Dokumentacja Projektu Zespołowego: Cyfrowy Kurator Kolekcji

Spis treści

[Autorzy i Podział Zadań 2](#_Toc204268568)

[Skład Zespołu 2](#_Toc204268569)

[Podział Zadań 2](#_Toc204268570)

[Cel i Zakres Projektu 2](#_Toc204268571)

[Cel 2](#_Toc204268572)

[Zakres podstawowy 2](#_Toc204268573)

[Zakres rozszerzony (nie zaimplementowany) 3](#_Toc204268574)

[Specyfikacja Wymagań 3](#_Toc204268575)

[Wymagania Funkcjonalne 3](#_Toc204268576)

[Wymagania Niefunkcjonalne 3](#_Toc204268577)

[Ograniczenia 3](#_Toc204268578)

[Architektura i Technologie 4](#_Toc204268579)

[Architektura Systemu 4](#_Toc204268580)

[Stos Technologiczny 4](#_Toc204268581)

[Technologie Backendowe: 4](#_Toc204268582)

[Technologie Frontendowe: 4](#_Toc204268583)

[Dokumentacja UML 4](#_Toc204268584)

[Diagram Klas 4](#_Toc204268585)

[Diagram Przypadków Użycia 5](#_Toc204268586)

[Diagram Aktywności 6](#_Toc204268587)

[Struktura Kodu i Dokumentacja Automatyczna 7](#_Toc204268588)

[Struktura Kodu 7](#_Toc204268589)

[Dokumentacja Automatyczna 8](#_Toc204268590)

[Prezentacja Implementacji 9](#_Toc204268591)

[Instrukcja uruchomienia 9](#_Toc204268592)

[Konta użytkowników 10](#_Toc204268593)

[Wyświetlanie publicznej kolekcji 13](#_Toc204268594)

[Dodawanie nowego przedmiotu 15](#_Toc204268595)

[Dodawanie notatek do przedmiotu 16](#_Toc204268596)

[Dodawanie zdjęć do przedmiotu 17](#_Toc204268597)

[Wyświetlanie własnej kolekcji 18](#_Toc204268598)

[Testowanie 19](#_Toc204268599)

[Narzędzia i Konfiguracja 19](#_Toc204268600)

[Struktura Testów 20](#_Toc204268601)

[Uruchomienie i Wyniki Testów 20](#_Toc204268602)

[Podsumowanie 21](#_Toc204268603)

[Wnioski 21](#_Toc204268604)

[Projekt "Cyfrowy Kurator Kolekcji" został zrealizowany pomyślnie, czego efektem jest stabilna aplikacja webowa. Zbudowano ją w oparciu o nowoczesny i sprawdzony stos technologiczny, gdzie za logikę odpowiada framework Flask z SQLAlchemy, a za interfejs użytkownika Bootstrap 5 i dynamiczne biblioteki JavaScript, takie jak DataTables. Modułowa architektura oparta na komponentach Flask Blueprints okazała się kluczowa dla utrzymania porządku w kodzie. Jakość i niezawodność aplikacji została potwierdzona przez wdrożenie zestawu testów automatycznych, które weryfikują poprawność działania wszystkich kluczowych warstw aplikacji. 21](#_Toc204268605)

[Propozycje rozwoju 21](#_Toc204268606)

[Bibliografia 22](#_Toc204268607)

# Autorzy i Podział Zadań

## Skład Zespołu

* **Hubert Michna** (indeks: w67259)
* **Patryk Pieniążek** (indeks: w67174)

## Podział Zadań

|  |  |
| --- | --- |
| **Zadanie / Obszar** | **Odpowiedzialna osoba** |
| Backend (Logika aplikacji, API) | Hubert Michna |
| Frontend (Interfejs użytkownika) | Hubert Michna |
| Baza danych (Projekt i zarządzanie) | Hubert Michna |
| Dokumentacja techniczna i UML | Patryk Pieniążek |
| Testowanie | Patryk Pieniążek |

# Cel i Zakres Projektu

## Cel

Tematem projektu jest **Cyfrowy kurator kolekcji**.

Głównym celem jest stworzenie elastycznej aplikacji webowej do kompleksowego zarządzania dowolnym typem osobistych kolekcji. Aplikacja ma umożliwiać użytkownikom śledzenie przedmiotów, dodawanie szczegółowych informacji, zdjęć oraz monitorowanie ich wartości rynkowej.

Pełny kod źródłowy projektu jest dostępny w publicznym repozytorium GitHub pod adresem: <https://github.com/Hubsik456/PRO_Projekt_Zespolowy/>.

## Zakres podstawowy

* Możliwość definiowania własnych kolekcji i ich pól.
* Szczegółowy opis przedmiotów (rok, stan, cena zakupu, lokalizacja, notatki).
* Zarządzanie multimediami (dodawanie zdjęć/skanów do każdego przedmiotu).
* Podstawowe raportowanie, sortowanie i filtrowanie z wykorzystaniem dynamicznych tabel.
* Ustawienia prywatności dla przedmiotów w kolekcji.
* Personalizacja wyglądu poprzez wybór motywu wizualnego (z opcją jasny/ciemny).
* Eksport danych do formatów CSV, PDF, XLSX oraz do schowka.

## Zakres rozszerzony (nie zaimplementowany)

* Opcje społecznościowe (udostępnianie kolekcji, komentowanie).
* Zaawansowane raporty i statystyki.
* Współdzielona, publiczna baza danych przedmiotów ułatwiająca katalogowanie popularnych obiektów.
* Wykresy i wizualizacje wartości kolekcji w czasie.

# Specyfikacja Wymagań

## Wymagania Funkcjonalne

* **Zarządzanie Użytkownikiem**
  + Rejestracja, logowanie, wylogowanie.
  + Edycja danych profilu.
  + Zmiana hasła i usunięcie konta.
* **Zarządzanie Kolekcją**
  + Dodawanie, edycja i usuwanie przedmiotów.
  + Dołączanie szczegółowych danych: opis, cena, wartość, kategoria.
  + Przesyłanie wielu zdjęć do każdego przedmiotu.
  + Dodawanie prywatnych notatek i niestandardowych pól.
* **Interakcja i Personalizacja**
  + Przeglądanie i filtrowanie przedmiotów kolekcji przedmiotów publicznych.
  + Przeglądanie i filtrowanie własnej kolekcji przedmiotów.
  + Wyświetlanie zdjęć przedmiotów w formie interaktywnej galerii (slider).
  + Możliwość zmiany motywu wizualnego aplikacji (z opcją trybu jasny/ciemny).
* **Eksport Danych**
  + Możliwość wyeksportowania danych o kolekcji do schowka lub formatów CSV, PDF czy XLSX.

## Wymagania Niefunkcjonalne

* **Wydajność:** Szybki czas odpowiedzi aplikacji, nieprzekraczający 2 sekund dla typowych operacji użytkownika.
* **Bezpieczeństwo:** Zabezpieczenie haseł użytkowników poprzez ich hashowanie. Ochrona przed podstawowymi atakami webowymi (np. CSRF).
* **Użyteczność:** Intuicyjny i spójny interfejs użytkownika, łatwy do opanowania dla nowej osoby.
* **Skalowalność:** Modułowa architektura (oparta na Flask Blueprints) umożliwiająca łatwą rozbudowę o nowe funkcje.
* **Kompatybilność:** Poprawne działanie aplikacji na najnowszych wersjach popularnych przeglądarek internetowych (Chrome, Firefox, Edge).

## Ograniczenia

* **Czasowe:** Projekt realizowany w miesiącach (Maj-Lipiec 2025).
* **Technologiczne:** Aplikacja oparta o stos technologiczny: Python, framework Flask, ORM SQLAlchemy oraz framework frontendowy Bootstrap.
* **Zasobowe:** Zespół projektowy składający się z dwóch osób.

# Architektura i Technologie

## Architektura Systemu

Aplikacja została zaprojektowana w architekturze **Klient-Serwer** z wykorzystaniem wzorca projektowego **Model-Widok-Kontroler (MVC)**.

* **Backend:** Aplikacja napisana w języku **Python** z wykorzystaniem lekkiego frameworka **Flask**. Jest on odpowiedzialny za logikę biznesową, routing, obsługę żądań HTTP oraz komunikację z bazą danych. Struktura aplikacji jest modułowa, oparta na komponentach **Flask** **Blueprints**.
* **Frontend:** Interfejs użytkownika jest renderowany po stronie serwera za pomocą silnika szablonów **Jinja2**. Wygląd i interaktywność są budowane w oparciu o framework **Bootstrap 5**, z wykorzystaniem motywów **Bootswatch** oraz dodatkowych bibliotek JavaScript.
* **Baza danych:** Rolę systemu zarządzania bazą danych pełni **SQLite**, a do interakcji z nią wykorzystywany jest pakiet Flask-**SQLAlchemy**, który dostarcza warstwę **ORM (Object-Relational Mapping)**.

## Stos Technologiczny

### Technologie Backendowe:

* **Język programowania**: Python 3.13
* **Framework**: Flask
* **Baza danych**: SQLite
* **ORM**: SQLAlchemy (poprzez Flask-SQLAlchemy)
* **Obsługa formularzy**: Flask-WTF / WTForms
* **Zarządzanie sesjami**: Flask-Login
* **Zarządzanie zmiennymi środowiskowymi**: python-dotenv

### Technologie Frontendowe:

* **Framework CSS:** Bootstrap 5 (z motywami Bootswatch)
* **Ikony**: Bootstrap Icons
* **Silnik szablonów**: Jinja2
* **Biblioteki JavaScript**:
  + **jQuery**: Wymagane jako zależność dla biblioteki DataTables.
  + **DataTables**: Zaawansowane tabele z funkcjami sortowania, filtrowania i eksportu danych (CSV, PDF, XLSX, schowek).
  + **Swiper.js**: Interaktywne galerie zdjęć (slidery).

# Dokumentacja UML

## Diagram Klas

Diagram klas przedstawia statyczną strukturę systemu, ilustrując kluczowe klasy (modele bazy danych), ich atrybuty oraz relacje między nimi. Stanowi on fundament architektury danych aplikacji.

Centralnym punktem modelu jest klasa **Przedmiot**, która reprezentuje pojedynczy obiekt w kolekcji użytkownika. Jest ona powiązana z wieloma innymi klasami:

* Każdy Przedmiot musi należeć do jednego Uzytkownika.
* Każdy Przedmiot jest przypisany do jednej Kategorii.
* Przedmiot posiada dwa odwołania do klasy Waluta – jedno dla ceny zakupu, drugie dla wartości rynkowej.
* Do jednego Przedmiotu można przypisać wiele obiektów klas Zdjecie, Notatka oraz PoleWlasne.
* Klasa PoleWlasne (reprezentująca niestandardową daną, np. "Materiał: Stal") jest powiązana z klasą PoleWlasneRodzaj, która definiuje typ tego pola (np. "Materiał").

Taka struktura zapewnia wysoką elastyczność i integralność danych w systemie.

Obraz zawierający tekst, diagram, Plan, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 1: Diagram klas

## Diagram Przypadków Użycia

Diagram przypadków użycia ilustruje funkcjonalności systemu z perspektywy użytkownika. Wyróżniono w nim dwa główne typy aktorów: **Gościa** (niezalogowanego użytkownika) oraz **Użytkownika Zalogowanego**.

**Gość** może przeglądać ogólnodostępne strony, **w tym publiczne kolekcje innych osób**, zarejestrować nowe konto oraz zalogować się do systemu.

**Użytkownik Zalogowany** dziedziczy wszystkie uprawnienia Gościa i dodatkowo posiada pełen zakres uprawnień do zarządzania swoim kontem i własną kolekcją. Może modyfikować swoje dane profilowe, zarządzać swoimi zbiorami, eksportować dane oraz personalizować interfejs.

Obraz zawierający tekst, diagram, linia, Równolegle

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 2: Diagram przypadków użycia

## Diagram Aktywności

Diagram aktywności przedstawia przepływ pracy związany z dodawaniem nowego przedmiotu do kolekcji przez zalogowanego użytkownika. Proces rozpoczyna się od wyboru opcji "Dodaj przedmiot".

Następnie system wyświetla formularz, który użytkownik musi wypełnić, podając kluczowe dane, takie jak nazwa, cena czy kategoria. Po przesłaniu formularza system dokonuje walidacji danych.

W tym miejscu ścieżka procesu może się rozgałęzić:

* Jeśli dane są **niepoprawne**, system ponownie wyświetla formularz wraz z komunikatami o błędach, a użytkownik musi poprawić wprowadzone informacje.
* Jeśli dane są **poprawne**, system zapisuje nowy przedmiot w bazie danych, wyświetla komunikat o sukcesie i przekierowuje użytkownika na stronę z listą jego kolekcji, gdzie nowy przedmiot jest już widoczny.

Diagram ten ilustruje zarówno idealny scenariusz, jak i obsługę błędów, co jest fundamentalnym aspektem interakcji użytkownika z aplikacją.

Obraz zawierający tekst, paragon, diagram, Równolegle

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 3: Diagram aktywności

# Struktura Kodu i Dokumentacja Automatyczna

## Struktura Kodu

Projekt posiada logiczną strukturę katalogów, która oddziela poszczególne warstwy i moduły aplikacji. Główne komponenty aplikacji znajdują się w katalogu Flask\_Aplikacja.

* **PRO\_Projekt\_Zespolowy**
  + **/Flask\_Aplikacja –** Główny kod aplikacji
    - **/Aplikacja**
      * /Debug – Blueprint dla narzędzi deweloperskich
      * /Kolekcja – Główny Blueprint aplikacji
      * /Konto – Blueprint do zarządzania użytkownikiem
      * /Main – Blueprint dla stron ogólnych
      * /Modele – Definicje modeli bazy danych SQLAlchemy
      * /Statyczne – Pliki statyczne (CSS, JS, obrazy)
      * /Szablony – Szablony HTML (Jinja2)
    - Konfiguracja.py – Główny plik konfiguracyjny
    - CRUD.py – Plik tworzący bazę z początkowymi danymi.
  + **/Dokumentacja** – Pliki źródłowe i konfiguracja Sphinx
    - /build – Wygenerowana dokumentacja HTML
    - /source – Pliki źródłowe .rst
  + **requirements.txt** – Zależności projektu

## Dokumentacja Automatyczna

Dokumentacja kodu została wygenerowana automatycznie na podstawie *docstringów* osadzonych w kodzie źródłowym przy użyciu narzędzia **Sphinx**. To podejście zapewnia, że dokumentacja jest zawsze spójna z aktualną wersją kodu.

Należy w terminalu przejść do katalogu **Dokumentacja**, a następnie wykonać polecenie:

*./make.bat html* lub *./make.bat singlehtml*

Wygenerowana strona główna dokumentacji znajduje się w pliku: *Dokumentacja/build/html/index.html.*

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 4: Dokumentacja automatyczna

# Prezentacja Implementacji

## Instrukcja uruchomienia

Aby uruchomić aplikację trzeba ją pobrać z repozytorium: <https://github.com/Hubsik456/PRO_Projekt_Zespolowy>

Następnie wejść do głównego katalogu projektu i wykonać w konsoli następujące polecenia:

* python -m venv .venv
* .venv\Scripts\activate
* pip install -r requirements.txt
* cd Flask\_Aplikacja
* flask seed-db
* flask run

Aplikacja domyślnie uruchamia się na porcie 5002, więc aby z niej korzystać trzeba wejść w link: <http://127.0.0.1:5002/>

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 5: Przygotowanie środowiska

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 6: Inicjalizacja bazy i uruchomienie serwera

## Konta użytkowników

Aplikacja umożliwia nowym użytkownikom założenie profilu poprzez formularz rejestracyjny. Po pomyślnym zalogowaniu użytkownik jest przekierowywany na stronę główną aplikacji. Dostęp do panelu zarządzania kontem, widocznego poniżej, pozwala na edycję danych osobowych, zmianę hasła lub trwałe usunięcie profilu.

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 7: Formularz rejestracji użytkownika

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 8: Panel zalogowanego użytkownika

## Wyświetlanie publicznej kolekcji

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 9: Wyświetlanie publicznej kolekcji

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 10: Wyświetlanie przedmiotu z publicznej kolekcji

## Dodawanie nowego przedmiotu

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 11: Formularz dodawania nowego przedmiotu

### Dodawanie notatek do przedmiotu

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 12: Formularz dodawania notatki

### Dodawanie zdjęć do przedmiotu

Obraz zawierający tekst, elektronika, zrzut ekranu, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 13: Formularz dodawania zdjęcia

## Wyświetlanie własnej kolekcji

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 14: Wyświetlanie własnej kolekcji

Obraz zawierający tekst, zrzut ekranu, Strona internetowa, oprogramowanie

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 15: Wyświetlanie własnego przedmiotu

# Testowanie

## Narzędzia i Konfiguracja

Do automatyzacji procesu testowania wykorzystano framework **pytest** wraz z rozszerzeniem **pytest-flask**. Taki zestaw narzędzi pozwala na efektywne i czytelne pisanie testów dla aplikacji webowych opartych na Flasku.

Aby zapewnić pełną izolację od danych produkcyjnych, testy są uruchamiane z wykorzystaniem dedykowanej konfiguracji (KonfiguracjaTestowa), która korzysta z tymczasowej bazy danych **SQLite** przechowywanej w pamięci. Wszystkie dane testowe są tworzone na początku i usuwane po zakończeniu każdej funkcji testowej dzięki mechanizmowi *fixtures* zdefiniowanemu w pliku conftest.py, co gwarantuje powtarzalność i niezależność każdego testu.

## Struktura Testów

Zestaw testów został podzielony na trzy główne kategorie, odzwierciedlające architekturę aplikacji:

1. **Testy Modeli (test\_models.py)**: Weryfikują one najniższą warstwę aplikacji. Sprawdzają, czy modele danych (np. Uzytkownik, Przedmiot) są poprawnie tworzone oraz czy relacje między nimi (np. przynależność przedmiotu do użytkownika) działają zgodnie z założeniami.
2. **Testy Formularzy (test\_forms.py)**: Te testy jednostkowe sprawdzają logikę walidacji każdego formularza w izolacji. Weryfikują, czy formularze poprawnie akceptują prawidłowe dane i odrzucają te, które nie spełniają określonych kryteriów (np. zbyt krótkie hasło).
3. **Testy Widoków (integracyjne)**: Rozdzielone na pliki test\_views\_main.py, test\_views\_konto.py oraz test\_views\_kolekcja.py, symulują one realne interakcje użytkownika z aplikacją. Testy te sprawdzają cały cykl żądania i odpowiedzi – od wysłania żądania HTTP, przez logikę w widoku, interakcję z bazą danych, aż po weryfikację finalnej odpowiedzi serwera, w tym przekierowań i komunikatów zwrotnych.

## Uruchomienie i Wyniki Testów

Pełen zestaw testów jest uruchamiany za pomocą polecenia pytest -v w katalogu Flask\_Aplikacja.

Wszystkie **35 zaimplementowanych testów** przechodzi pomyślnie, co potwierdza stabilność i poprawność działania zaimplementowanych funkcjonalności.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 16: Wyniki testów

Obraz zawierający tekst, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 17: Szczegółowe wyniki testów

# Podsumowanie

## Wnioski

## Projekt "Cyfrowy Kurator Kolekcji" został zrealizowany pomyślnie, czego efektem jest stabilna aplikacja webowa. Zbudowano ją w oparciu o nowoczesny i sprawdzony stos technologiczny, gdzie za logikę odpowiada framework Flask z SQLAlchemy, a za interfejs użytkownika Bootstrap 5 i dynamiczne biblioteki JavaScript, takie jak DataTables. Modułowa architektura oparta na komponentach Flask Blueprints okazała się kluczowa dla utrzymania porządku w kodzie. Jakość i niezawodność aplikacji została potwierdzona przez wdrożenie zestawu testów automatycznych, które weryfikują poprawność działania wszystkich kluczowych warstw aplikacji.

## Propozycje rozwoju

Stworzona baza technologiczna otwiera drogę do przekształcenia aplikacji w kompleksowy ekosystem dla kolekcjonerów. Potencjalne kierunki rozwoju można podzielić na kilka strategicznych obszarów:

* Inteligentna Analiza Rynku i Integracje Zewnętrzne
  + Zaawansowane śledzenie wartości: Rozwinięcie integracji z serwisami aukcyjnymi (Allegro, eBay, Catawiki) oparte na uczeniu maszynowym. System mógłby analizować zakończone aukcje, rozpoznawać trendy i prognozować zmiany wartości poszczególnych przedmiotów i całych kolekcji w czasie.
  + Identyfikacja unikatów: W oparciu o globalną, współdzieloną bazę danych przedmiotów, algorytmy mogłyby identyfikować rzadkie warianty posiadanych przez użytkownika obiektów (np. moneta z rzadkim błędem menniczym, książka z unikatowym wariantem okładki), sugerując ich potencjalnie wyższą wartość.
  + Asystent Zakupów: Moduł, który na podstawie zdefiniowanych przez użytkownika "list życzeń" skanowałby serwisy aukcyjne i powiadamiał o pojawieniu się interesujących go przedmiotów w atrakcyjnych cenach.
* Rozbudowa Społeczności i Gamifikacja
  + Wirtualne wystawy i "Gildie Kolekcjonerskie": Umożliwienie użytkownikom tworzenia tematycznych, wirtualnych wystaw ze swoich zbiorów, które mogliby udostępniać publicznie. Możliwość tworzenia grup (gildii) zrzeszających kolekcjonerów o podobnych zainteresowaniach, z własnymi forami i rankingami.
  + System Osiągnięć i Rzadkości: Wprowadzenie systemu gamifikacji, w którym użytkownicy zdobywaliby odznaki za skompletowanie określonych zestawów (np. "Wszystkie monety z 1934 roku", "Kompletna seria komiksów Thorgal"), posiadanie rzadkich przedmiotów czy aktywność na platformie.
  + Bezpieczny system wymiany i handlu: Stworzenie wewnętrznego, bezpiecznego marketplace'u, gdzie zweryfikowani użytkownicy mogliby wystawiać swoje przedmioty na sprzedaż lub wymianę, z systemem reputacji i escrow.
* Implementacja Nowoczesnych Technologii
  + Rozpoznawanie Obrazów (AI/ML): Stworzenie funkcji, która po zrobieniu zdjęcia smartfonem (w dedykowanej aplikacji mobilnej ) automatycznie rozpozna przedmiot, wyszuka go we współdzielonej bazie danych i zaproponuje uzupełnienie podstawowych pól formularza.
  + Rzeczywistość Rozszerzona (AR): Możliwość "umieszczenia" cyfrowego modelu 3D swojego przedmiotu (np. rzeźby, figurki) we własnym pokoju za pomocą kamery w smartfonie, co pozwoliłoby na tworzenie wirtualnych ekspozycji.
  + Blockchain i NFT: Dla przedmiotów o wysokiej wartości, wprowadzenie opcji stworzenia cyfrowego certyfikatu autentyczności w formie tokenu NFT. Taki certyfikat, zapisany w technologii blockchain, mógłby w sposób niezaprzeczalny dokumentować historię własności (proweniencję) i autentyczność danego egzemplarza.

# Bibliografia

* **Flask** - Dokumentacja: <https://flask.palletsprojects.com/>
* **SQLAlchemy** - Dokumentacja: <https://www.sqlalchemy.org/>
* **Bootstrap 5** - Dokumentacja: <https://getbootstrap.com/>
* **Jinja2** - Dokumentacja: <https://jinja.palletsprojects.com/>
* **pytest** - Dokumentacja: <https://docs.pytest.org/>
* **Flask-Login** - Dokumentacja: <https://flask-login.readthedocs.io/>
* **Flask-WTF** - Dokumentacja: <https://flask-wtf.readthedocs.io/>
* **DataTables** - Dokumentacja: <https://datatables.net/>
* **Swiper.js** - Dokumentacja: <https://swiperjs.com/>
* **Sphinx** - Dokumentacja: <https://www.sphinx-doc.org/>