StudentCards

Oliwia Strzelec, Kamil Marszałek, Piotr Matoszka

WPROWADZENIE

Aplikacja StudentCards to narzędzie edukacyjne stworzone z myślą o użytkownikach, którzy chcą skutecznie i interaktywnie przyswajać wiedzę. Projekt powstał w ramach zajęć Architektura Oprogramowania, a jego celem jest dostarczenie prostego systemu, który pozwala użytkownikom na naukę poprzez tworzenie, zarządzanie i wykorzystywanie zestawów fiszek.

Aplikacja umożliwia:

- Naukę za pomocą zestawów fiszek – zarówno stworzonych przez użytkownika, jak i publicznie udostępnionych przez innych.
- Tworzenie i zarządzanie własnymi zestawami fiszek – w tym dodawanie nowych fiszek, ich edytowanie, usuwanie oraz zmiana statusu zestawu na publiczny.
- Korzystanie z zestawów publicznych – dostępnych do nauki bez konieczności tworzenia konta.

RODZAJ ARCHITEKTURY

Do zrealizowania projektu została wybrana architektura klient-serwer z wykorzystaniem REST API. Jest to popularne podejście w aplikacjach webowych, które zapewnia elastyczność, skalowalność oraz możliwość rozwoju systemu w przyszłości. W tym przypadku, React działa jako klient, który komunikuje się z serwerem opartym na Django za pomocą protokołów HTTP, a dane są wymieniane w formacie JSON przez REST API. Wybór architektury klient-serwer pozwala na:

Rozdzielenie warstw aplikacji

Django pełni rolę backendu, zarządzając logiką biznesową, bazą danych i generowaniem odpowiedzi API, podczas gdy React odpowiada za interfejs użytkownika. Dzięki temu można łatwo zarządzać i rozwijać zarówno część frontendową, jak i backendową aplikacji niezależnie.

Elastyczność i skalowalność

REST API zapewnia łatwą integrację z innymi systemami lub aplikacjami zewnętrznymi w przyszłości. Umożliwia również łatwiejszą modyfikację backendu lub frontendowego interfejsu bez wpływu na resztę systemu.

Wydajność

Dzięki wykorzystaniu REST API komunikacja między frontendem (React) a backendem (Django) jest efektywna. W przypadku tego projektu jest to istotnym elementem, ponieważ aplikacja wymaga częstych interakcji z bazą danych, jak w przypadku zarządzania fiszkami.

Łatwość testowania i utrzymywania
REST API jest dobrze udokumentowane i łatwe do testowania. Można przeprowadzać testy jednostkowe zarówno na frontendzie, jak i backendzie, co ułatwia utrzymanie aplikacji.

ATRYBUTY JAKOŚCIOWE

Atrybuty jakościowe to konkretne cechy systemu, które wpływają na sposób realizacji funkcji i jakości użytkowania. Skupiają się na tym, jak system spełnia potrzeby użytkowników i biznesu.

Atrybuty użytkowe:

Koncentrują się na doświadczeniu użytkownika i jego interakcji z aplikacją.

Używalność
Aplikacja jest skierowana do szerokiego grona użytkowników, w tym osób z niewielkim doświadczeniem technicznym. Prosty i intuicyjny interfejs użytkownika zwiększa jej atrakcyjność oraz poprawia doświadczenia użytkowników.

W jaki sposób atrybut jest realizowany?
- Czytelny i responsywny interfejs użytkownika.
- Proste nawigowanie między funkcjami, takimi jak tworzenie fiszek, gra i przeglądanie zestawów.

Dostępność
Użytkownicy oczekują możliwości korzystania z aplikacji na różnych urządzeniach (komputery, tablety, telefony).

W jaki sposób atrybut jest realizowany?
- Responsywny design frontendu.
- Niskie wymagania wobec urządzeń użytkowników.

Bezpieczeństwo
Aplikacja wymaga ochrony danych użytkowników, szczególnie haseł i dostępu do prywatnych zestawów fiszek.

W jaki sposób atrybut jest realizowany?
- Szyfrowanie haseł w bazie danych.
- Proste mechanizmy uwierzytelniania oparte na JWT (JSON Web Tokens)

Atrybuty efektywne:

Odnoszą się do aspektów technicznych aplikacji, które wpływają na jej stabilność, wydajność, oraz przyszły rozwój.

Wydajność
Szybki czas ładowania stron i minimalizacja opóźnień w aplikacji są kluczowe dla zapewnienia płynnego działania, zwłaszcza przy dużych zbiorach danych.

W jaki sposób atrybut jest realizowany?

- Tworzenie kodu z myślą o najbardziej optymalnej wersji

Niezawodność
Stabilność aplikacji i minimalizacja błędów oraz awarii są kluczowe dla zapewnienia ciągłego dostępu do funkcji.

W jaki sposób atrybut jest realizowany?

- Implementacja kodu w taki sposób, aby przewidywał i obsługiwał potencjalne błędy

Modularność
Struktura aplikacji, która umożliwia łatwe dodawanie nowych funkcji i rozbudowę bez wpływu na inne części systemu.

W jaki sposób atrybut jest realizowany?

- Architektura oparta na komponentach i modułach.
- Każda funkcjonalność jest zaimplementowana jako osobny moduł, co ułatwia rozwój i testowanie.

Modyfikowalność
Aplikacja może być rozwijana w przyszłości, co wymaga łatwego wprowadzania nowych funkcji, takich jak quiz wiedzy, losowa kolejność fiszek czy zmiany w interfejsie.

W jaki sposób atrybut jest realizowany?

- Frontend został zbudowany w React, co umożliwia reużywalność komponentów, ułatwiając wprowadzanie zmian w interfejsie i dodawanie nowych funkcji.
- Komponentowy model React pozwala na łatwą integrację nowych elementów bez konieczności przebudowy całej aplikacji.

Testowalność
Możliwość przeprowadzania testów jednostkowych, integracyjnych i systemowych.

W jaki sposób atrybut jest realizowany?

- Projektowanie aplikacji z myślą o testowalności - izolacja funkcjonalności poszczególnych elementów

MECHANIZMY NAPĘDZAJĄCE

Mechanizmy napędzające w architekturze systemu to kluczowe wymagania biznesowe, jakościowe i funkcjonalne, które mają największy wpływ na kształtowanie architektury projektu. Wynikają z potrzeby stworzenia narzędzia edukacyjnego dostępnego na różnych urządzeniach, które wspiera personalizację nauki. Ich identyfikacja i zrozumienie są kluczowe dla efektywnego projektowania systemu.

Mechanizmy napędzające w tym projekcie koncentrują się na prostocie użytkowania, elastyczności, i dostępności. Obecna architektura jest zoptymalizowana dla niewielkich środowisk użytkowników, ale uwzględnia możliwość rozwoju, co zapewnia, że aplikacja będzie mogła ewoluować zgodnie z potrzebami.

Prosta i intuicyjna obsługa

Aplikacja jest skierowana do użytkowników indywidualnych. Intuicyjny interfejs zwiększy jej atrakcyjność i zachęci do korzystania. Interfejs użytkownika musi być przejrzysty i responsywny. Backend powinien wspierać szybkie operacje, np. natychmiastowe ładowanie zestawów fiszek.

Elastyczność i możliwość rozwoju

Brak obecnych ograniczeń ilościowych na liczbę użytkowników, zestawów czy fiszek wymaga architektury, która będzie mogła się skalować, jeśli aplikacja zyska popularność. Wyraźny podział funkcji (backend), jak i komponentów (frontend), takich jak zarządzanie fiszkami i gra z fiszkami, sprzyja lepszemu zrozumieniu systemu i ułatwia dodawanie nowych funkcji w przyszłości. Zastosowanie API REST umożliwi łatwą integrację z innymi systemami.

Dostępność

Aplikacja musi być dostępna na różnych urządzeniach (desktop, mobilne), aby trafić do szerokiej grupy użytkowników. Responsywny design frontendu zapewnia dostosowanie interfejsu do ekranów o różnych rozmiarach.

Bezpieczeństwo

System musi chronić dane użytkowników, szczególnie w kontekście logowania i przechowywania haseł. Z tego względu uwierzytelnianie i autoryzacja użytkowników z zostanie zaimplementowane z wykorzystaniem JWT. Szyfrowanie danych wrażliwych (np. haseł) w bazie danych jest koniecznością.

CYKL BIZNESOWY

Cykl biznesowy aplikacji StudentCards obejmuje wszystkie kluczowe działania, które wspierają użytkowników w korzystaniu z platformy, tworzeniu treści i osiąganiu ich celów edukacyjnych.

Pozyskanie i uporządkowanie wiedzy

Cele użytkowników:

- Nauka i zapamiętywanie informacji w sposób interaktywny.
- Tworzenie własnych zestawów fiszek dostosowanych do ich potrzeb edukacyjnych.
- Organizacja materiałów w logiczne kategorie.

Reguły i oczekiwania:

- Użytkownik oczekuje szybkiego i intuicyjnego dostępu do funkcji aplikacji.
- Treści muszą być łatwe do edycji, aktualizacji i usuwania.
- System musi być dostępny na różnych urządzeniach.

Identyfikacja głównych procesów biznesowych

Kluczowe procesy dla sukcesu aplikacji:

1. Zarządzanie kontem użytkownika (rejestracja i logowanie)
2. Zarządzanie zestawami fiszek (tworzenie, edytowanie, usuwanie)
3. Nauka poprzez przeglądanie interaktywnych fiszek.

Modelowanie procesów biznesowych

Do modelowania procesów aplikacji wykorzystane zostały narzędzia takie jak:

- Diagram czynności: Przedstawia sekwencję wykonywanych kroków, np. logowanie, tworzenie zestawu fiszek czy gra.
- Diagram przypadków użycia: Opisuje interakcje użytkowników (aktorów) z systemem.

Model dziedziny

Słownik pojęć:

- Fiszka: Jednostka składająca się z pytania i odpowiedzi.
- Zestaw fiszek: Grupa fiszek tematycznych.
- Gra z fiszkami: Interaktywny proces nauki na podstawie zestawów.

Diagram klas:

- Klasa User (zapewniona przez Django): Dane użytkownika (login, hasło).
- Klasa FlashcardSet: Nazwa zestawu, lista fiszek.
- Klasa Flashcard: Pytanie, odpowiedź, kategoria.

Oszacowania ilościowe: brak

Brak limitów wynika z obecnych założeń aplikacji jako małego systemu, który koncentruje się na funkcjonalności i przyjazności użytkownika. W przypadku rozwoju aplikacji i wzrostu

liczby użytkowników można wprowadzić ograniczenia techniczne, aby zapewnić stabilność i wydajność systemu (np. limit 100 fiszek na zestaw).

Modelowanie wymagań użytkownika

Identyfikacja aktorów:

1. Niezalogowany użytkownik może przeglądać publiczne zestawy fiszek.
2. Zalogowany użytkownik ma dostęp do swoich zestawów. Może edytować i zarządzać własnymi zestawami oraz fiszkami.

Główne scenariusze sukcesu:

- Użytkownik z powodzeniem loguje się do systemu.
- Tworzy nowy zestaw fiszek i rozpoczyna grę.
- Osiąga określone postępy w nauce.

Scenariusze alternatywne:

- Nieprawidłowe hasło podczas logowania (system wyświetla komunikat o błędzie).

Budowa modelu systemowego

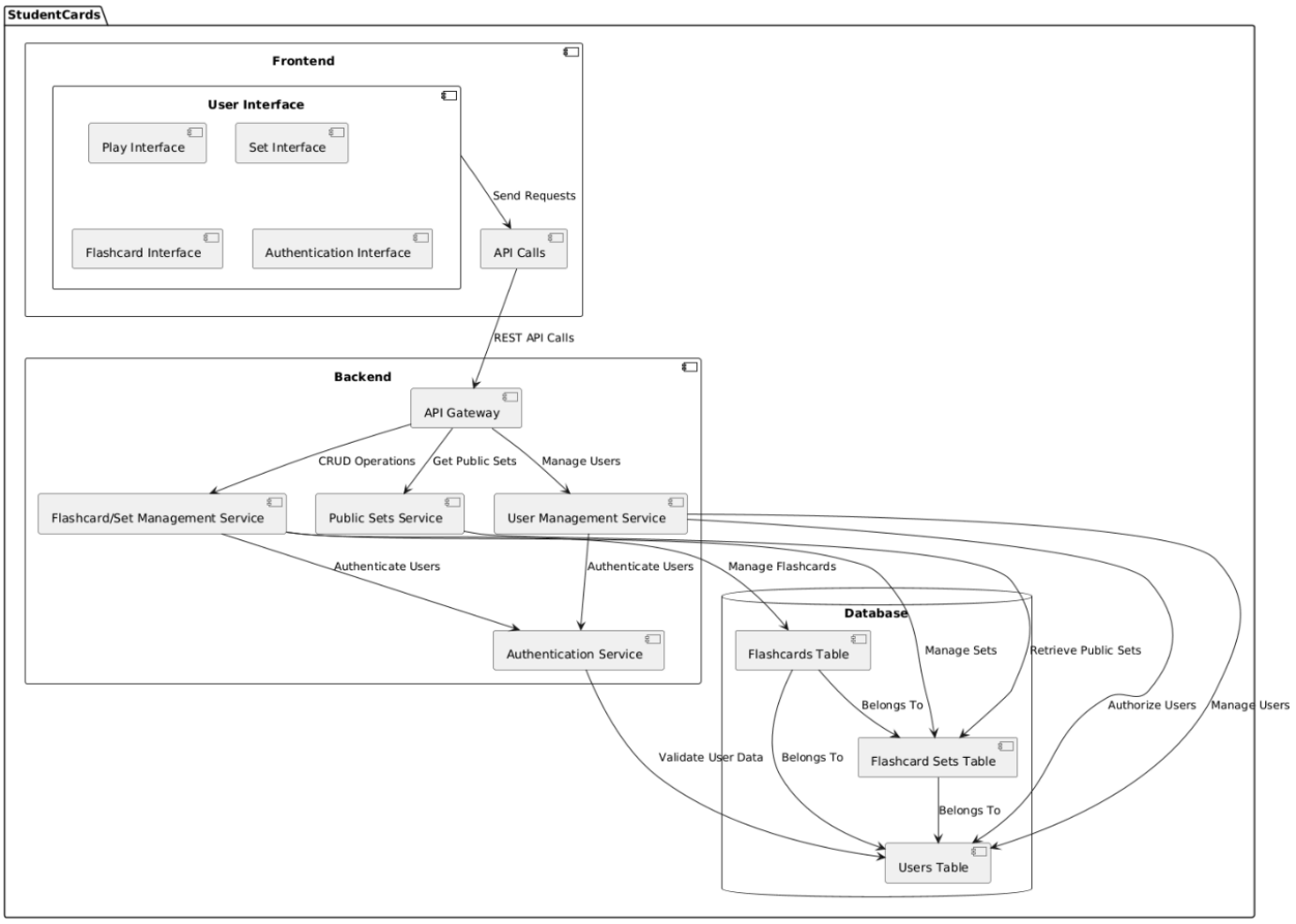
Model systemowy definiuje, jak procesy biznesowe są realizowane za pomocą technologii informatycznych:

1. Frontend: Interfejs użytkownika umożliwia korzystanie z funkcji aplikacji.
2. Backend: Bezpieczne przechowywanie danych użytkownika oraz serwisy obsługujące logowanie, zarządzanie fiszkami i grę.
3. Baza danych: struktura tabel wspierająca zarządzanie zestawami i fiszkami.

WIDOK 4 + 1

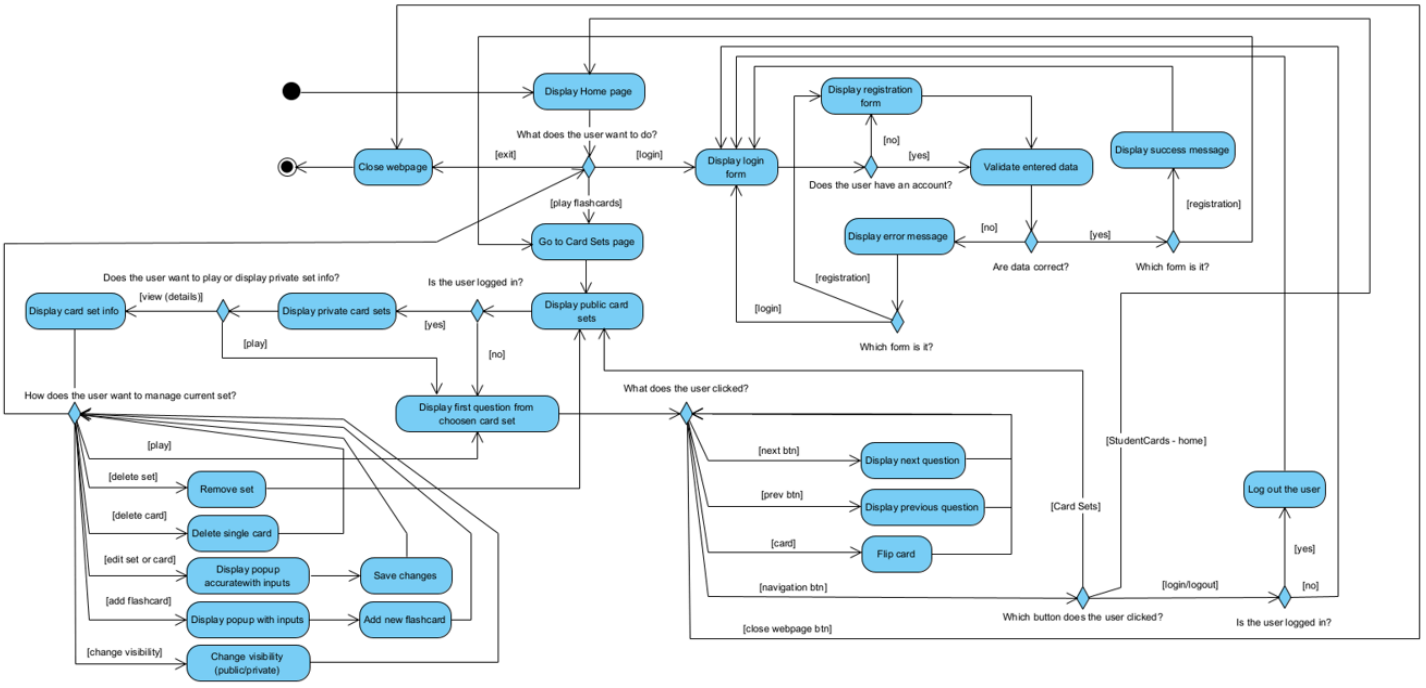
Widok deweloperski - diagram dekompozycji modułowej

Diagram przedstawia modułową dekompozycję systemu StudentCards, podzielonego na Frontend, Backend i Bazę danych. Moduł zarządzania fiszkami komunikuje się z bazą danych poprzez API, a wyniki są wyświetlane w interfejsie użytkownika. Frontend składa się z interfejsów użytkownika (Play, Set, Flashcard i Authentication), które wysyłają żądania REST API do backendu. Backend obejmuje bramkę API, która kieruje żądania do odpowiednich usług: zarządzania fiszkami, publicznymi zestawami oraz użytkownikami. Usługi współpracują z bazą danych, zawierającą tabele: Users, Flashcard Sets i Flashcards, służące do przechowywania danych o użytkownikach, zestawach i fiszkach. Usługa autoryzacji weryfikuje użytkowników, zapewniając bezpieczne zarządzanie danymi.



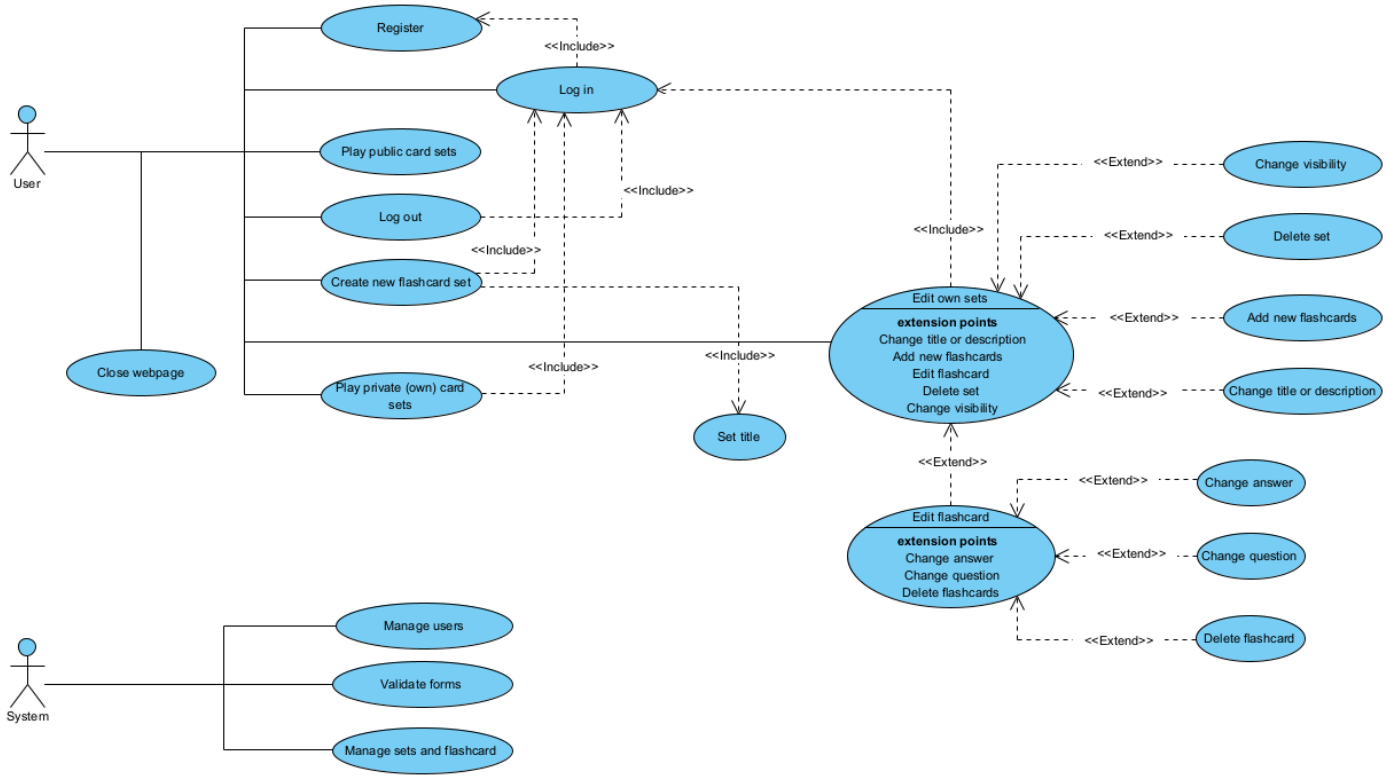
Widok procesów - diagram czynności

Diagram przedstawia przepływ aktywności użytkownika w systemie StudentCards. Rozpoczyna się od strony głównej. Użytkownik może zalogować się, zarejestrować, przeglądać zestawy kart (publiczne lub prywatne) lub grać w fiszki. Akcje obejmują edycję zestawów (dodawanie/usuwanie kart, zmiana widoczności), wybór pytania, przegląd kolejnych/poprzednich kart oraz ich odwracanie. Użytkownik może także wylogować się lub zamknąć stronę. Walidacja danych i odpowiednie komunikaty są wyświetlane w procesie rejestracji/logowania.



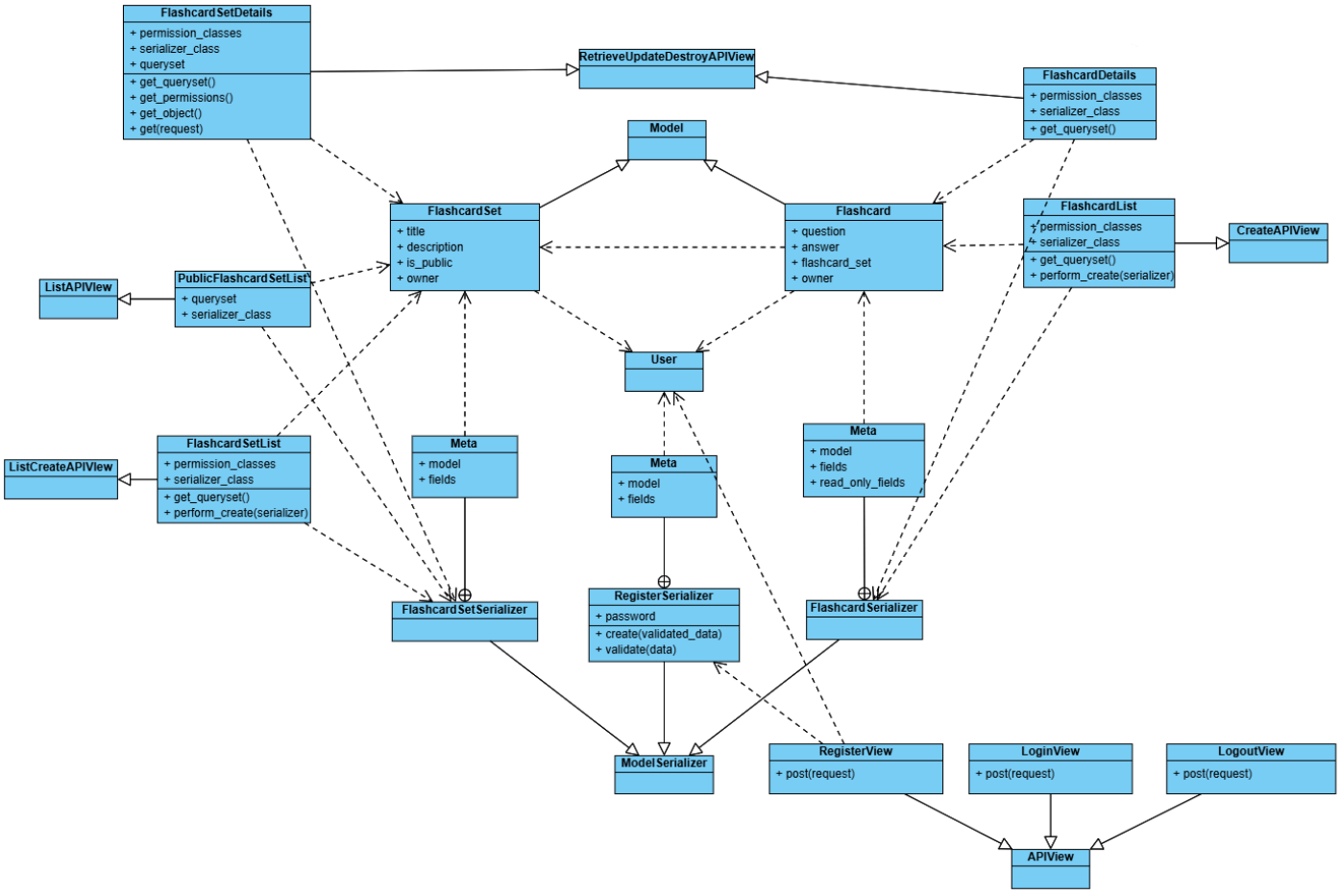
Widok scenariuszy - diagram przypadków użycia

Diagram przypadków użycia przedstawia interakcje użytkownika i systemu w aplikacji StudentCards. Użytkownik może rejestrować się, logować, grać w publiczne lub prywatne zestawy kart, tworzyć nowe zestawy oraz edytować istniejące (zmieniać tytuł, dodawać/usuwać fiszki, zmieniać widoczność). System wspiera zarządzanie użytkownikami, walidację formularzy oraz operacje na zestawach i fiszkach. Rozszerzenia obejmują edycję fiszek (zmiana pytań/odpowiedzi, usuwanie) oraz zarządzanie zestawami. Użytkownik może również wylogować się lub zamknąć stronę.



Widok logiczny - diagram klas

Diagram klas ilustruje strukturę aplikacji do zarządzania zestawami fiszek, pokazując relacje między modelami, serializerami i widokami API. FlashcardSet reprezentuje zestaw fiszek, który zawiera tytuł, opis, widoczność oraz właściciela, natomiast Flashcard składa się z pytania, odpowiedzi, powiązania z zestawem i właściciela. Model User przechowuje dane użytkownika i zarządza jego relacjami z fiszkami oraz zestawami. Serializer FlashcardSetSerializer i FlashcardSerializer konwertują dane na format JSON oraz je weryfikują, a RegisterSerializer zajmuje się rejestracją użytkowników. Widoki API, takie jak RetrieveUpdateDestroyAPIView i CreateAPIView, realizują operacje CRUD, podczas gdy LoginView i LogoutView odpowiadają za proces logowania. Relacje między klasami zapewniają spójne działanie aplikacji.



Widok fizyczny - opis tekstowy

1. Komponenty w warstwie fizycznej:
    - **Frontend (klient):**
        - Frontend działa po stronie użytkownika jako aplikacja webowa uruchamiana w przeglądarce.
        - Jest hostowany na serwerze HTTP
        - Komunikuje się z backendem przesyłając żądania HTTP do REST API i odbierając dane w formacie JSON.
    - **Backend (serwer aplikacji):**
        - Backend, zbudowany na Django REST Framework, obsługuje wszystkie operacje serwerowe.
        - Odpowiada za logikę biznesową, uwierzytelnianie, walidację danych i zarządzanie operacjami na bazie danych.
        - Działa na serwerze aplikacji, który jest połączony z frontendem i bazą danych.
        - Serwer aplikacji nasłuchuje na żądania HTTP z frontendu, przetwarza je i zwraca odpowiedzi.
    - **Baza danych:**
        - System korzysta z relacyjnej bazy danych SQLite.
        - Przechowuje trwałe dane, takie jak informacje o użytkownikach, zestawach fiszek oraz poszczególnych fiszkach.

2. Fizyczne połączenia pomiędzy komponentami:
    - **Frontend ↔ Backend:**
        - Połączenie pomiędzy frontendem a backendem odbywa się przez protokół HTTP.
        - Żądania są kierowane do REST API backendu w celu pobierania danych (GET), wysyłania danych (POST), aktualizacji (PUT) lub ich usuwania (DELETE).
    - **Backend ↔ Baza danych:**
        - Backend komunikuje się z bazą danych za pomocą ORM Django, który generuje zapytania SQL.
        - Połączenie odbywa się za pośrednictwem protokołu TCP/IP i jest zabezpieczone hasłami oraz konfiguracją użytkownika bazy danych.
        - Dane są przesyłane między backendem, a bazą danych w formie zapytań i odpowiedzi SQL.

MOŻLIWOŚĆ MODYFIKACJI PO WDROŻENIU

Zadbano o możliwość modyfikacji oprogramowania po wdrożeniu, stosując odpowiednie praktyki programistyczne i projektowe, które zapewniają elastyczność i łatwość rozwoju w przyszłości. Oto kluczowe aspekty, brane pod uwagę w procesie wytwarzania oprogramowania:

Zastosowanie REST API

Dzięki architekturze opartej na REST API, wszelkie zmiany w logice biznesowej (na backendzie) mogą być wdrażane bez konieczności zmiany kodu frontendowego. Na przykład, jeśli zmienia się struktura danych, można dostosować API do nowych wymagań, nie wpływając na interfejs użytkownika.

Oddzielenie frontendu od backendu

Ponieważ frontend (React) jest oddzielony od backendu (Django), modyfikacje w jednym z komponentów (np. aktualizacja wyglądu aplikacji lub zmiana algorytmów przetwarzania danych na backendzie) mogą być realizowane bez ingerencji w drugi komponent. Może to obejmować dodawanie nowych funkcji, zmianę interfejsu użytkownika, optymalizację wydajności, a także dodawanie nowych punktów końcowych API.

Modularna budowa aplikacji

Zarówno w React, jak i w Django, projekt został zorganizowany w sposób modularny, co pozwala na łatwą modyfikację lub rozbudowę poszczególnych części systemu. Na przykład, jeśli w przyszłości pojawi się potrzeba dodania nowych funkcji (np. systemu powiadomień lub analizy danych użytkownika), można to zrobić, rozszerzając odpowiednie moduły w aplikacji bez konieczności przepisywania całej aplikacji.

Śledzenie wersji i zarządzanie projektem

W projekcie wykorzystywane są narzędzia do kontroli wersji, takie jak Git, oraz platforma GitHub, które umożliwiają efektywne śledzenie zmian w kodzie. Dzięki GitHub można łatwo zarządzać repozytorium projektu, współpracować z innymi oraz zachować historię wszystkich modyfikacji. Użycie Gita pozwala na szybkie wprowadzanie zmian i cofanie się do wcześniejszych wersji, co zapewnia stabilność i kontrolę nad rozwojem aplikacji.