Dokumentacja Projektu Zespołowego: Cyfrowy Kurator Kolekcji

Spis treści

[Autorzy i Podział Zadań 2](#_Toc204099256)

[Skład Zespołu 2](#_Toc204099257)

[Podział Zadań 2](#_Toc204099258)

[Cel i Zakres Projektu 2](#_Toc204099259)

[Cel 2](#_Toc204099260)

[Zakres podstawowy 2](#_Toc204099261)

[Propozycje rozwinięcia 2](#_Toc204099262)

[Specyfikacja Wymagań 3](#_Toc204099263)

[Wymagania Funkcjonalne 3](#_Toc204099264)

[Wymagania Niefunkcjonalne 3](#_Toc204099265)

[Ograniczenia 3](#_Toc204099266)

[Architektura i Technologie 3](#_Toc204099267)

[Architektura Systemu 3](#_Toc204099268)

[Stos Technologiczny 4](#_Toc204099269)

[Technologie Backendowe: 4](#_Toc204099270)

[Technologie Frontendowe: 4](#_Toc204099271)

[Dokumentacja UML 4](#_Toc204099272)

[Diagram Klas 4](#_Toc204099273)

[Diagram Przypadków Użycia 5](#_Toc204099274)

[Diagram Aktywności 6](#_Toc204099275)

[Struktura Kodu i Dokumentacja Automatyczna 7](#_Toc204099276)

[Struktura Kodu 7](#_Toc204099277)

[Dokumentacja Automatyczna 8](#_Toc204099278)

[Prezentacja Implementacji 9](#_Toc204099279)

[Testowanie 9](#_Toc204099280)

[Podsumowanie 9](#_Toc204099281)

[Bibliografia 9](#_Toc204099282)

# 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.

## Zakres podstawowy

* Możliwość definiowania własnych typów 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 i PDF.

## Propozycje rozwinięcia

* Integracja z zewnętrznymi serwisami (np. Allegro, eBay) w celu śledzenia orientacyjnych cen rynkowych.
* Opcje społecznościowe (udostępnianie kolekcji, komentowanie).
* Zaawansowane raporty i statystyki.
* Dedykowana aplikacja mobilna.
* Współdzielona, publiczna baza danych przedmiotów ułatwiająca katalogowanie popularnych obiektów.
* Wykresy i wizualizacje wartości kolekcji w czasie.

# Specyfikacja Wymagań

## Wymagania Funkcjonalne

* **Zarządzanie Użytkownikiem**
  + Rejestracja, logowanie, wylogowywanie.
  + 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 formatów CSV i PDF.

## 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
* **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).
  + **Swiper.js**: Interaktywne galerie zdjęć (slidery).
  + **Google Charts**: Generowanie dynamicznych wykresów i wizualizacji danych.

# 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*

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

# 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 5: Wyniki testów

Obraz zawierający tekst, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 6: Szczegółowe wyniki testów

# Podsumowanie

# Bibliografia