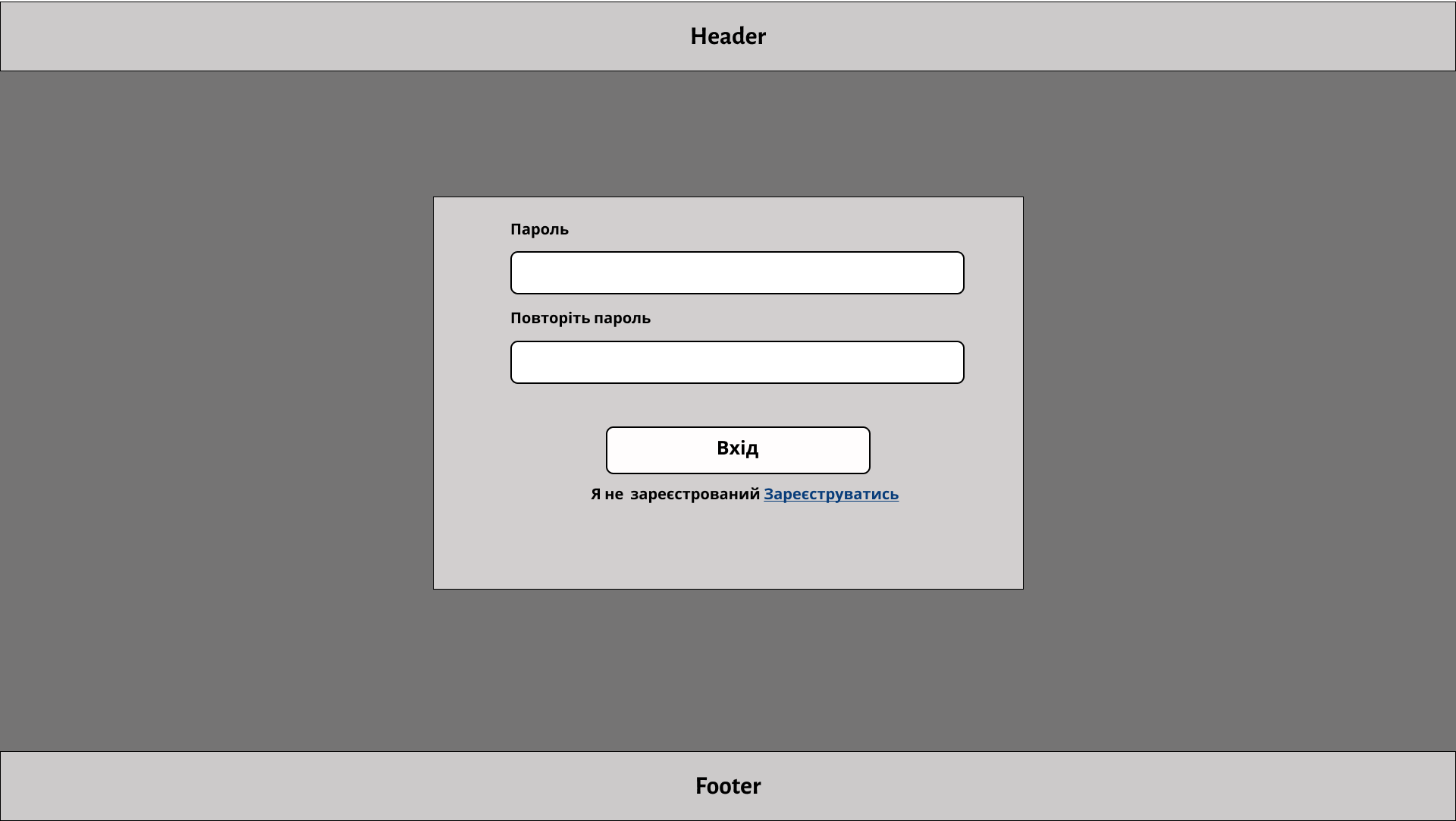


Рисунок 5.2 – Прототип інтерфейсу сторінки входу

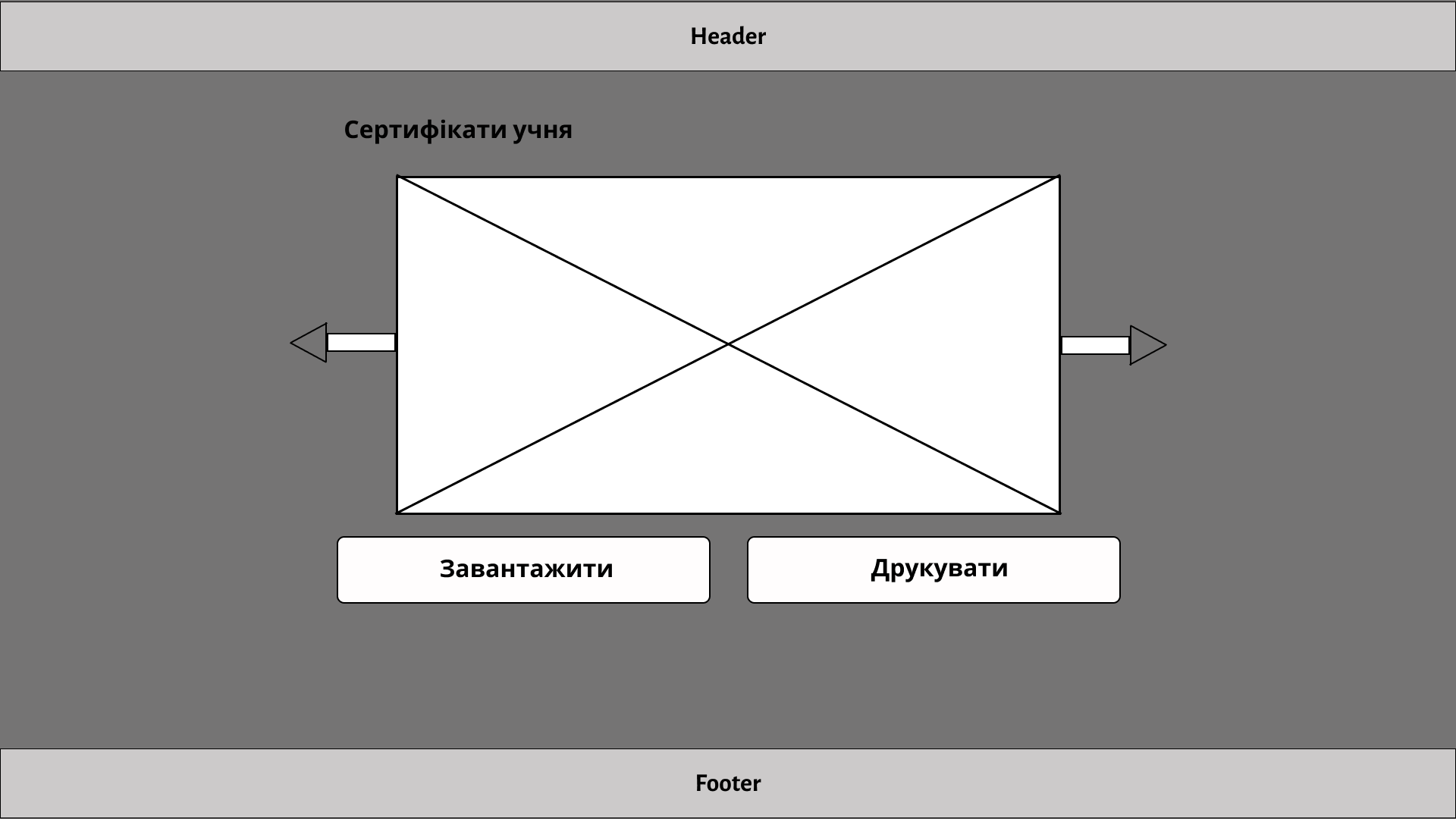


Рисунок 5.3 – Прототип інтерфейсу сторінки учня

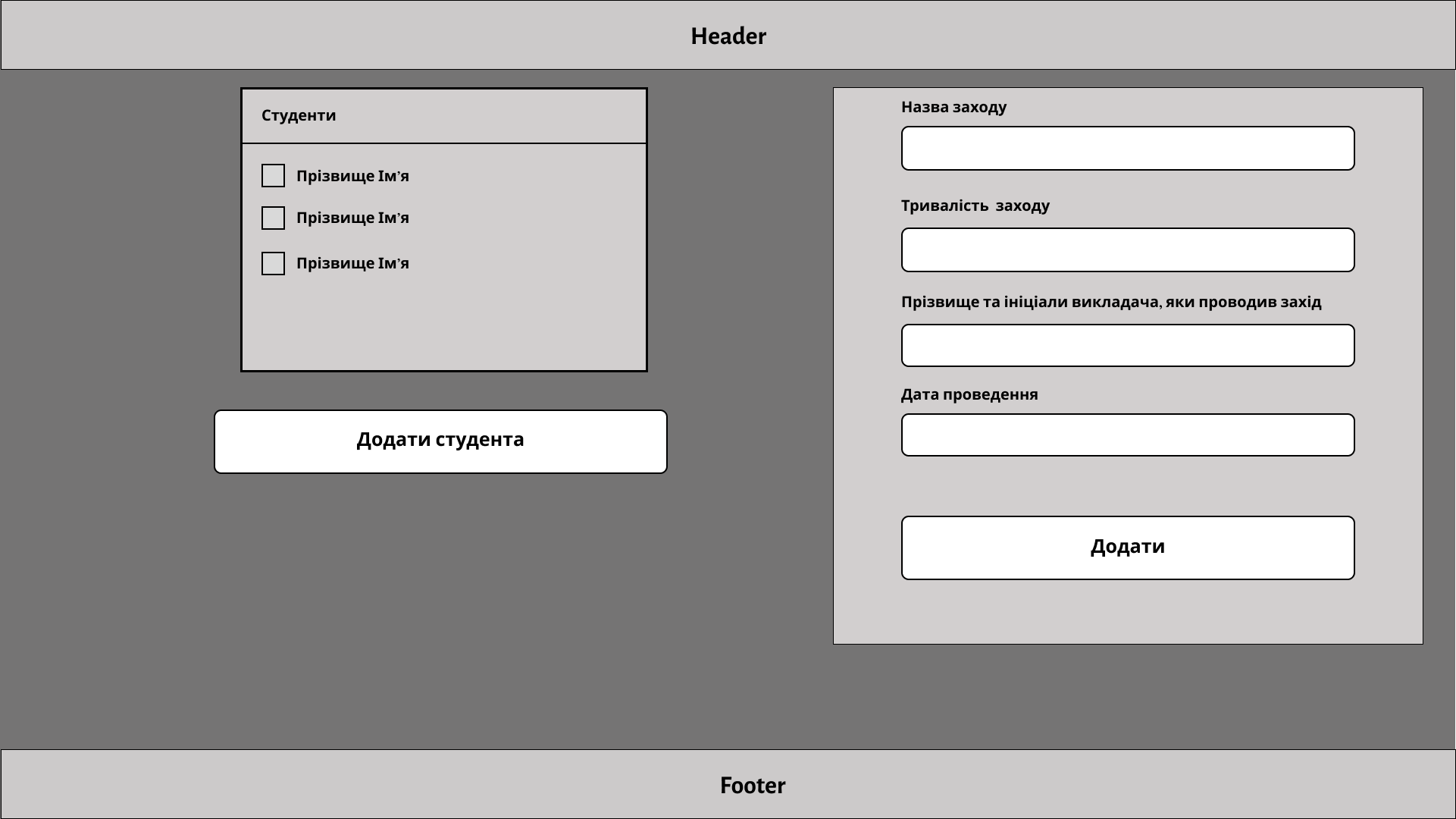


Рисунок 5.4 – Прототип інтерфейсу заповнення даних про сертифікат

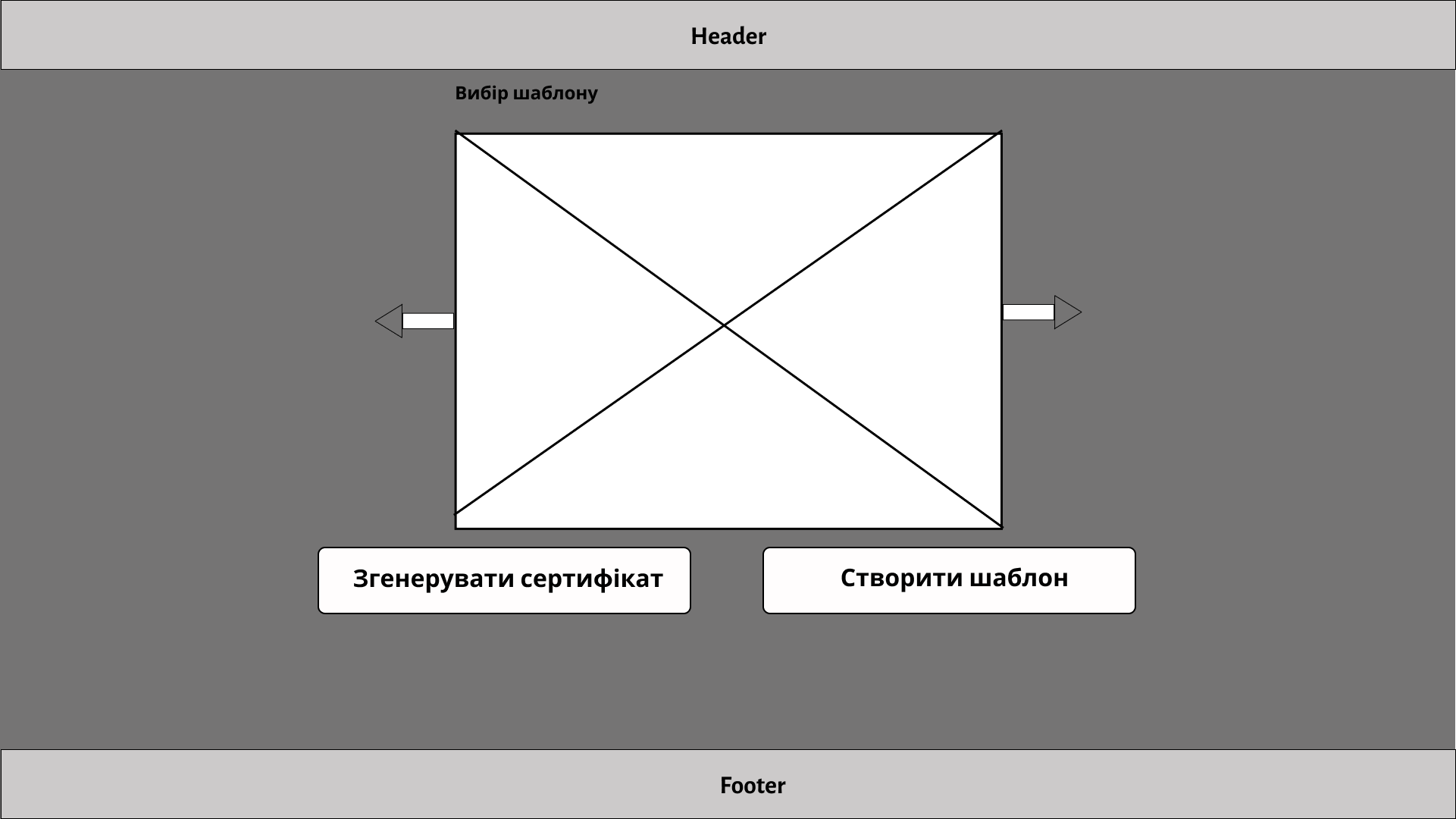


Рисунок 5.5 – Прототип інтерфейсу генерації сертифікату (Частина 1)

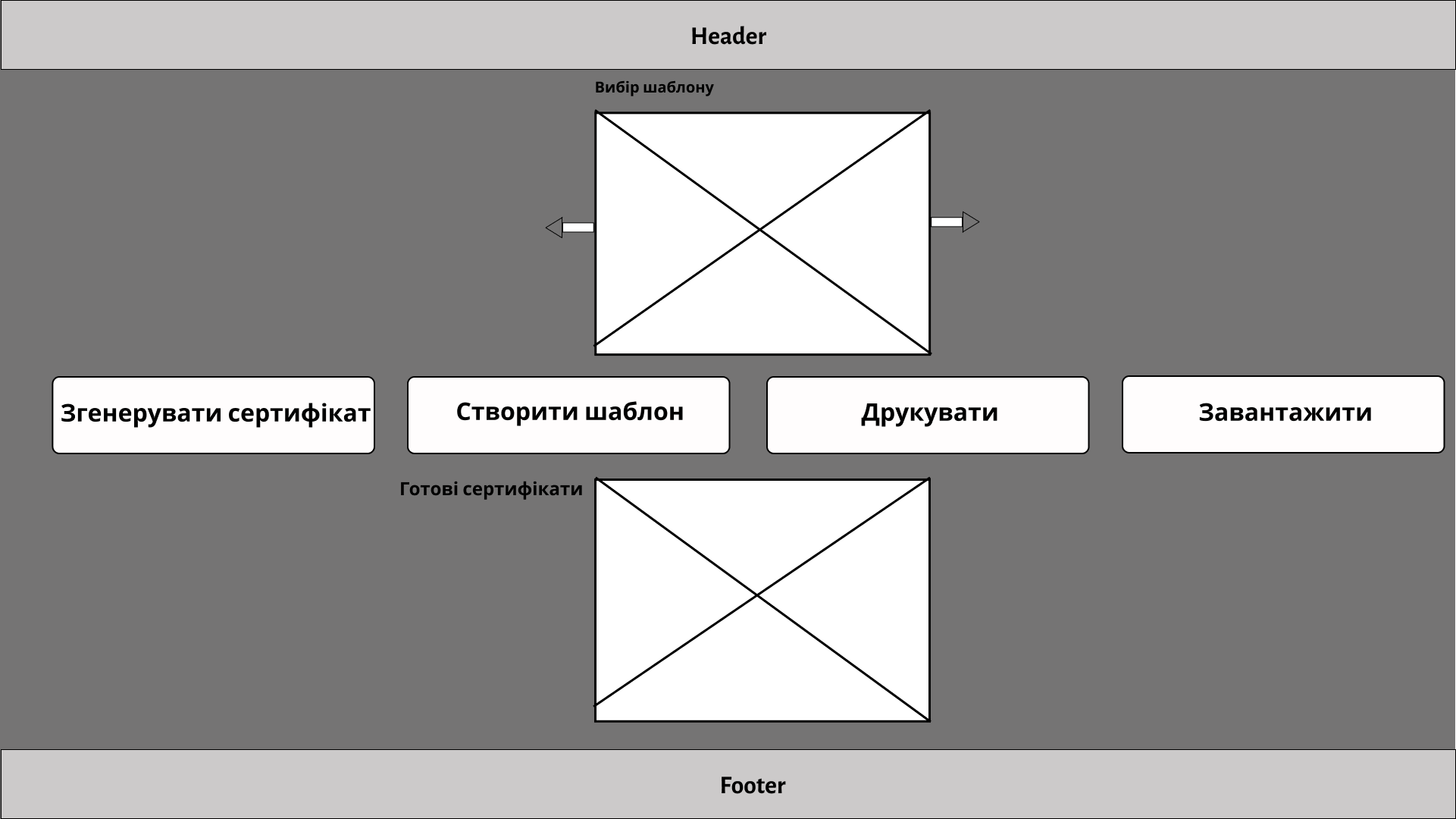


Рисунок 5.6 – Прототип інтерфейсу генерації сертифікату (Частина 2)

## 5.3. Технологічні аспекти проєктування інтерфейсу

Технологічне проєктування інтерфейсів є критичним етапом у розробці будь-якого програмного забезпечення чи системи. Його ціль – забезпечити ефективний, зручний та інтуїтивний спосіб взаємодії між користувачем і системою.

Першим кроком у технологічному проєктуванні інтерфейсу є збір та аналіз вимог до нього. Це включає в себе розуміння цільової аудиторії, їх потреб, можливостей та обмежень. Вимоги також повинні враховувати контекст використання системи, наприклад, чи буде інтерфейс використовуватися на мобільних пристроях, десктопах або інших платформах.

Вибір правильних технологій для створення інтерфейсу визначається великою мірою вимогами проєкту. Наприклад, для веб-інтерфейсів можуть використовуватися HTML, CSS та JavaScript, тоді як для мобільних додатків - Java або Kotlin для Android і Swift для iOS. При виборі технологій слід також враховувати фактори, такі як продуктивність, масштабованість, підтримка та безпека.

Методи проєктування інтерфейсу включають в себе використання кращих практик у дизайні, таких як створення зручних та естетичних інтерфейсів, які легко розуміються користувачами. До цього також входить увага до деталей, таких як розташування елементів керування, використання кольорів та шрифтів і реалізація анімації для покращення враження від користування.

Тестування інтерфейсу є критичним етапом у процесі розробки, оскільки воно дозволяє виявити та виправити помилки та недоліки, які можуть виникнути під час взаємодії користувача з системою. Тестування може включати в себе ручне та автоматизоване тестування, тестування з участю реальних користувачів та збір зворотного зв'язку для подальшого вдосконалення інтерфейсу.

Технологічне проєктування інтерфейсу вимагає уваги до деталей, розуміння потреб користувачів та використання відповідних технологій. Здійснення ефективного зв'язку між користувачем та системою є ключовим для створення успішного продукту, який задовольнить потреби своїх користувачів. Враховуючи вищезазначені аспекти, розробники можуть створити інтерфейси, які забезпечать зручну та приємну взаємодію з програмним забезпеченням чи системою.

**Висновки за розділом**

У розділі "Проектування інтерфейсу користувача" розглянуто ключові аспекти створення зручного, ефективного та привабливого інтерфейсу, який відповідає потребам та очікуванням користувачів.

Визначено поняття графічного інтерфейсу користувача (GUI) та його основні компоненти, такі як вікна, іконки, меню та вказівник. GUI базується на принципах ергономіки, спрямованих на створення інтуїтивно зрозумілої та простої взаємодії з комп'ютером.

Детально розглянуті переваги та недоліки графічного інтерфейсу користувача, від інтуїтивності та зручності до вимог до апаратного забезпечення та складності.

# 6. ОПИС ПРОЕКТНИХ РІШЕНЬ, ІНСТРУМЕНТІВ ТА ПІДХОДІВ ДО РОЗРОБКИ

## 6.1. Опис мови програмування

Для розробки дипломного проекту, було використано мову програмування JavaScript та її фреймворки.

JavaScript – це високорівнева скриптова мова програмування, яка широко використовується для створення інтерактивних застосунків і сайтів. JavaScript додає анімацію і спливаючі повідомлення та оновлює контент і змушує інтерфейс реагувати на наші дії. Наприклад, наводиш курсор на кнопку, а вона змінює колір (або зникає після натискання). Мова JavaScript робить сторінки функціональними.

JavaScript виконується в реальному часі в середовищі браузера, без необхідності попередньої компіляції. Розробник вносить зміни в код JavaScript і відразу бачить результат.

Тип змінних у JavaScript визначається автоматично під час виконання коду. Програмістам не потрібно присвоювати його заздалегідь, що спрощує роботу. Код JavaScript може бути виконаний на різних платформах і пристроях. Автоматичне збирання сміття полегшує керування пам’яттю, автоматично видаляючи об’єкти, що не використовуються.

JavaScript підтримує основні парадигми програмування. Це дає змогу створювати складні структури даних, покращувати організацію коду, застосовувати функціональні концепції та описувати точні кроки для досягнення мети.

За допомогою таких механізмів, як promise та async/await, JavaScript ефективно опрацьовує операції, що не потребують негайного завершення без блокування інших задач.

Вбудована підтримка DOM (Document Object Model) використовується для динамічної взаємодії з елементами вебсторінки, оброблення подій і внесення змін.

Навколо JavaScript сформувалася величезна спільнота розробників, які постійно створюють нові курси, підручники, фреймворки та бібліотеки JavaScript. Саме цей факт полегшує знаходження відповідей на запитання, які виникають у процесі роботи або навчання [22].

Фреймворк (framework) – це набір інструментів, бібліотек та правил, який використовується для створення програмних продуктів. Він зазвичай являє собою структуру, яка визначає, як компоненти програми повинні взаємодіяти між собою, які шаблони використовувати для створення інтерфейсів і які методи використовувати для роботи з базами даних та іншими зовнішніми ресурсами.

Фреймворк призначений для спрощення розробки програм, оскільки він надає готові рішення для поширених завдань. Це дозволяє програмістам зосередитися на створенні бізнес-логіки програми, а не витрачати час на написання коду для вирішення спільних завдань.

Крім того, він може бути спеціалізованим для певного типу програми. Наприклад, для веб-програм або мобільних застосунків. Він також може надавати інструменти управління проектом, такі як система контролю версій або інтеграція з іншими інструментами.

Основне завдання фреймворку **–** допомогтиправильно налаштувати робочі процеси та побудувати бізнес-логіку. Звичайно, фреймворки не є універсальним рішенням для всіх завдань і іноді може знадобитися написати додаток “з нуля”, щоб задовольнити унікальні вимоги. Однак, у більшості випадків використання фреймворків дозволяє значно прискорити та спростити процес розробки та створити більш надійний та ефективний додаток.

Люди, які не стикалися з фреймворками, часто плутають їх з бібліотеками. Але насправді це різні інструменти. Бібліотека містить у собі певний набір функцій, що дозволяє вирішити конкретне завдання з певної галузі. Наприклад, зустрічаються бібліотеки для роботи з датою або часом, HTTP-запитами тощо [23].

## 6.2. Використані технології та бібліотеки

Для розробки дипломного проекту були використані технології Node.js та Vue.js.

Node.js – це однопоточне кросплатформове середовище виконання з відкритим вихідним кодом і бібліотека, яка використовується для запуску вебдодатків, написаних на JavaScript, поза браузером клієнта. Простіше кажучи, Node.js – це програмне середовище, яке дозволяє запускати програми, написані мовою Javascript, поза браузером. Історично програми, написані на Javascript, на відміну від інших мов програмування, можна було запустити лише у браузерах, які мали спеціальний вбудований движок виконання коду цієї мови. Поза браузером Javascript, можна сказати, не працював. При розробці Node.js за основу було взято рушій виконання JavaScript під назвою V8, який був створений компанією Google і використовувався в браузері Google Chrome.

Оскільки після створення Node.js Javascript код можна запустити фактично в будь-якому середовищі, за допомогою цієї бібліотеки можна написати не лише фронтенд, а й серверну частину веб-програми. Простіше кажучи, це означає, що цілі сайти тепер можуть працювати з використанням єдиного «стека», що робить розробку та обслуговування набагато швидшими та легшими, дозволяючи зосередитися на досягненні бізнес-цілей проєкту. Node.js має відкритий код, тому працювати з ним можна абсолютно безкоштовно. Його і сьогодні продовжує розвивати та покращувати глобальна спільнота розробників. Важливо розуміти, що Node.js насправді не фреймворк і бібліотека, як у випадку з традиційним програмним забезпеченням, а середовище виконання. Воно є легким, гнучким і простим у розгортанні, а всі його функції допоможуть оптимізувати та прискорити вашу програму.

Важливою особливістю Node.js є асинхронний характер. Термін асинхронний означає, що сервер, створений з використанням Node.js, не повинен чекати, поки дані повернуться, при виконанні різних внутрішніх запитів. Водночас він також має неблокуючий введення-виведення. Це означає, що кілька різних процесів можуть виконуватись паралельно, не блокуючи один одного. Обидві ці властивості роблять Node.js вкрай швидкою і забезпечують кращий інтерфейс користувача.

У Node.js всі запити є однопоточними і збираються в циклі обробки подій (Event loop). Це означає, що всі програми виконуються в одному потоці, починаючи з отримання запиту і закінчуючи виконанням необхідного завдання та надсиланням відповіді клієнту назад. Ця функція Node.js запобігає повторному завантаженню запитів і скорочує час їх обробки, що робить його більш економічним у використанні.

Node.js використовується для різних програм. Нижче буде розглянуто кілька популярних варіантів використання:

* чати у реальному часі;

Завдяки своїй однопоточній асинхронній природі Node.js можна використовувати для обробки повідомлень у реальному часі. Він легко масштабується і часто використовується для створення чат-ботів. Node.js також спрощує створення додаткових функцій чату, які розраховані на багато користувачів чату і push-сповіщення.

* інтернет речей (концепція IoT);

Програми IoT зазвичай містять кілька датчиків, оскільки часто відправляють невеликі фрагменти даних, які можуть накопичуватися при великій кількості запитів. Node.js в такому випадку – гарний вибір, тому що він може швидко обробляти ці одночасні запити.

* потокова передача даних;

Такі компанії, як Netflix, працюють із Node.js для потокової передачі. В основному це пов’язано з тим, що Node.js є легким і швидким. Крім того, Node.js надає власний потоковий API. Ці потоки дозволяють користувачам передавати дані швидше, ніж при використанні інших технологій.

* складні односторінкові програми (SPA);

У SPA вся програма завантажується один раз на одній HTML-сторінці. Зазвичай це означає, що у фоновому режимі виконується кілька запитів зміни компонентів на екрані користувача. Тут на допомогу приходить цикл обробки подій Node.js, оскільки він обробляє подібні запити неблокуючим чином.

* програми на основі REST API.

JavaScript може використовуватись як у зовнішньому, так і внутрішньому інтерфейсі сайтів. Таким чином, сервер може легко взаємодіяти із зовнішнім інтерфейсом через REST API, використовуючи Node.js. Node.js також пропонує такі пакети, як Express.js і Koa, які дозволяють створити вебсервер з нуля ще простіше і швидше [24].

Загалом Node.js має великі перспективи для майбутнього розвитку та використання. Його швидкість, гнучкість і масштабованість роблять його привабливим вибором для розробників і компаній, які прагнуть створювати інноваційні програми та послуги. Сьогодні сотні успішних проектів із тривалим життєвим циклом працюють над Node.js, тому він точно не зникне найближчим часом.

Vue – це фреймворк, який працює на JavaScript, створений для розробки користувацьких інтерфейсів. Він працює на базі звичайного HTML, CSS та JavaScript, з можливостями декларативно програмувати користувацькі інтерфейси будь-якої складності на основі компонентів.

Vue розширює стандартний HTML шаблонним синтаксисом, який дозволяє декларативно задавати структуру HTML на основі стану описаного у JavaScript. Він автоматично відстежує зміни стану описаного у JavaScript і з максимальною ефективністю оновлює DOM, коли відбуваються зміни.

Vue – це фреймворк та екосистема, яка охоплює більшість функцій, необхідних для розробки інтерфейсу. Але веб додатки надзвичайно різноманітні – речі, які створюються в рамках веб додатків, можуть кардинально відрізнятися за формою та масштабом. Зважаючи на це, Vue розроблено таким чином, щоб він міг бути гнучким і адаптивним. Залежно від випадку та задач, Vue можна використовувати різними способами:

* розширення статичного HTML;
* вбудовувати як веб-компонент на будь-яку сторінку;
* створення одно-сторінкового додатку (SPA);
* додаток з рендерингом на стороні серверу (SSR);
* генерація статичного додатку (SSG);
* створення десктопних, мобільних, WebGL додатків.

У більшості проєктів Vue, що використовують інструменти збірки та потребують етапу збірки, створюються компоненти Vue, використовуючи HTML-подібний формат файлу під назвою Single-File Component (також відомий як файли \*.vue, скорочено SFC [25]). Vue SFC, як випливає з назви, інкапсулює логіку компонента (JavaScript), шаблон (HTML) і стилі (CSS) в одному файлі. SFC є однією з основних функцій Vue і рекомендованим способом створення компонентів Vue, якщо задача вимагає використання інструментів збірки [26].

У розробці дипломного проекту використовувалися різноманітні бібліотеки для реалізації різних функціональних можливостей.

Для налаштування серверної частини використовувався Express.js.

Express.js – це JavaScript web-фреймворк для програмної платформи Node.js. Швидкий, гнучкий і лаконічний каркас вважається стандартом для розробки web-додатків в цьому середовищі. Express використовують для створення надійних API (наборів функцій, що організовують взаємодію між компонентами програми, а також зовнішні сервіси, наприклад, для онлайн-оплати).

Express – це безліч додаткових плагінів, що полегшують розробку, і дозволяють простіше вирішувати нестандартні завдання. Його можна використовувати для бекенд, зокрема з БД MongoDB, а також фронтенда разом з бібліотеками Vue.js, React або AngularJS [27].

Mongoose – це ODM (Object Document Mapper - об'єктно-документний відображувач). Це означає, що Mongoose дозволяє визначати об'єкти зі строго-типізованою схемою, що відповідає документу MongoDB.

Mongoose дає величезний набір функціональних можливостей для створення та роботи зі схемами. Наразі Mongoose має вісім SchemaTypes (типи даних схеми), котрі може мати властивість, що зберігається до MongoDB. Ці типи наступні:

* string;
* number;
* date;
* buffer;
* boolean;
* mixed;
* objectId (унікальний ідентифікатор об'єкта, первинний ключ, \_id);
* array.

Для кожного типу даних можна:

* зазначити значення за налаштуванням;
* зазначити функцію користувача для перевірки даних;
* зазначити, що поле необхідно заповнити;
* зазначити get-функцію (геттер), яка дозволяє вам проводити маніпуляції над даними до їх повернення у вигляді об'єкта;
* зазначити set-функцію (\* сеттер), яка дозволяє вам проводити маніпуляції над даними перед їх зберіганням до бази даних;
* визначити індекси для більш швидкого отримання даних.

Окрім цих спільних можливостей для деяких типів даних також можна налаштувати особливості зберігання та отримання даних із бази даних. Наприклад, для типу даних String можна зазначити наступні додаткові опції:

* конвертація даних до нижнього регістру;
* конвертація даних до верхнього регістру;
* обрізання даних перед зберіганням;
* визначення регулярного виразу, який дозволяє в процесі перевірки даних обмежити дозволені для зберігання варіанти даних;
* визначення переліку, який дозволяє встановити список припустимих рядків.

Для властивостей типу Number і Date можна зазначити мінімально та максимально допустиме значення.

Більшість з восьми допустних типів даних мають бути вам добре знайомі. Однак, деякі (Buffer, Mixed, ObjectId та Array) можуть викликати труднощі.

Тип даних Buffer дозволяє вам зберігати двійкові дані. Типовим прикладом двійкових даних може бути зображення чи закодований файл, наприклад, документ у PDF-форматі (формат документа що передається).

Тип даних Mixed використовується для перетворення властивості в "неперебірливе" поле (поле, в якому припустимі дані будь-якого типу). Так само, як багато розробників використовують MongoDB для різних цілей, у цьому полі можна зберігати дані різного типу, оскільки відсутня визначена структура. Обачливо використовуйте цей тип даних, оскільки він обмежує можливості, надавані Mongoose, наприклад, перевірку даних та відстежування змін сутності для автоматичного оновлення властивості при зберіганні.

Тип даних ObjectId використовується звичайно для визначення посилання на інший документ у вашій базі даних. Наприклад, якщо б у вас була колекція книг та авторів, документ книги міг би містити властивість ObjectId, що посилається на визначеного автора документа.

Тип даних Array дозволяє вам зберігати JavaScript-подібні масиви. Завдяки цьому типу даних ви можете виконувати над даними типові JavaScript операції над масивами, наприклад, push, pop, shift, slice тощо [28].

Multer **–** цепрограмне забезпечення для керування завантаженням файлів у веб-додатках Node.js, особливо тих, що базуються на фреймворку Express.

Використовуючи Multer, ви можете легко приймати файли, надіслані на сервер через запити форм HTTP (наприклад, форми завантаження файлів).

Це дозволяє встановити різні параметри, такі як максимальний розмір файлу, дозволені типи файлів тощо.

Multer автоматично обробляє завантажені файли та робить їх доступними у вашій веб-програмі, щоб ви могли виконувати їх подальшу обробку, наприклад зберігати їх на сервері або обробляти для використання пізніше.

NodeHtmlToImage – це бібліотека Node.js, яка дозволяє генерувати зображення з HTML-коду. За допомогою цієї бібліотеки можна створювати зображення з вмістом HTML сторінок або елементів та використовувати їх у вашому додатку. Зазвичай це використовується для автоматизації створення зображень із статичного контенту, як, наприклад, створення знімків електронної пошти, генерація ескізів веб-сторінок, створення мініатюр зображень тощо.

Ця бібліотека приймає HTML-розмітку як вхідні дані. Також вона надає різні налаштування для контролю розміру, формату та інших атрибутів зображення, а також можливість зберігати зображення у вигляді файлу або повертати у вигляді буфера.

PDFKit – це бібліотека для створення PDF-документів у середовищі Node.js. Вона спрощує процес створення складних, багатосторінкових документів, які можна використовувати для друку або інших цілей. API PDFKit включає як базові операції, так і високорівневі абстракції, що дозволяють створювати документи з різними рівнями складності.

API PDFKit розроблено з урахуванням простоти використання, тому створення навіть складних документів може бути настільки ж простим, як кілька викликів функцій. Можна легко додавати сторінки, текст, зображення, таблиці та інші елементи до вашого PDF-документа, налаштовуючи їх розміри, положення та стилізацію. Це дозволяє створювати документи з високою точністю та професійним виглядом за допомогою Node.js.

Bcryptjs – це бібліотека для хешування паролів у Node.js. Вона базується на алгоритмі хешування bcrypt, який є одним із найбільш надійних способів зберігання паролів у веб-додатках. Основна мета використання bcrypt або bcryptjs полягає у тому, щоб уникнути зберігання паролів у відкритому вигляді у базі даних.

Основні функції Bcryptjs включають можливість створення хешів паролів, яка робить їх важкими до розшифрування навіть у випадку, якщо база даних буде скомпрометована. Крім того, вона надає методи для порівняння введеного користувачем пароля з хешем пароля, збереженим у базі даних, для перевірки правильності введеного пароля.

Бібліотека Bcryptjs зазвичай використовується у веб-додатках для безпечного зберігання та перевірки паролів користувачів, що дозволяє забезпечити високий рівень безпеки при автентифікації.

З боку Vue було використано Vuetify.

Vuetify – це бібліотека компонентів користувацького інтерфейсу для Vue.js, яка дозволяє створювати красиві та функціональні веб-додатки за допомогою готових компонентів, включаючи елементи дизайну, анімації, макети та багато іншого. Вона базується на принципах Material Design від Google та надає широкий спектр готових компонентів для швидкої розробки веб-додатків.

Компоненти Vuetify легко налаштовувати та адаптувати під конкретні потреби проекту. Вони пропонують безліч опцій та властивостей для зміни зовнішнього вигляду та поведінки.

Усі компоненти Vuetify адаптивні та можуть легко масштабуватися для різних пристроїв та роздільної здатності екрану.

Бібліотека підтримує створення користувацьких тем, що дозволяє легко змінювати кольори, шрифти та інші стилі компонентів відповідно до дизайну вашого додатку.

Vuetify повністю інтегрується з екосистемою Vue.js, що дозволяє використовувати його компоненти та функціональність у будь-якому проекті Vue.js.

Ця бібліотека стала популярним інструментом для розробки веб-додатків на Vue.js завдяки своїй простоті використання, гнучкості та красивому дизайну, що робить процес розробки веб-додатків більш ефективним та приємним [29].

## 6.3. Середовище розробки

Весь процес розробки відбувався в IDE (Integrated Development Environment) Visual Studio Code [30].

Visual Studio Code (VS Code) – це потужний та популярний текстовий редактор, розроблений компанією Microsoft. Він став відомим завдяки своїм багатофункціональним можливостям, легкості використання та відкритому коду.

VS Code підтримує широкий спектр мов програмування, включаючи JavaScript, TypeScript, Python, Java, C++, HTML, CSS та багато інших.

Редактор має потужні функції, такі як підсвічування синтаксису, автодоповнення, вбудований дебагер, контроль версій і багато іншого.

VS Code має широкий вибір розширень, які дозволяють налаштовувати редактор під свої потреби. Це включає теми оформлення, розширення для підтримки конкретних мов програмування, інструменти для роботи з git та багато іншого.

VS Code інтегрується з іншими інструментами розробки, такими як Git, Docker, Kubernetes, та інші, що дозволяє зручно працювати з різними аспектами розробки програмного забезпечення.

VS Code підтримується на різних операційних системах, включаючи Windows, macOS та Linux, що робить його доступним для широкого кола розробників.

Розробники можуть легко працювати з хмарними сервісами, такими як Azure, AWS, та іншими, завдяки інтеграції VS Code з цими платформами.

VS Code має активне співтовариство користувачів та розробників, яке надає підтримку, допомогу та поширення розширень та корисних порад.

Visual Studio Code є безкоштовним та має відкритий вихідний код, що означає, що його можна використовувати, модифікувати та розповсюджувати безкоштовно. Інтерфейс редактора є простим та інтуїтивно зрозумілим (Рисунок 6.1), його виконано у типовому для продуктів Microsoft стилі.

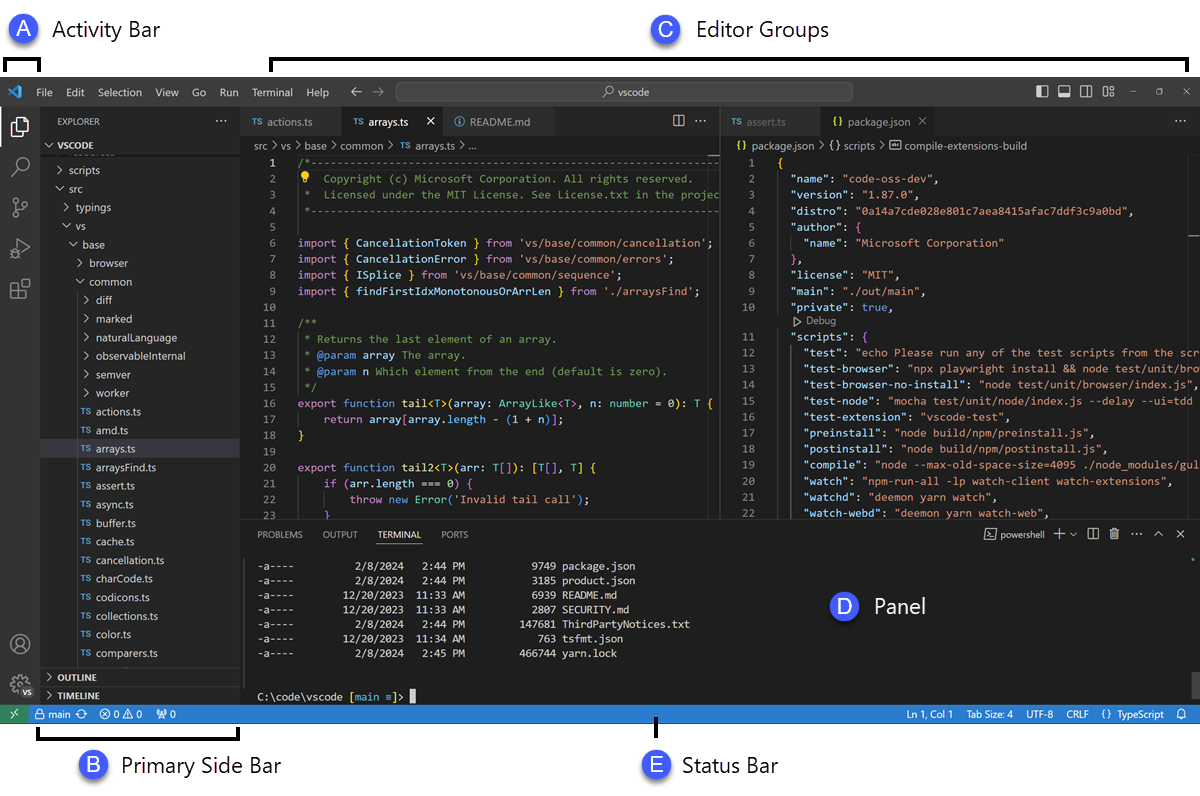


Рисунок 6.1 – Загальний вигляд Visual Studio Code

## 6.4. Методологія розробки

У процесі розробки дипломного проекту, було прийнято рішення використати методологію Scrum для розробки веб-застосунку "Генератор сертифікатів".

На сьогоднішній день переважна більшість проєктів використовують гнучку філософію Agile. А метод Scrum є однією із найпоширеніших реалізацій Agile-підходу.

Скрам базується на головних ідеях Agile :

* учасники процесу та їх комунікація важливіші за інструменти і сам процес;
* працюючий продукт важливіший за докладну документацію;
* постійна взаємодія із замовником важливіша за узгодження договору;
* готовність змінювати і модифікувати продукт і робочий процес важливіший за строге дотримання плану.

Головна мета цих принципів – забезпечити на виході продукт, який максимально відповідатиме потребам та очікуванням замовника. Без гнучкої методології управління проєктом це завдання дуже складно вирішити. Справа в тому, що більшість продуктів створюються не за один день чи тиждень. Робота може тривати місяці та роки. На ринку, що постійно розвивається, вимоги до товару змінюються вже у процесі. Крім того, замовник на початковому етапі часто не враховує або об'єктивно не може передбачити всіх нюансів, а потім у нього періодично виникає гостре бажання вносити "правки".

Scrum став революційним методом, який, з одного боку, спрощує координацію роботи та оцінку ефективності команди, а з іншого – дає можливість швидко вносити зміни та адаптувати процес до нових актуальних завдань.

Методологія Scrum передбачає роботу короткими відрізками – спринтами (Рисунок 6.2). Тривалість мінімального робочого циклу – від 1 до 4 тижнів. Як правило, результат спринту – це робочий продукт з очікуваним приростом функціоналу. Ідея полягає в тому, щоб кожен відрізок приносив результат, який можна продемонструвати замовнику. Після схвалення та коментаря від клієнта команда приступає до наступного спринту і весь процес повторюється. Через таку циклічність роботи Scrum називають ітеративним методом.

Поділ всього обсягу завдань на невеликі робочі цикли дає змогу оперативно виявляти проблеми та непорозуміння із замовником. Постійний зворотний зв'язок допомагає легко вносити зміни на кожному етапі. Scrum-підхід робить проєкт максимально відповідним очікуванням клієнта та кінцевого користувача.

Основою методології Scrum є три складові:

* ролі;
* заходи;
* артефакти [31].

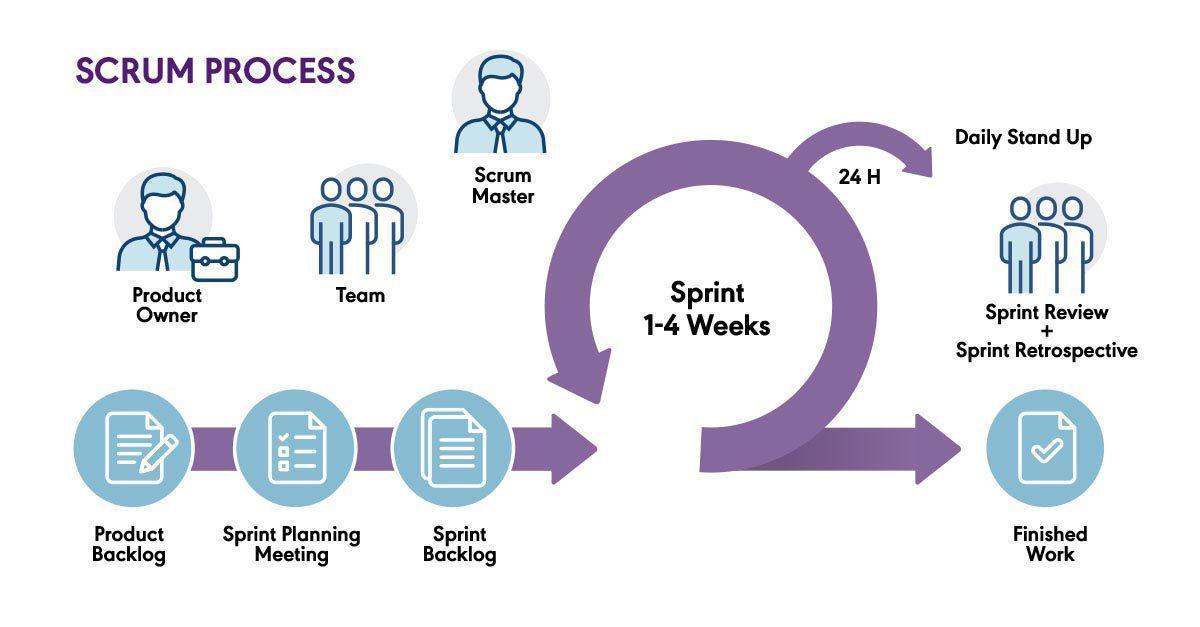


Рисунок 6.2 – Методологія Scrum

**Висновки за розділом**

В розділі було досліджено використання мови програмування JavaScript та її фреймворків для реалізації дипломного проекту.

Також у розділі описано використання бібліотек, таких як Express.js і Mongoose, для налаштування серверної частини та роботи з базою даних MongoDB. Використання фреймворків, таких як Node.js і Vue.js, дозволило прискорити та спростити процес розробки, забезпечивши готові рішення для поширених завдань та структури, що описують взаємодію компонентів програми.

У результаті цього дослідження можна зробити висновок, що JavaScript та його фреймворки, такі як Node.js і Vue.js, разом з використанням додаткових бібліотек, надають широкі можливості для розробки різноманітних веб-додатків з високим рівнем інтерактивності та функціональності.

# 7. МЕТОДИ ТА РЕЗУЛЬТАТИ ТЕСТУВАННЯ

## 7.1. Поняття про тестування, види тестів

Тестування програмного забезпечення – це ітеративний процес, який передбачає виконання різноманітних тестів і аналіз результатів для виявлення помилок і дефектів у програмі. Основна мета тестування – переконатися, що програмне забезпечення працює належним чином, відповідає всім вимогам і очікуванням користувачів, а також працює надійно та безпечно. Цей процес допомагає виявити й усунути потенційні проблеми до запуску програми, покращуючи якість продукту та задовольняючи потреби користувачів.

Тестування програмного забезпечення може проводитися як вручну, так і за допомогою автоматизації. Обидва підходи мають свої переваги та недоліки, які варто враховувати при плануванні тестових стратегій. Розглянемо коротко основні аспекти кожного з них.

Ручне тестування (Manual testing) – це набагато більше, ніж просто виконання тестів. Це спосіб пошуку вразливостей та свідомого виявлення непомічених раніше дефектів, у якому людський фактор є основним додатковим засобом для впевненості в якості продукту. У світі багаторівневого програмного забезпечення ручне тестування є важливим етапом, який розкриває те, що запрограмовані тести не можуть. Це процес, у якому відображається мистецтво та вміння виявляти помилки, які інші методи можливо пропустили б.

Автоматичне тестування (Automation testing) – це процес виконання тестів та перевірки результатів за допомогою автоматизованих засобів, таких як тести без користувача, скрипти тестування, програмні платформи тощо. Цей метод стає все більш важливим у сучасній розробці програмного забезпечення, оскільки дозволяє ефективно автоматизувати та прискорити процес тестування, зменшуючи людський вплив та забезпечуючи послідовність та швидкість виконання тестів.

Існує кілька видів тестування програмного забезпечення, які включають у себе різноманітні методики та підходи:

* функціональне тестування: це перевірка окремих функцій або можливостей програми, щоб впевнитися, що вони працюють правильно;
* модульне тестування (Unit Testing): це тестування окремих модулів та компонентів програмного забезпечення, щоб переконатися у їх коректності;
* інтеграційне тестування (Integration Testing): це перевірка взаємодії між різними модулями або компонентами програми;
* системне тестування (System Testing): це оцінка всієї системи як одного цілого згідно з вимогами та очікуваннями користувачів;
* тестування продуктивності (Performance Testing): це вимірювання реакції системи на різні навантаження та обсяги даних;
* тестування стійкості (Stress Testing): це випробування системи під екстремальними навантаженнями для визначення її меж і стійкості;
* тестування масштабованості (Scalability Testing): це визначення можливості системи масштабуватися з ростом обсягу даних або користувачів;
* тестування безпеки: це перевірка системи на вразливості та контроль доступу до неї;
* тестування регресії: це перевірка того, чи не з'явилися нові помилки після внесення змін у програмне забезпечення.

Крім описаних вище, існують і інші види тестування, такі як тестування сумісності з різними пристроями, тестування відмов для оцінки надійності системи, та тестування користувацького інтерфейсу для перевірки зручності та інтуїтивності взаємодії з програмою. Кожен тип тестування має свої особливості та вимоги, і вибір конкретного виду залежить від потреб вашого проєкту [32].

## 7.2. Обрані види тестів та результати тестування

Для тестування Node.js у проекті було використано фреймворк для тестування Jest [33].

Jest – це потужна платформа тестування JavaScript, створена Facebook, яка набула популярності завдяки своїй простоті, швидкості та функціям.

Jest призначений для тестування клієнтських і внутрішніх програм, забезпечуючи універсальність і гнучкість для розробників.

Головною перевагою Jest є простота використання. Цей фреймворк автоматично знаходить і запускає всі тестові файли у проекті, значно спрощуючи налаштування.

Не потрібно витрачати час на складне налаштування, оскільки Jest автоматично налаштовується та готовий до використання відразу після встановлення. Це дозволяє швидко почати писати тести та зосередитися на логіці коду.

Jest підтримує різні типи тестування, включаючи модульне тестування, інтеграційне тестування та наскрізне тестування. Це дозволяє тестувати окремі функції, взаємодію між компонентами та загальну поведінку програми.

Однією з чудових функцій Jest є тестування знімків. Тестування миттєвих знімків дозволяє зберігати стан елементів інтерфейсу користувача як миттєві знімки та порівнювати їх із збереженими зразками під час наступних тестових прогонів. Це допомагає виявляти небажані зміни у зовнішньому вигляді програми та забезпечує стабільність інтерфейсу користувача.

Jest добре інтегрується з такими популярними фреймворками та бібліотеками, як React, Vue та Angular. Це надає можливість легко писати тести для компонентів UI, перевіряючи їх коректність і стабільність.

Jest також підтримує модульні системи, такі як модулі ES6 і CommonJS, що робить його сумісним з більшістю сучасних проектів.

Крім того, Jest підтримує паралельне виконання тестів, що значно прискорює час тестування для великих проектів.

Цей фреймворк автоматично розподіляє тестові завдання між кількома працівниками, використовуючи всі доступні ядра ЦП, що дозволяє скоротити час, витрачений на очікування результатів тесту.

Іншою важливою особливістю Jest є можливість створення покриття коду. Покриття коду дозволяє відстежувати, які частини коду були перевірені, а які ні.

Це допомагає виявити не повністю охоплені частини коду та покращує загальну якість тестування.

Одним словом, Jest – це потужний і зручний інструмент для автоматизації тестування програм JavaScript.

**Висновки за розділом**

Простота використання Jest, його гнучкість, підтримка різних типів тестування та можливості тестування знімків роблять його ідеальним вибором для забезпечення якості коду та прискорення процесу розробки.

Результати тестування розробленого застосунку показані на Рисунках 7.1-7.2.

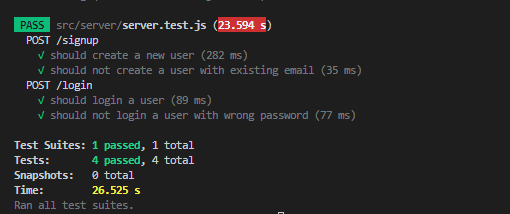


Рисунок 7.1 – Тестування коду Node.js на реєстрацію та логін користувача

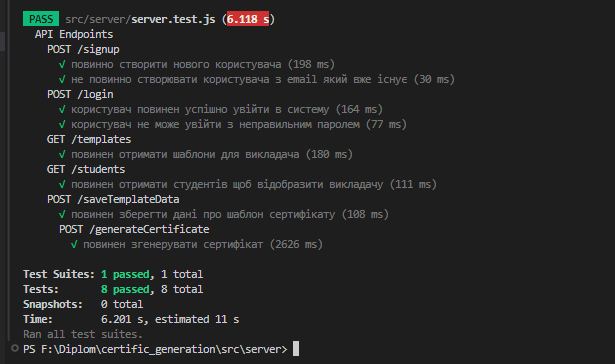


Рисунок 7.2 – Тестування всіх основних функції коду на Node.js

# ВИСНОВКИ

В роботі було поставлено завдання розробити веб-застосунок для генерації сертифікатів на підтвердження навчальних досягнень, який мав спростити та автоматизувати процес створення персоналізованих сертифікатів для учасників подій, семінарів, курсів та конференцій. Слід відзначити, що ця мета була успішно досягнута.

Процес розробки включав в себе детальний огляд предметної області, аналіз аналогічного програмного забезпечення , визначення вимог, побудову UML діаграм , розробку програмного продукту, тестування, підготовку до розгортання на сервері, а також опис проектних рішень та використаних технологій.

Веб-застосунок був побудований з використанням Node.js та Vue.js. Він складається з ряду підсистем, включаючи інтерфейс користувача, серверну частину, базу даних та логіку генерації сертифікатів. Вихідні коди продукту розміщені в публічному GitHub-репозиторії, фрагмент коду також представлено у Додатку А. Знімки екрану готового продукту виміщено у додатку Б.

Отже, можна зазначити, що розроблений веб-застосунок відповідає поставленим вимогам, забезпечуючи автоматизацію та оптимізацію процесу створення сертифікатів. Його застосування може значно полегшити роботу організаторів подій та забезпечити зручний доступ до сертифікатів для учасників.

Загалом, розроблений веб-застосунок є чудовим прикладом використання веб-технологій в освітній галузі. Як вже було зазначено вище, вихідні коди продукту є публічно доступними, тож будь-який освітній заклад може скористатись ними аби створити власне рішення для генерації сертифікатів, яке відповідає їх специфічним вимогам та потребам.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Лавріщева К. М. ПРОГРАМНА ІНЖЕНЕРІЯ: підручник. Київ, 2008. 319 с.
2. Тлумачення з "Словника української мови" // СловникUA. 2024. URL: https://slovnyk.ua/index.php?swrd=подія (дата звернення: 19.02.2024).
3. СЕМІНАР ЯК МЕТОД ПІДВИЩЕННЯ КВАЛІФІКАЦІЇ // UBI Конференц Холл. 2017. URL: https://ubi-hall.com.ua/uk/2017/09/21/seminar-iak-metod-  
   pidviennia-kvalifik/ (дата звернення: 19.02.2024).
4. Що таке онлайн-курс? // Самоосвіта. 2024. URL: https://samoosvita.in.ua/  
   scho-take-onlayn-kurs/ (дата звернення: 19.02.2024).
5. Тренінг як одна з форм підвищення кваліфікації бібліотечного персоналу: посібник. Львів: Регіональний тренінговий центр Львівської обласної універсальної наукової бібліотеки, 2010. 24 с.
6. ЩО ТАКЕ ВЕБ ДОДАТОК? РІЗНИЦЯ МІЖ САЙТОМ, ВЕБ-ДОДАТКОМ, SPA І PWA // WEBCASE. URL: https://webcase.com.ua/uk/blog/  
   cho-takoe-web-prilozhenie-vse-vidy/#f2 (дата звернення: 19.02.2024).
7. Free Certificate Maker for Custom Certificates: [Веб-сайт]. URL: https://www.visme.co/certificate-maker/ (дата звернення: 24.05.2024).
8. Free Online Certificate Maker: [Веб-сайт]. URL: https://www.canva.com/  
   create/certificates/ (дата звернення: 24.05.2024).
9. Online Certificate Maker: [Веб-сайт]. URL: https://www.bannerbear.com/  
   generators/free-online-certificate-generator/ (дата звернення: 24.05.2024).
10. A Glossary of Requirements Engineering Terminology . Zurich. 130 с.
11. Моделювання бізнес-процесів та архітектура програмного забезпечення: опорний конспект лекцій. Тернопіль, 2015. 105 с.
12. Функціональність - це: [Веб-сайт]. 2016. URL: https://ua.waykun.com  
    /articles/funkcionalnist-ce.php (дата звернення: 10.03.2024).
13. СИСТЕМНА ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: конспект лекцій. Київ, 2020. 45 с.
14. Архітектура веб додатків // Medium. 2023. URL: https://medium.com/@IvanZmerzlyi/архітектура-веб-додатків-ca4c82f75bcf (дата звернення: 05.02.2024).
15. Що краще моноліт чи мікросервіси? Як обрати архітектуру проєкту? // IAMPM. 2024. URL: https://iampm.club/ua/blog/shho-krashhe-monolit  
    -chi-mikroservisi-yak-obrati-arhitekturu-projektu/ (дата звернення: 05.02.2024).
16. Архітектура клієнт-сервер // Електронна бібліотека ВСП «Павлоградський фаховий коледж НТУ «Дніпровська політехніка» Шукати:. URL: http://inter.ptngu.com/kompyuterni-merezhi/arhitektura-kliyent-server (дата звернення: 02.05.2024).
17. Класи в програмуванні: занурення в об’єктно-орієнтоване програмування // Foxminded. URL: https://foxminded.ua/klasy-v-prohramuvanni/ (дата звернення: 03.05.2024).
18. MongoDB Підручник [Електронний ресурс], Режим доступу: https://w3schoolsua.github.io/mongodb/index.html#gsc.tab=0 Назва з екрану (дата звернення 02.05.2024)
19. Що таке MongoDB? вступ, Архітектура, функції та приклад // Guru99. URL: https://www.guru99.com/uk/what-is-mongodb.html (дата звернення: 02.05.2024).
20. Render: [Веб-сайт]. URL: https://dashboard.render.com/ (дата звернення: 18.05.2024).
21. ЩО ТАКЕ ДИЗАЙН ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА GUI? // QATestLab . URL: https://training.qatestlab.com/blog/technical-articles/  
    what-is-graphical-user-interface-design/ (дата звернення: 20.04.2024).
22. Що таке JavaScript і для чого він потрібен // GOIT. 2024. URL: https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/ (дата звернення: 24.05.2024).
23. Що таке фреймворк: пояснюємо простими словами // Вrainlab. 2023. URL: https://brainlab.com.ua/uk/blog-uk/shho-take-frejmvork-poyasnyuyemo-prostymy-slovamy (дата звернення: 21.05.2024).
24. Що таке Node JS простими словами // Dan-it. URL: https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/ (дата звернення: 05.06.2024).
25. Одно-файлові компоненти // Vue-Mastery. URL: https://ua.vuejs.org/guide/  
    scaling-up/sfc.html (дата звернення: 05.06.2024).
26. Що таке Vue? // Vue-Mastery. URL: https://ua.vuejs.org/guide/  
    introduction.html#what-is-vue (дата звернення: 05.06.2024).
27. Розробка вебсайтів TECHNOLOGIES EXPRESS.JS // Brander. URL: https://brander.ua/technologies/expressjs (дата звернення: 05.06.2024).
28. Вступ до Mongoose для MongoDB та Node.js // envato-tuts+. URL: https://code.tutsplus.com/uk/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527a (дата звернення: 05.06.2024).
29. Vue Component Framework // Vuetify. URL: https://vuetifyjs.com/en/ (дата звернення: 06.05.2024).
30. Code Editing. Redefined: [Веб-сайт]. URL: https://code.visualstudio.com/ (дата звернення: 06.06.2024).
31. Метод Scrum - як виправдати очікування замовника // Apix-drive. URL: https://apix-drive.com/ua/blog/useful/metod-scrum (дата звернення: 06.06.2024).
32. ЩО ТАКЕ ТЕСТУВАННЯ ПЗ, ЙОГО ЕТАПИ, ВИДИ, ІНСТРУМЕНТИ // Sigma Software University. URL: https://university.sigma.software/  
    what-is-software-testing/ (дата звернення: 06.06.2024).
33. Jest: [Веб-сайт]. URL: https://jestjs.io/uk/ (дата звернення: 10.06.2024).
34. Класифікація небезпечних і шкідливих виробничих факторів // Управління Держпраці у Вінницькій області. URL: https://vn.dsp.gov.ua/novini-upravlinnya/klasifikatsiya-nebezpechnih/ (дата звернення: 12.06.2024).
35. Одчаренко М. С., Одчаренко А. М., Степанов В. І. Основи охорони праці. Харків. 341 с.
36. Орлеан С. І. МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ РОЗДІЛУ «РОЗРАХУНОК ЕКОНОМІЧНИХ ПОКАЗНИКІВ» ДИПЛОМНИХ РОБІТ для здобувачів освіти спеціалізації 121 Інженерія програмного забезпечення спеціальності «Розробка програмного забезпечення»: методичні рекомендації. Полтава: Відокремлений структурний підрозділ «Полтавський політехнічний фаховий коледж Національного технічного університету «Харківський політехнічний інститут», 2023. 24 с.
37. Малєєва, Ю.А. Методичні вказівки з дипломного проектування бакалаврів спеціальності 122 «Комп’ютерні науки» [Текст]: навч. посіб. до виконання розрахункових робіт / Ю. А. Малєєва, Т. М. Соляник, А. В. Попов. – Харків: Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2019. – 72 с.

# 

# ДОДАТОК А. ВИХІДНІ КОДИ (ФРАГМЕНТ)

app.post(

"/generateCertificate",

[checkToken, checkRole("teacher")],

async (req, res) => {

**const** { CertData } = req.body;

**const** selectedStudentIds = req.body.selectedStudents;

**try** {

**const** selectedStudents = await User.find({

\_id: { $in: selectedStudentIds },

});

**const** templateId = req.body.selectedTemplateId;

**const** template = await Template.findById(templateId);

**if** (template.image) {

**const** dataURI = template.image;

**let** studentsCnt = selectedStudents.**length**;

**const** sertificates = [];

**const** targetWidth = 1200;

**const** scaleFactor = template.imageWidth / targetWidth;

selectedStudents.**forEach**((student) => {

**const** certNumber = `CERT-${Date.now()}-${Math.**floor**(

Math.**random**() \* 1000

)}`;

**const** adjustedTemplate = {

...template,

title\_top: template.title\_top / scaleFactor + 15,

title\_left: template.title\_left / scaleFactor,

duration\_top: template.duration\_top / scaleFactor + 15,

duration\_left: template.duration\_left / scaleFactor,

teacherSurname\_top: template.teacherSurname\_top / scaleFactor + 15,

teacherSurname\_left: template.teacherSurname\_left / scaleFactor,

studentName\_top: template.studentName\_top / scaleFactor + 15,

studentName\_left: template.studentName\_left / scaleFactor,

dateOfGiving\_top: template.dateOfGiving\_top / scaleFactor + 10,

dateOfGiving\_left: template.dateOfGiving\_left / scaleFactor,

title\_is\_centred: template.title\_is\_centred,

duration\_is\_centred: template.duration\_is\_centred,

teacherName\_is\_centred: template.teacherName\_is\_centred,

studentName\_is\_centred: template.studentName\_is\_centred,

date\_is\_centered: template.date\_is\_centered,

};

**const** center = "transform: translateX(-50%);";

**const** html = `

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Certificate Template</title>

<style>

body {

font-family: Arial, sans-serif;

width: ${targetWidth}px;

height: auto;

position: relative;

}

.certificate {

width: ${targetWidth}px;

height: auto;

position: relative;

}

.certificate .title {

position: absolute;

font-size: 20px;

color: ${adjustedTemplate.title\_color};

top: ${adjustedTemplate.title\_top}px;

left: ${adjustedTemplate.title\_left}px;;

${adjustedTemplate.title\_is\_centred ? center : ""}

z-index: 999;

}

.certificate .duration {

position: absolute;

font-size: 18px;

color: ${adjustedTemplate.duration\_color};

top: ${adjustedTemplate.duration\_top}px;

left: ${adjustedTemplate.duration\_left}px;

${adjustedTemplate.duration\_is\_centred ? center : ""}

z-index:999

}

.certificate .teacher {

position: absolute;

font-size: 18px;

color: ${adjustedTemplate.teacherSurname\_color};

top: ${adjustedTemplate.teacherSurname\_top}px;

left: ${adjustedTemplate.teacherSurname\_left}px;

${adjustedTemplate.teacherName\_is\_centred ? center : ""}

z-index:999

}

.certificate .student {

position: absolute;

font-size: 18px;

color: ${adjustedTemplate.studentName\_color};

top: ${adjustedTemplate.studentName\_top}px;

left: ${adjustedTemplate.studentName\_left}px;

${adjustedTemplate.studentName\_is\_centred ? center : ""}

z-index: 999;

}

.certificate .givingDate {

position: absolute;

font-size: 18px;

color: ${adjustedTemplate.dateOfGiving\_color};

top: ${adjustedTemplate.dateOfGiving\_top}px;

left: ${adjustedTemplate.dateOfGiving\_left}px;

${adjustedTemplate.date\_is\_centered ? center : ""}

z-index: 999;

}

.cert-picture{

position: relative;

width: ${targetWidth}px;

height: auto;

}

</style>

</head>

<body>

<div class="certificate">

<div class="title">${CertData.title}</div>

<div class="duration">${CertData.duration}</div>

<div class="teacher">${CertData.teacherSurname}</div>

<div class="student">${student.name}</div>

<div class="givingDate">${CertData.dateOfGiving}</div>

<img class="cert-picture" src="{{imageSource}}" alt="Certificate Template">

</div>

</body>

</html>

`;

nodeHtmlToImage({

html,

content: { imageSource: dataURI },

encoding: "buffer",

}).then(async (imageBuffer) => {

**const** base64Image = imageBuffer.**toString**("base64");

**const** newCertificate = **new** Certificate({

studentId: student.id,

templateId: template.id,

image: base64Image,

certNumber: certNumber,

title: CertData.title,

duration: CertData.duration,

teacherSurname: CertData.teacherSurname,

dateOfGiving: CertData.dateOfGiving,

});

await newCertificate.save();

sertificates.push(base64Image);

studentsCnt--;

**if** (studentsCnt <= 0) {

res.status(200).json(sertificates);

}

});

});

}

} **catch** (error) {

console.error("Error generating certificate:", error);

}

}

);

# 

# 

# 

# 

# ДОДАТОК Б. ЗНІМКИ ЕКРАНУ ГОТОВОГО ПРОДУКТУ

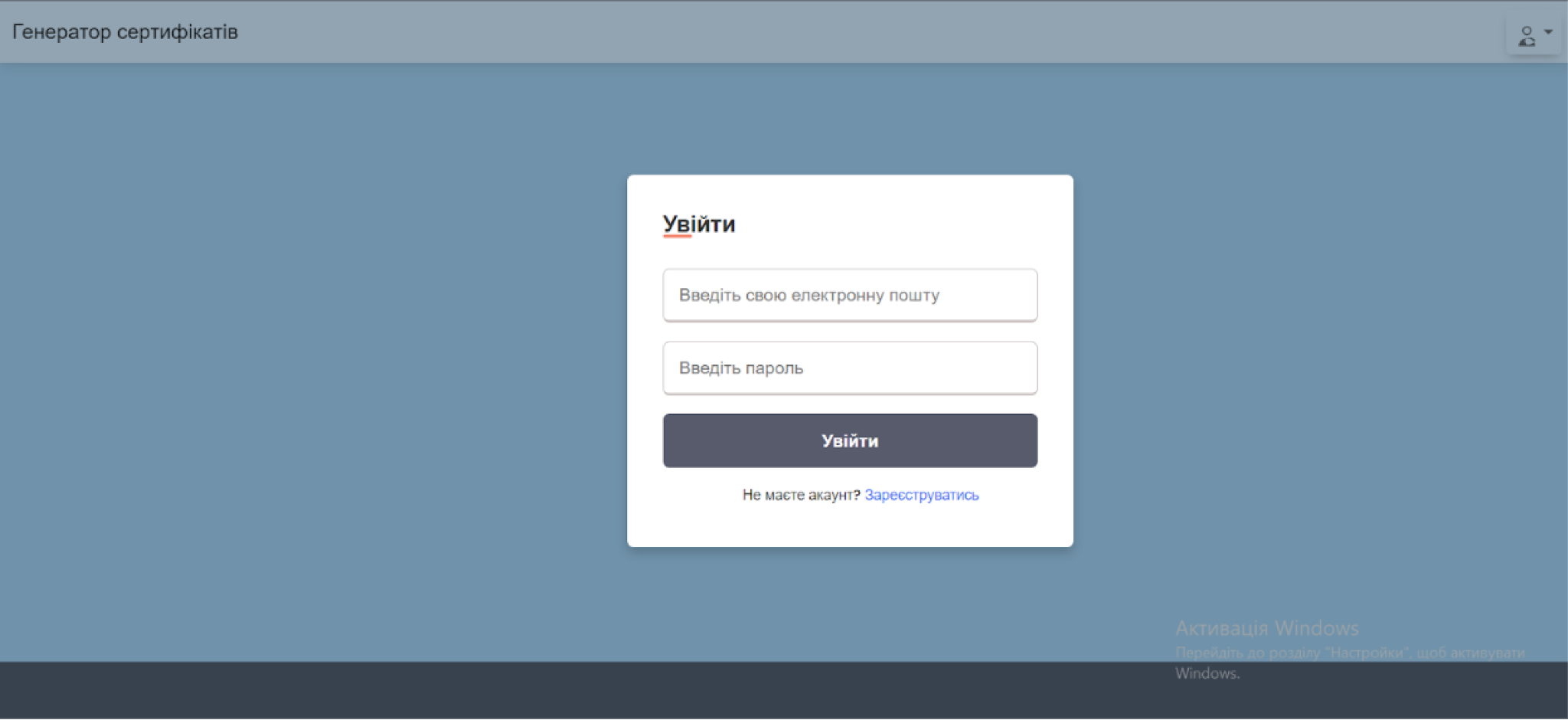


Рисунок Б.1 – Сторінка входу

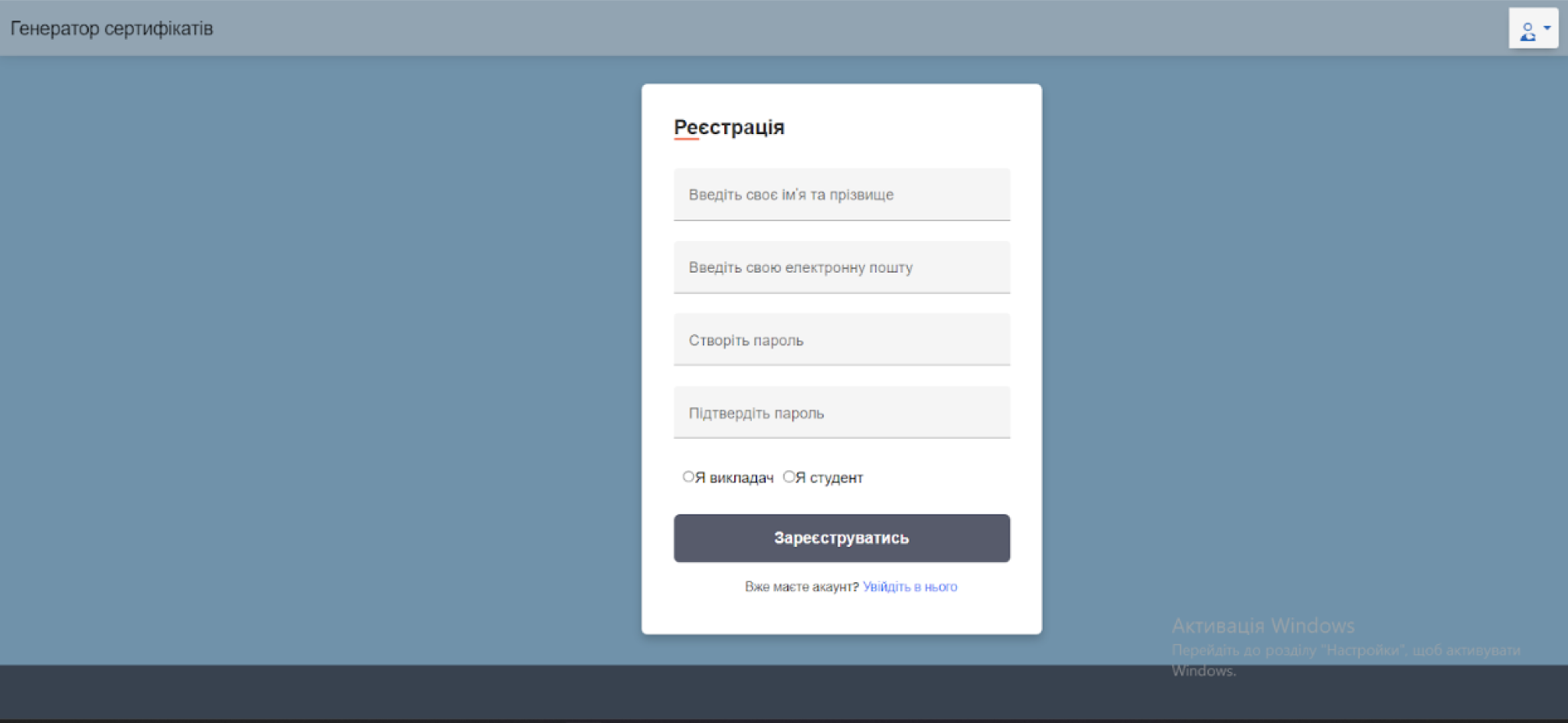


Рисунок Б.2 – Сторінка реєстрації

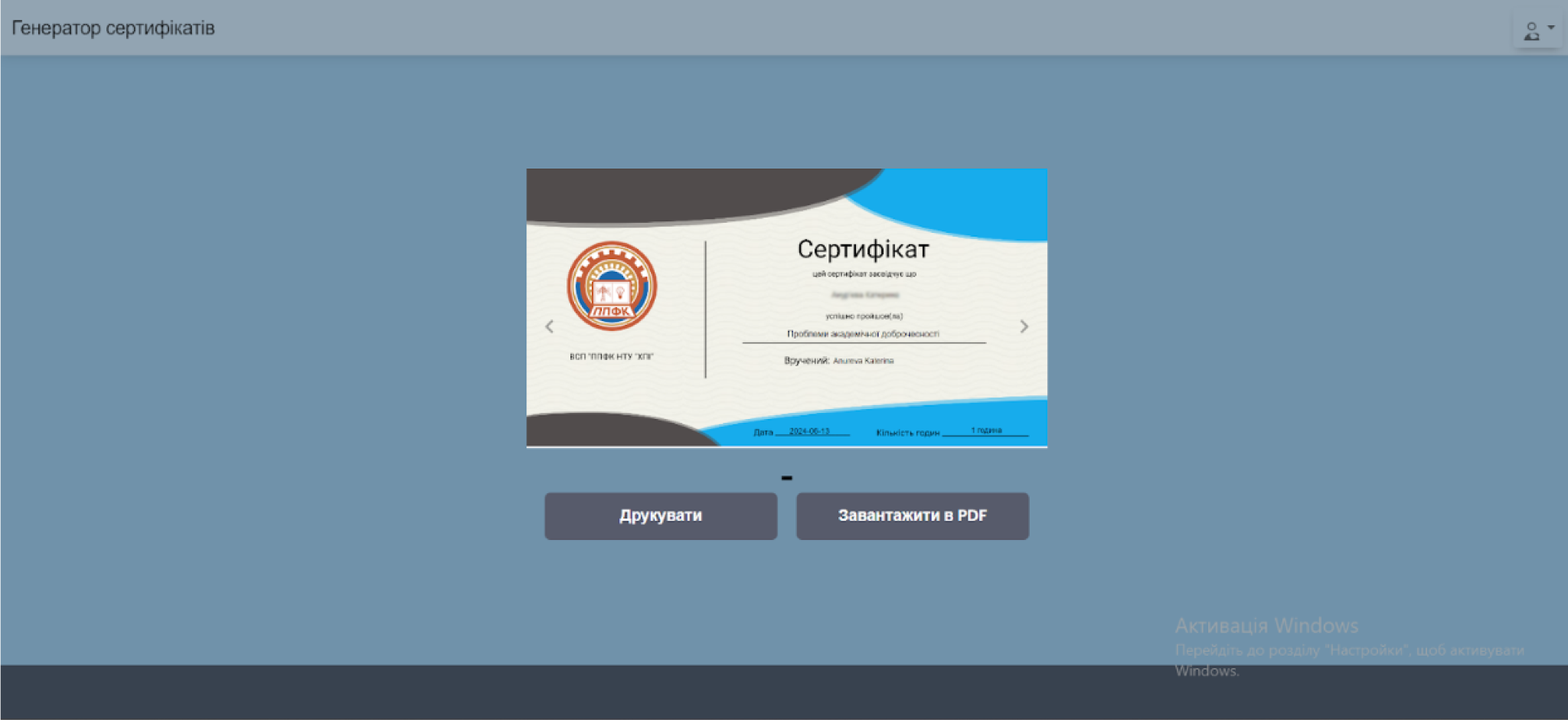


Рисунок Б.3 – Сторінка слухача

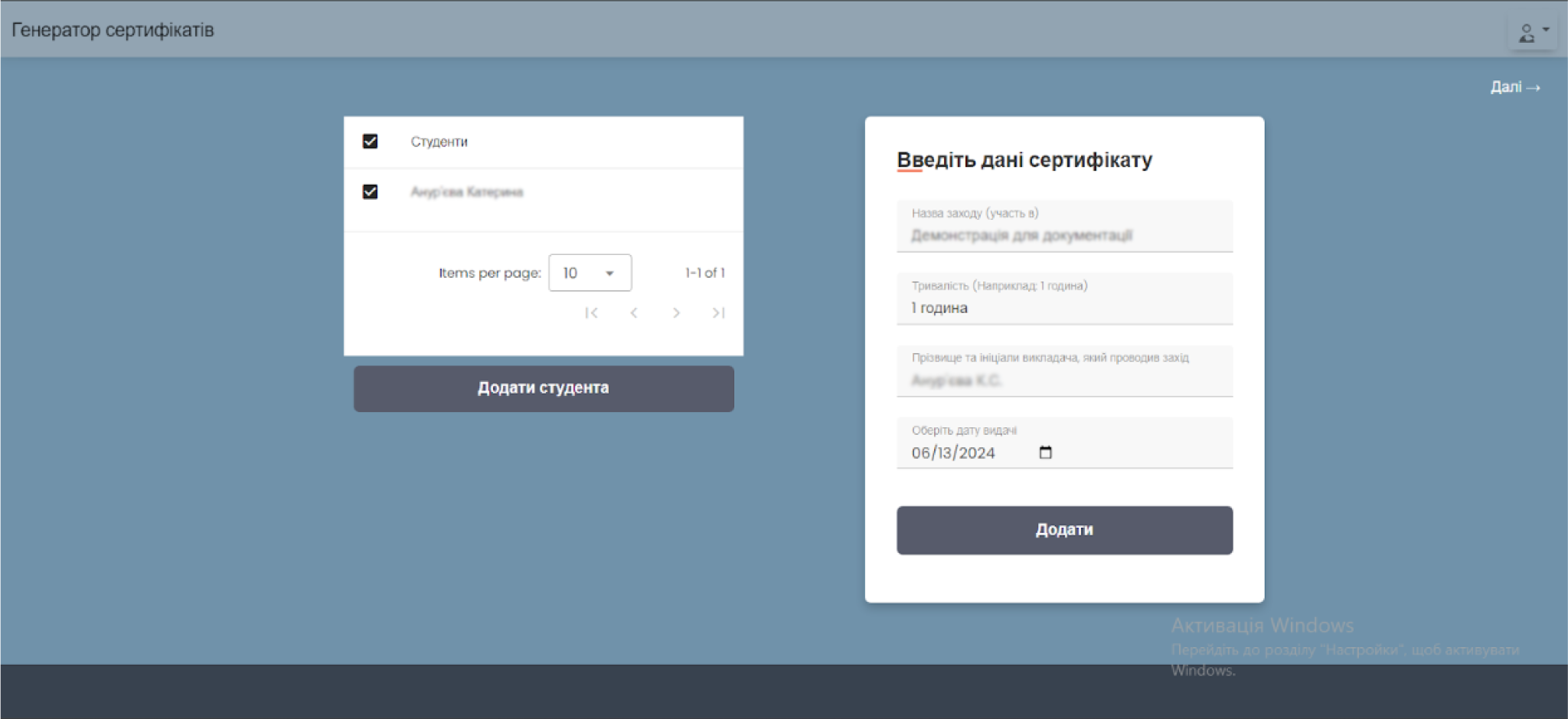


Рисунок Б.4 – Заповнення даних про сертифікат

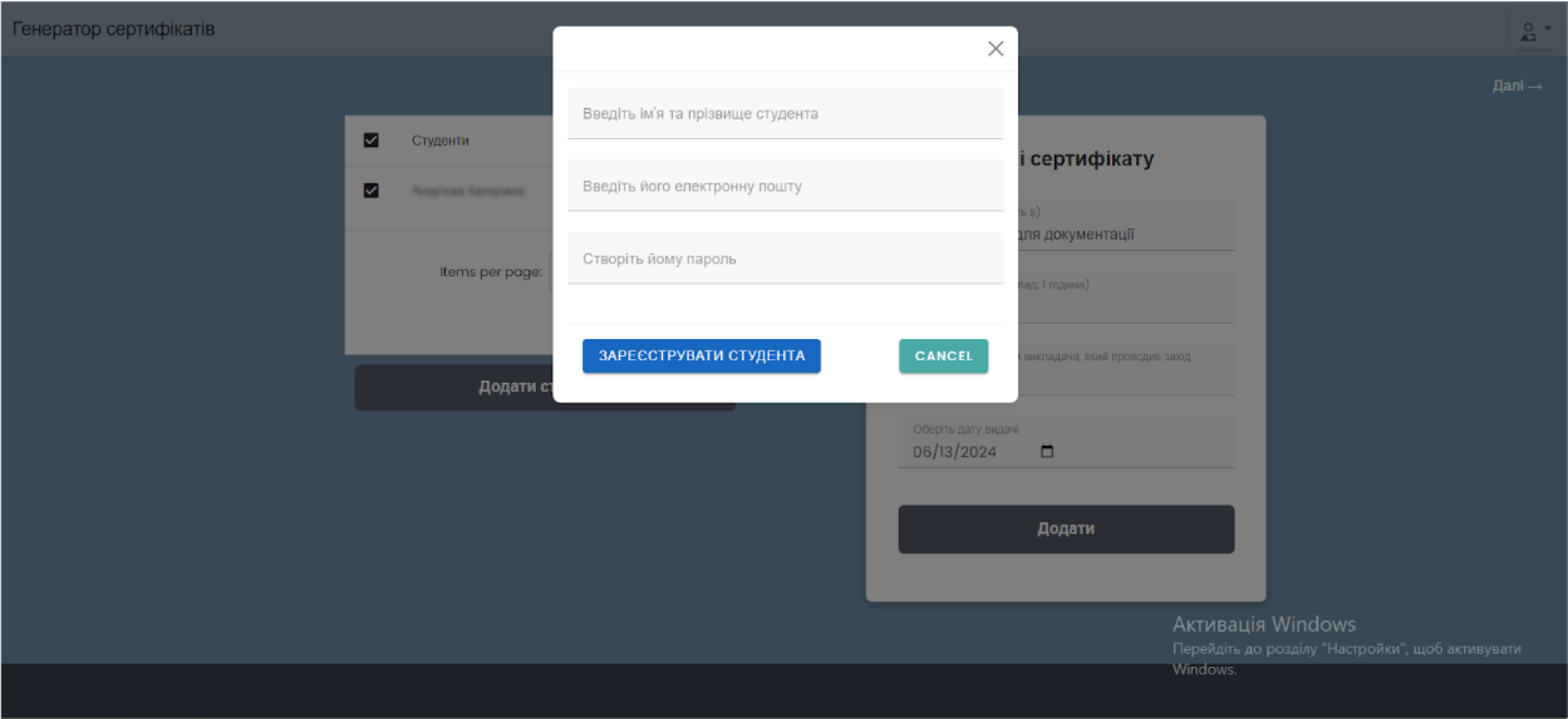


Рисунок Б.5 – Можливість додати слухача, якщо він відсутній в списку

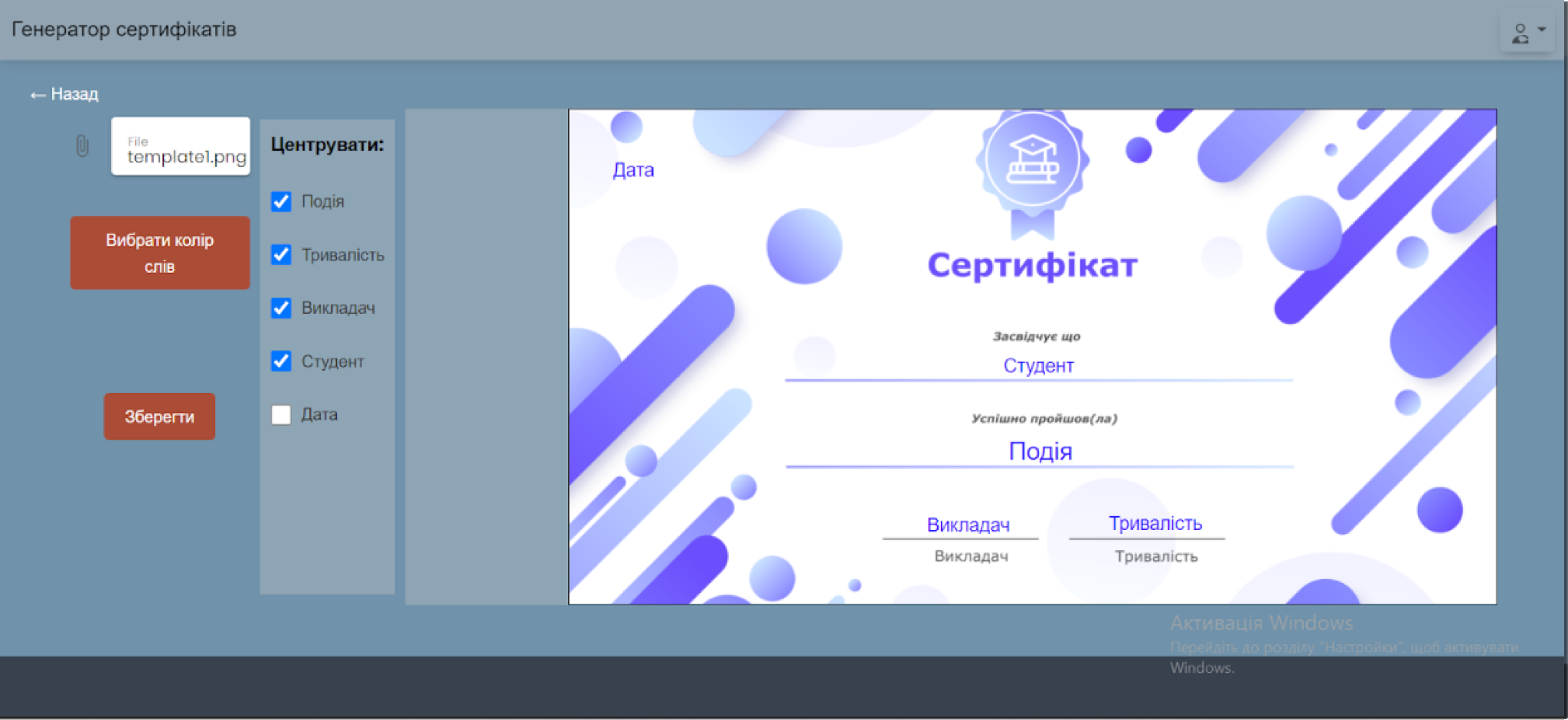


Рисунок Б.6 – Створення шаблону сертифікату

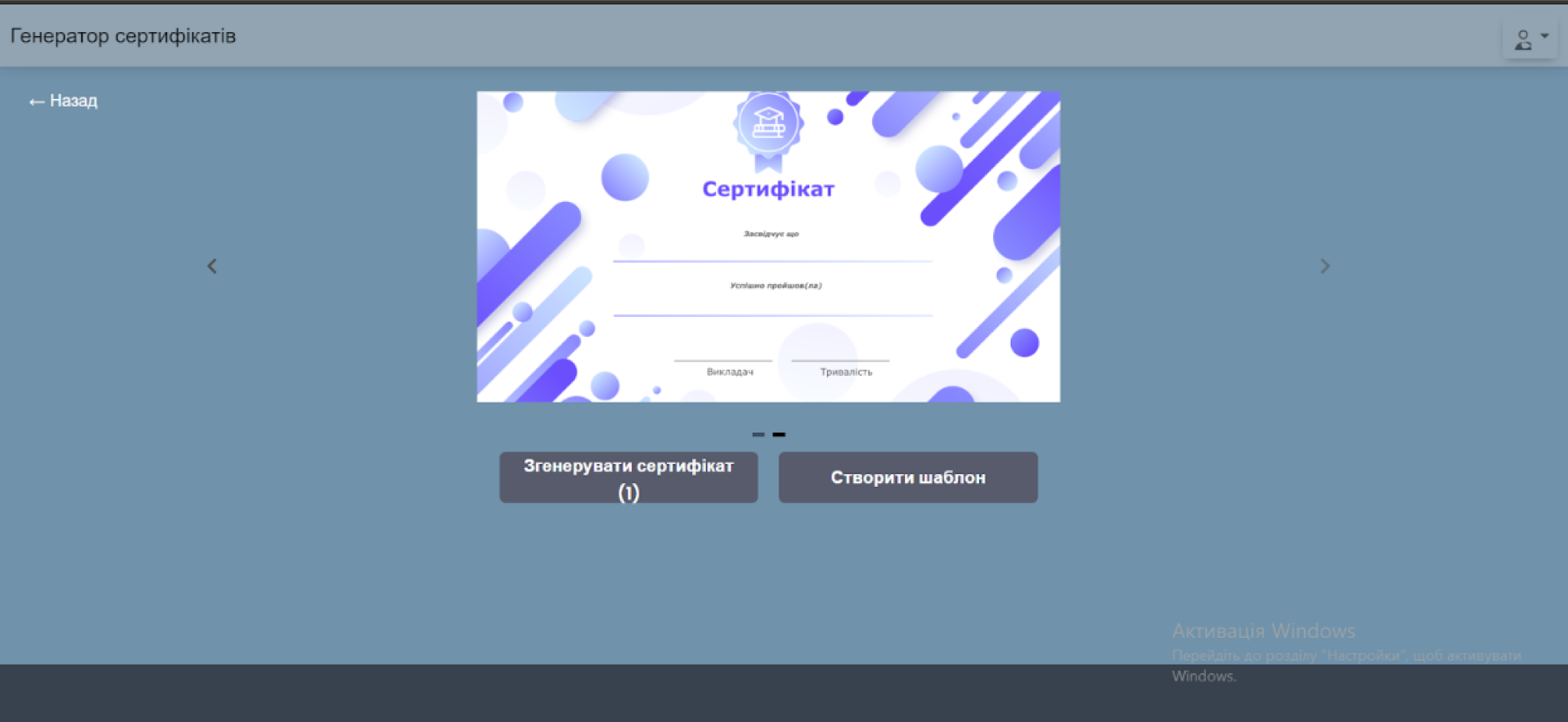


Рисунок Б.7 – Генерація сертифікату (Частина 1)

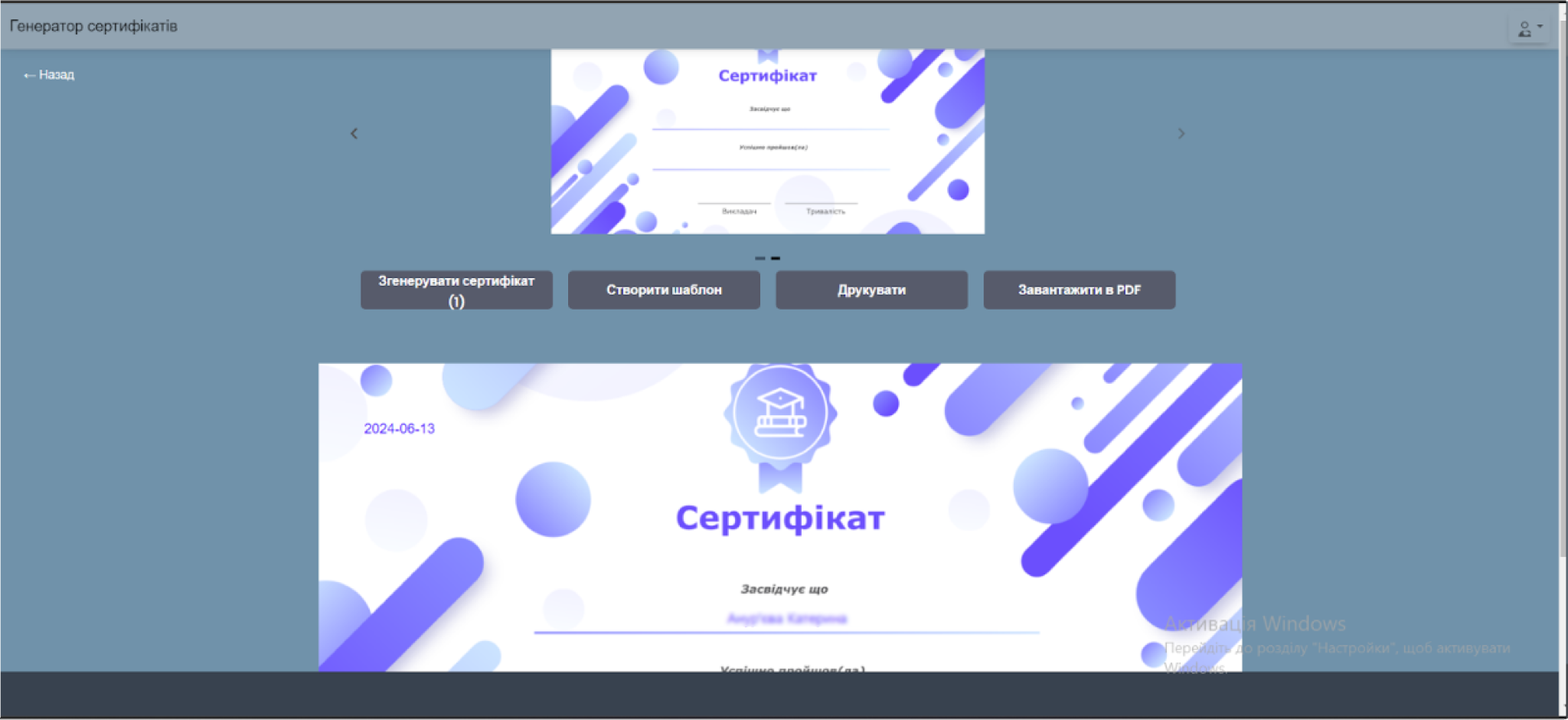


Рисунок Б.8 – Генерація сертифікату (Частина 2)