



POLSKO-JAPOŃSKA
AKADEMIA TECHNIK
KOMPUTEROWYCH

Gdańsk, 19 czerwca 2024

Kierunek studiów: Informatyka

Rodzaj studiów: Zaoczne

Praca dyplomowa

Temat pracy:

Fishki - multiplatformowa aplikacja służąca do nauki pamięciowej

Temat w języku angielskim:

Fishki - a multiplatform application for memory learning

Opiekun pracy:

dr Puźniakowski Tadeusz

Wykonawcy:

Nazwisko, imię	Nr albumu
Kossak Oliwier	s22018
Klimowski Daniel	s18504
Krieger Wiktor	s23638
Żurawski Jakub	s23047

Streszczenie:

Bieżący dokument stanowi prace dyplomową projektu Fishki, którego celem było wytworzenie aplikacji umożliwiającej naukę przez powtórki z wykorzystaniem nowoczesnych technologii. W tekście omówiono problematykę w oparciu o którą zdefiniowane zostały założenia projektu. Udokumentowane zostały: proces planowania, analiza otoczenia, definicja wymagań, implementacja, testy i prezentacja wypracowanego systemu. Uzyskanymi produktami jest aplikacja mobilna dla systemów Android i iOS oraz aplikacja webowa dostępna z poziomu przeglądarki internetowej. System działa w oparciu o chmurową infrastrukturę serwerową, obsługującą i utrzymującą działanie aplikacji.



POLSKO-JAPOŃSKA
AKADEMIA TECHNIK
KOMPUTEROWYCH

Karta projektu

Temat projektu: Fishki - multiplatformowa aplikacja służąca do nauki pamięciowej		Akronim: Fishki Data ustalenia tematu 11.10.2023
Promotor: dr Puźniakowski Tadeusz		Konsultanci: 1. — brak —
Cele projektu: Dostarczenie systemu do efektywnej nauki z wykorzystaniem techniki zapamiętywania w postaci fiszek. Cel projektu odpowiada na problem rozumiany jako trudności w organizacji oraz korzystania z materiałów służących do opanowywania pojęć.		
Rezultaty i zakres	Oczekiwane produkty/usługi: Strona internetowa, aplikacja mobilna, serwer obsługujący utrzymanie i żywotność produktu Główne funkcjonalności i/lub cechy: Nauka metodą fishek, obsługa talii tj. zestawów fiszek, obsługa talii fiszek za pomocą mowy, tworzenie fiszek z pomocą sztucznej inteligencji.	
Miary sukcesu: Wytworzenie działającej strony internetowej i aplikacji mobilnej, opracowanie techniczne projektu z wykorzystaniem infrastruktury serwerowej, zaimplementowanie silnika sztucznej inteligencji, zrealizowanie wymagań systemowych na poziomie ‘wymagane’.		
Ograniczenia: Zdalny charakter pracy nad projektem, budżet, brak doświadczenia w pracy nad projektem o zadanej złożoności, nauka i wykorzystanie nowych technologii.		

Wykonawcy	Numer albumu	Specjalizacja	Tryb studiów
Kossak Oliwier	s22018	Sztuczna Inteligencja	Niestacjonarny
Klimowski Daniel	s18504	Sztuczna Inteligencja	Niestacjonarny
Krieger Wiktor	s23638	Sztuczna Inteligencja	Niestacjonarny
Żurawski Jakub	s23047	Sztuczna Inteligencja	Niestacjonarny

Oświadczenie autorów pracy dyplomowej

Świadomi odpowiedzialności prawnej oświadczamy, że niniejszą pracę dyplomową w zakresie przedstawionym przez nasz zespół projektowy wykonaliśmy samodzielnie i nie zawiera ona treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczamy również, że praca w przedstawionym przez nas zakresie nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu ukończenia studiów wyższych.

Oświadczamy ponadto, że niniejsza wersja pracy dyplomowej jest identyczna z załączoną wersją elektroniczną.

Spis treści

1 Wstęp	9
1.1 O projekcie	9
2 Omówienie problemu	11
2.1 Czym są fiszki?	11
2.2 Przedstawienie problemu	12
2.3 Zalety nauki z wykorzystaniem metody fiszek	13
2.3.1 Krzywa zapominania Hermanna Ebbinghausa	13
2.3.2 Przykłady praktycznych zastosowań fiszek	15
2.3.3 Personalizacja fiszek	16
2.4 Słownik pojęć	16
3 Projekt w kontekście	17
3.1 Omówienie zakresu projektu	17
3.2 Proponowane rozwiązanie	17
3.3 Grupa docelowa	18
3.4 Analiza rozwiązań konkurencji	18
3.4.1 Quizlet	18
3.4.2 Fiszkontakteka	20
3.4.3 Gizmo	21
3.4.4 Audio Flashcards	22
3.5 Analiza szans oraz zalet względem konkurencyjnych rozwiązań	24
3.5.1 Szanse/Zalety względem konkurencyjnych rozwiązań	24
3.5.2 Wady względem konkurencyjnych rozwiązań	24
3.6 Społeczny aspekt projektu	24
3.6.1 Gamifikacja w kontekście produktu	25
3.6.2 Ryzyko związane z nauką przy pomocy systemu Fishki	25
3.6.3 Podsumowanie	26
4 Organizacja pracy	27
4.1 Harmonogram pracy	27
4.2 Wybrana metodyka	28
4.3 Zespół i podział obowiązków	29
4.4 Analiza ryzyka	30
5 Analiza wymagań	32
5.1 Diagramy	32
5.2 Wymagania ogólne	36
5.2.1 Moduł autoryzacji	36
5.2.2 Moduł zarządzania talią	36
5.2.3 Moduł uczenia	37
5.2.4 Moduł udogodnień	37
5.2.5 Moduł wspomagania tworzenia talii	38
5.3 Wymagania funkcjonalne	38
5.3.1 Rejestracja konta użytkownika	38
5.3.2 Logowanie do systemu	39
5.3.3 Wylogowanie z systemu	39

5.3.4	Edycja danych użytkownika	40
5.3.5	Usunięcie konta użytkownika	41
5.3.6	Tworzenie talii fiszek	41
5.3.7	Usunięcie talii fiszek	42
5.3.8	Edycja talii fiszek	42
5.3.9	Tryb uczenia się z talii fiszek	43
5.3.10	Sterowanie talią przy użyciu mowy	44
5.3.11	Import talii innych użytkowników	45
5.3.12	Tryb dark mode i light mode	45
5.3.13	Kalendarz śledzący aktywność	46
5.3.14	Weryfikacja konta użytkownika poprzez email	46
5.3.15	Tworzenie talii fiszek poprzez zeskanowanie dokumentu	47
5.3.16	Generowanie treści fiszki na podstawie słów kluczowych	47
5.3.17	Resetowanie hasła na konta	48
5.4	Interfejs z otoczeniem	48
5.5	Wymagania pozafunkcjonalne	49
5.5.1	Instrukcja korzystania z trybu sterowania głosem	49
5.5.2	Limit błędów dotyczących logowania	49
5.5.3	Dostępność systemu	50
5.5.4	Responsywność	50
5.5.5	Kompatybilność	50
5.5.6	Hashowanie haseł	51
5.6	Wymagania na środowisko docelowe	51
5.6.1	Przeglądarka	51
5.6.2	System Android 10 i iOS 16	51
5.6.3	Kontenery dockerowe	52
5.6.4	Baza danych	52
5.6.5	Python	52
5.6.6	Node.js	53
5.6.7	System operacyjny	53
5.7	Udziałowcy	54
6	Architektura projektu i użyte technologie	56
6.1	Narzędzia organizacji pracy	56
6.2	Aplikacja webowa	59
6.3	Aplikacja mobilna	60
6.4	Backend aplikacji	60
6.5	Baza danych	60
6.6	Infrastruktura serwerowa	61
6.7	Narzędzia programistyczne	61
7	Implementacja	62
7.1	Faza planowania	62
7.2	Faza implementacji	63
7.2.1	Sprint 1	63
7.2.1.1	Wykonane zadania:	63
7.2.2	Sprint 2	63
7.2.2.1	Wykonane zadania:	64
7.2.3	Sprint 3	64
7.2.3.1	Wykonane zadania:	65
7.2.4	Sprint 4	65
7.2.4.1	Wykonane zadania:	66
7.2.5	Sprint 5	66
7.2.5.1	Wykonane zadania:	67
7.2.6	Sprint 6	67
7.2.6.1	Wykonane zadania:	68
7.2.7	Sprint 7	69
7.2.7.1	Wykonane zadania:	69
7.3	Szczegóły implementacji	71

7.3.1	Rejestracja konta użytkownika	71
7.3.1.1	Backend	71
7.3.1.2	Aplikacja moblina i aplikacja webowa	73
7.3.2	Logowanie do systemu	74
7.3.2.1	Backend	74
7.3.2.2	Aplikacja mobilna i aplikacja webowa	75
7.3.3	Wylogowanie z systemu	76
7.3.3.1	Backend	76
7.3.3.2	Aplikacja mobilna i aplikacja webowa	77
7.3.4	Edycja danych użytkownika	78
7.3.4.1	Backend	78
7.3.4.2	Aplikacja mobilna i aplikacja webowa	81
7.3.5	Usunięcie konta użytkownika	82
7.3.5.1	Backend	82
7.3.5.2	Aplikacja moblina	83
7.3.5.3	Aplikacja webowa	84
7.3.6	Tworzenie talii fiszek	84
7.3.6.1	Backend	85
7.3.6.2	Aplikacja mobilna	85
7.3.6.3	Aplikacja webowa	86
7.3.7	Usunięcie talii fiszek	87
7.3.7.1	Backend	87
7.3.7.2	Aplikacja moblina	87
7.3.7.3	Aplikacja webowa	88
7.3.8	Edycja talii fiszek	89
7.3.8.1	Backend	89
7.3.8.2	Aplikacja moblina	90
7.3.8.3	Aplikacja webowa	91
7.3.9	Tryb uczenia się z talii fiszek	92
7.3.9.1	Backend	92
7.3.9.2	Aplikacja moblina	93
7.3.9.3	Aplikacja webowe	94
7.3.10	Sterowanie talią przy użyciu mowy	96
7.3.10.1	Backend	96
8 Testy		98
8.1 Testy integracyjne		98
8.2 Testy akceptacyjne		99
9 Prezentacja projektu		108
9.1 Strona internetowa		108
9.1.1 Logowanie		108
9.1.2 Rejestracja		109
9.1.3 Strona domowa		110
9.1.4 Profil użytkownika		111
9.1.5 Create deck		112
9.1.6 My Decks		114
9.1.7 Public decks		117
9.1.8 Ranking użytkowników i talii		118
9.1.9 Panel moderatora		120
9.2 Aplikacja mobilna		122
9.2.1 Logowanie		122
9.2.2 Rejestracja		123
9.2.3 Widok domowy		124
9.2.4 My Decks, My Downloaded Decks i Create Deck		125
9.2.5 Deck Preview i Downloaded Deck Preview		127
9.2.6 Review flashcards		129
9.2.7 Create Flashcard i Edit Flashcard		130
9.2.8 Learn with flashcards		132

9.2.9	Voice control	134
9.2.10	Memorized flashcards i Unmemorized flashcards	135
9.2.11	Settings	135
9.2.12	Public decks i Users ranking	137
9.2.13	Podgląd talii publicznej	138
9.2.14	Podgląd innego użytkownika	139
9.2.15	Profil	139
9.2.16	Widok narzędzi moderatora	141
10	Nakład pracy	142
10.1	Oliwier Kossak	143
10.2	Wiktor Krieger	145
10.3	Jakub Żurawski	147
10.4	Daniel Klimowski	149
11	Podsumowanie	152
11.1	Problemy i wyzwania	152
11.1.1	Pycharm Community Edition	152
11.1.2	Kontener bazy danych	153
11.1.3	Loader w aplikacji mobilnej	153
11.1.4	WSL i model nlp	153
11.1.5	Obsługa przy pomocy Mozilla Firefox	153
11.1.6	Api i HTTPS	153
11.2	Zmiany w trakcie realizacji	154
11.2.1	Limit logowań	154
11.2.2	Generowanie definicji przez ChatGTP	154
11.2.3	System operacyjny w infrastrukturze serwerowej	154
11.3	Porzucone pomysły	155
11.3.1	Dyktowanie treści fiszki	155
11.3.2	Sterowanie głosowe przez ChatGPT	155
11.3.3	Aktywacja konta i reset hasła	155
11.4	Przyszłość projektu	156
11.5	Zakończenie	156

Rozdział 1

Wstęp

1.1 O projekcie

Niniejszy projekt i praca podejmuje temat wytworzenia multiplatformowej aplikacji przeznaczonej do nauki metodą fiszek. Głównym celem projektu jest usprawnienie procesu nauki poprzez wykorzystanie nowoczesnych technologii.

Geneza projektu wywodzi się ze środowiska zespołu projektowego oraz otoczenia uczelnianego, bezpośredniego zapotrzebowania na aplikacje, która spełni wszystkie wymagane funkcjonalności, co skutkować będzie tym, iż proces nauki przez powtórki będzie skuteczny i wygodny. Te funkcjonalności oraz właściwości zostały zdefiniowane początkowo jako pomysł z osobistych doświadczeń członków zespołu z alternatywnymi aplikacjami. Później na podstawie przeprowadzonej w niniejszej pracy analizie konkurencyjnych rozwiązań, pomysł przeobraził się w ostateczną wizję produktu.

Wśród wielu aplikacji przeznaczonych do nauki przy pomocy różnych metod, ich wspólnym mianownikiem są regularne powtórki. Zdefiniowanie wymagań dla aplikacji tego typu nie jest możliwe bez dobrze sprecyzowanej problematyki związanej z zagadnieniem utrwalania wiedzy. Problematyka ta została omówiona w oparciu o badania podejmujące temat systematyki w nauce. W kolejnym rozdziale pracy przyjrzymy się bardziej szczegółowo kwestii związanej z tym, czy regularne powtarzanie materiału jest skuteczne i jak nowoczesne technologie mogą ten proces usprawnić.

Niniejsza praca także podsumowuje realizację podjętego projektu. Opisuje szczegółowo m.in. podjętą metodykę, organizację, podział i nakład pracy, analizę konkurencji, zdefiniowane wymagania, architekturę projektu, użyte technologie, implementację techniczną oraz testy.

Realizowany projekt ma na celu nie tylko zbudowanie użytkowej aplikacji, ale także przyczynienie się do zrozumienia, jak nowoczesne technologie mogą wspierać edukację i w jakim kierunku należy rozwijać dostępne narzędzia. Dlatego też obowiązkowo podejmujemy temat dalszego rozwoju oraz przyszłości wdrożonego systemu.

Rozdział 2

Omówienie problemu

2.1 Czym są fiszki?

Nauka z wykorzystaniem metody fiszek opiera się na systemie regularnych powtórek rozłożonych w czasie. Jest jednym z najpopularniejszych i najbardziej efektywnych sposobów pozwalających przywrócić i utrważyć wiedzę. Technika ta w klasycznej wersji polega na regularnym dobieraniu kart z talii, w której wszystkie są zapisane z dwóch stron – na jednej stronie każdej fiszki znajduje się zagadnienie, na drugiej, natomiast przeznaczone do zapamiętania klucz odpowiedzi lub definicja. Karty są dobierane z talii kolejno w sposób losowy, po każdorazowym dobraniu należy odczytać zagadnienie lub kłopotliwy termin, a następnie spróbować odpowiedzieć zgodnie z kluczem zapisanym na drugiej stronie. Wtedy dopiero można dokonać porównania i zestawić swoją odpowiedź z prawidłową, zapisaną na odwrocie fiszki. Następnie, karta powinna wrócić do talii, tj. zbioru.

Rozwój technologiczny oraz dobiegająca zewsząd cyfryzacja powoli wypierają z użytku klasyczną formę stosowania fiszek na rzecz jej organizacji poprzez aplikacje mobilne oraz strony internetowe. Zmiana ta niesie wiele szans oraz możliwości udoskonalenia metody fiszek w jej skuteczności oraz dostępności. Fundamentalną zmianą jest samo przyśpieszenie całego procesu sporządzania talii oraz ich przeglądania – przy wykorzystaniu komputera lub smartfonu staje się to znacznie prostsze i szybsze niż ręczne zapisywanie kart, wycinania ich i gromadzenie.

2.2 Przedstawienie problemu

Celem projektu jest zwiększenie efektywności uczenia się metodą fiszek poprzez jej integrację z interfejsem bezdotykowym. Pomimo dużego wkładu w przystępcość i zwiększenie dostępności tej metody przez cyfrowe rozwiązania, istniejące aplikacje do tworzenia i przeglądania fiszek wymagają zasadniczo aktywnego zaangażowania użytkownika poprzez klikanie lub wpisywanie na klawiaturze – zarówno przy użyciu smartfonów, jak i komputerów osobistych. Ta konieczność bezpośredniej interakcji ogranicza możliwości nauki do określonych sytuacji, co stanowi barierę w momentach, gdy użytkownik wykonuje inne czynności, na przykład proste prace domowe, a urządzenie obsługujące aplikacje nie może być efektywnie wykorzystane.

Dodatkowo proces tworzenia fiszek, który wymaga od użytkowników manualnego redagowania, przepisywania lub kopiowania definicji, znacząco opóźnia przygotowanie materiałów do nauki i wydłuża czas potrzebny na stworzenie talii kart. Ta czasochłonna procedura potrafi stanowić istotną przeszkodę, ograniczając proces uczenia się. Czas, który mógłby być przeznaczony na utrwalanie wiedzy, jest poświęcany na przygotowanie zasobów służących do nauki. Sama potrzeba redagowania fiszek może odebrać użytkownikowi chęci by w ogóle korzystać z tej metody. Celem projektu jest częściowa automatyzacja procesu tworzenia talii poprzez wzbogacenie funkcjonalności o narzędzia sztucznej inteligencji. Skutkiem tego jest możliwość generowania kluczowych odpowiedzi fiszek odpowiednio do podanych zagadnień.

W odpowiedzi na te wyzwania, opisany w niniejszej pracy system dąży do zminimalizowania ograniczeń związanych z tradycyjną obsługą aplikacji oferujących możliwość nauki metodą fiszek. Podjęty projekt proponuje rozwiązań, które integruje proces przyswajania wiedzy z codziennymi czynnościami użytkownika oraz umożliwia jednocześnie szybsze i bardziej intuicyjne tworzenie materiałów do nauki. Celem realizowanego projektu jest stworzenie aplikacji, w której nauka jest nie tylko bardziej efektywna, ale również dostępna w szerokim zakresie sytuacji z pomocą bezdotykowej obsługi.

Poniżej znajduje się graficzna reprezentacja problemu ujętego w formie Rich Picture:



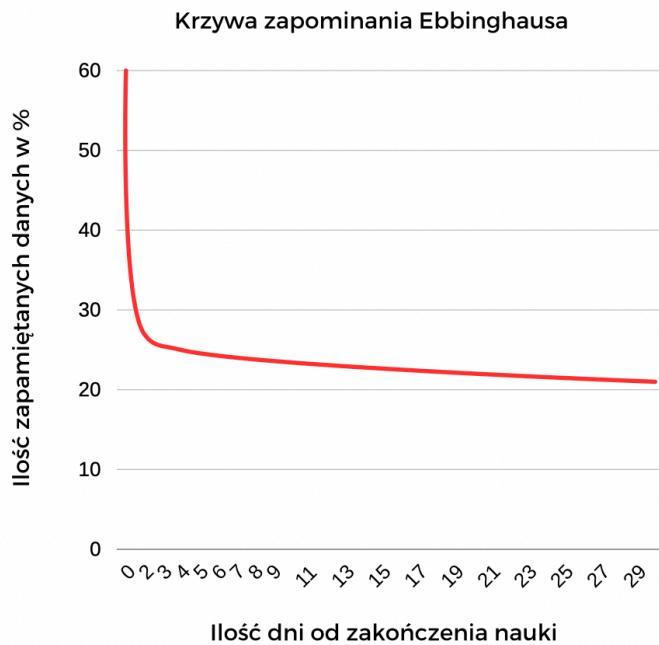
Rysunek 2.1: Problem przedstawiony w rich picture.

2.3 Zalety nauki z wykorzystaniem metody fiszek

Nauka z wykorzystaniem fiszek to jedna z najbardziej popularnych i elastycznych metod. Dzięki jej prostocie i skuteczności znajduje zastosowanie w różnych kontekstach edukacyjnych. W poniższym podroziale szczegółowo przyjrzymy się mechanizmowi, dzięki któremu fiszki tak skutecznie wspierają proces nauki.

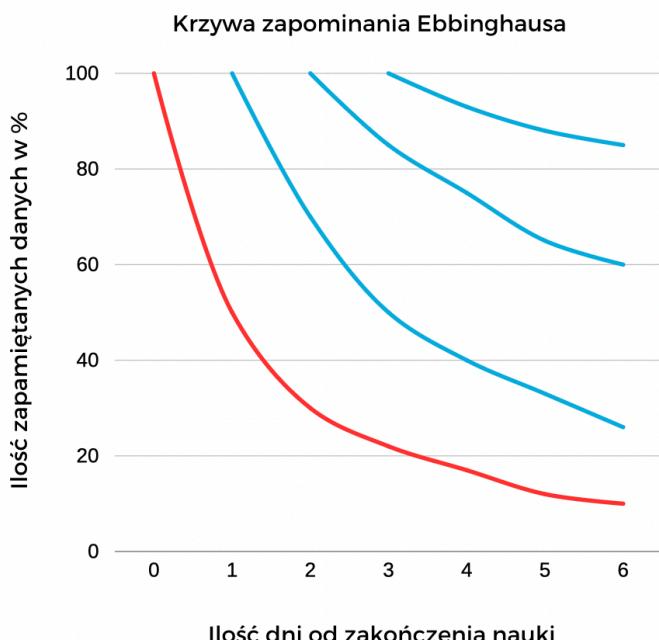
2.3.1 Krzywa zapominania Hermanna Ebbinghausa

Opierając się na systemie regularnych powtórek, praktykowana metoda fiszek pozwala na utrzymanie stanu wiedzy na wysokim poziomie zgodnie z zasadami krzywej zapominania Ebbinghausa. Mowa o zależności opisanej przez niemieckiego psychologa, Hermanna Ebbinghausa. Według wyników badań opublikowanych w 1885 roku w pracy "Uber das Gedächtnis"[1] (w tłumaczeniu „O pamięci”), najlepszym sposobem na zapamiętanie jest rozłożenie nauki w czasie. Praktyka ta jest znacznie skuteczniejsza od skupienia nauki w jednym okresie. Istotnym elementem zagadnienia jest wykres omawianej krzywej:



Rysunek 2.2: Wizualizacja wyniku badań Ebbinghausa.

Wykres z rysunku 2.2 reprezentuje tempo, w którym następuje zapominanie nauczonych treści po zakończeniu nauki. W przeciągu pierwszych kilku dni od jej zakończenia następuje gwałtowny spadek zapamiętanych informacji, nawet do poziomu 25% względem całkowitej ilości powtarzanych informacji. Dopiero po upływie tego czasu krzywa się stabilizuje, a tempo zapominania znacznie zwalnia. Najskuteczniejszym według badania sposobem by zapobiec temu procesowi jest regularne powtarzanie materiału. Poniższy wykres przedstawia ilość przyswojonej wiedzy względem każdej kolejnej sesji nauki:



Rysunek 2.3: Wizualizacja wyniku badań Ebbinghausa.

Przyjmuje się, że metoda nauki z regularnymi powtórkami, jaką oferuję metoda fiszek, pozwala na skuteczne utrwalenie wiedzy i znaczne spowolnienie zapominania jej. Użyte w badaniu informacje, które zostały podjęte nauce były danymi „bezużytecznymi”. Hermann Ebbinghaus celowo używał losowo utworzonych sylab, aby zminimalizować zależności między danymi a osobami, które się uczyły. Powszechnie akceptowane w psychologii edukacyjnej jest twierdzenie, że przyswojenie informacji, które indywidualnie uznaje się za jakikolwiek interesujące dodatkowo ułatwia proces ich zapamiętania. Idące za tym wnioski plasują metodę regularnych powtórek jako tym bardziej przystępna, ponieważ znajduje zastosowanie nawet w przypadku zapamiętywania informacji, które niekoniecznie muszą być dla odbiorcy atrakcyjne.

2.3.2 Przykłady praktycznych zastosowań fiszek

Metoda nauki z wykorzystaniem fiszek znajduje szerokie zastosowanie w różnych dziedzinach nauki i życia codziennego, jej przystępna forma odpowiada poniższym przykładom:

- Nauka języków obcych: fiszki są powszechnie stosowane w celu utrwalenia słownictwa, zwrotów lub zasad gramatycznych języków obcych; szybkie przypominanie i testowanie wiedzy jest w tym przypadku kluczowe w procesie opanowywania języka innego niż ojczysty;
- Nauki techniczne: ze względu na dużą ilość szczegółowych informacji, które istotnie trzeba spać, fiszki są odpowiednim narzędziem dla studentów lub pracowników branży technicznych. Pomagają one w efektywnym przyswajaniu skomplikowanych terminów;
- Przygotowanie do egzaminów: uczniowie i studenci przygotowujący się do egzaminów mogą wykorzystać fiszki w celu opanowania kluczowych pojęć, definicji czy zestawu odpowiedzi.

2.3.3 Personalizacja fiszek

Do najważniejszych zalet nauki metodą fiszek zalicza się możliwość personalizacji materiałów naukowych. Osoby korzystające z fiszek mogą tworzyć własne talie, dostosowane do ich indywidualnych potrzeb i preferencji. Dodatkowo fiszki mogą być łatwo dostosowane do różnych sposobów uczenia się, przykładowo poprzez stosowanie trybu obsługi głosowej.

2.4 Słownik pojęć

1. Fiszka - karta zapisana z dwóch stron, na jednej stronie znajduje się pytanie/zagadnienie, na drugiej odpowiedź/definicja.
2. Talia - zbiór, pula fiszek.
3. System - przedmiot projektu, ogólne określenie wytworzonej aplikacji mobilnej, webowej.
4. Gamifikacja/Grywalizacja - wykorzystanie elementów gier w niezwiązanym z grami kontekście [2].
5. Model nlp - zastosowany w projekcie moduł rozpoznawania i analizy instrukcji głosowych.
6. Podobieństwo kosinusowe - miara używana do określenia podobieństwa między dwoma wektorami poprzez obliczenie kąta między nimi [**kosinus**].

Rozdział 3

Projekt w kontekście

3.1 Omówienie zakresu projektu

W odpowiedzi na zdefiniowany i omówiony w Rozdziale II problem, niniejszy projekt zakłada wytworzenie aplikacji edukacyjnej, która umożliwi naukę metodą wykorzystującą fiszki. Projekt obejmuje dostarczenie aplikacji webowej dostępnej z poziomu przeglądarki internetowej oraz aplikacji mobilnej dla urządzeń z systemem Android lub iOS. W zakres prac wlicza się także zbudowanie pełnej infrastruktury wspierającej aplikację - backend oraz serwer utrzymujący cały system.

3.2 Proponowane rozwiązanie

Projekt bazuje na kilku kluczowych założeniach, które mają na celu bezpośrednie rozwiązanie problemów opisanych w Rozdziale II:

- Obsługa głosowa: integracja obsługi głosowej aplikacji ma ułatwić korzystanie z niej w sytuacjach, w których użytkownik ma ograniczone możliwości fizycznej obsługi urządzenia. Dzięki temu nauka jest możliwa w każdych warunkach, co odpowiada na problem ograniczonej dostępności tradycyjnych metod nauki;
- Narzędzie AI do tworzenia fiszek: udostępnienie narzędzia umożliwiającego automatyczne generowanie i dobieranie definicji/odpowiedzi do zagadnień w fiszkach. Pozwala przyśpieszyć i uprzystępnić proces redagowania fiszek.

3.3 Grupa docelowa

Główną grupę użytkowników stanowią uczniowie i studenci, a także osoby, które nie mają czasu na tradycyjną formę przyswajania wiedzy, na przykład czytanie lub naukę przy biurku w domowym zaciszu. Generowanie definicji przy użyciu programu ChatGPT pozwala na szybkie tworzenie zestawów fiszek, co może być bardzo wygodne dla osób ceniących sobie czas. Głosowa obsługa talii fiszek pozwala na wykonywanie sesji nauki w warunkach niesprzyjających innym formom uczenia się. Użytkownik ma

możliwość powtórzenia materiału, wykonując inne obowiązki, takie jak gotowanie, sprzątanie lub inne czynności, którym nie trzeba poświęcać większej uwagi.

3.4 Analiza rozwiązań konkurencji

Ważnym aspektem tworzenia nowego produktu lub usługi jest analiza konkurencji. Dostrzeżenie wad i zalet konkurencyjnych rozwiązań pozwala na ulepszenie produktów dostępnych na rynku i dotarcie do nowej grupy klientów.

3.4.1 Quizlet

Jedna z pierwszych dostępnych na rynku aplikacja do fiszek. Utworzona w 2005 roku narzędzie do nauki zgromadziło ogromną bazę użytkowników szacowaną na około 50 milionów. Quizlet posiada stronę internetową i aplikacje mobilne dostępne na Android i iOS.



Rysunek 3.1: Aplikacja mobilna Quizlet.

Zalety:

- Możliwość tworzenia testów w różnych trybach: prawda/fałsz, wielokrotny wybór, pisemny.
- Odczytanie przez lektora treści fiszki poprzez kliknięcie przycisku.
- Dostęp do materiałów, które zostały zweryfikowane przez ekspertów.
- Personalizacja fiszek za pomocą obrazów i dźwięków.

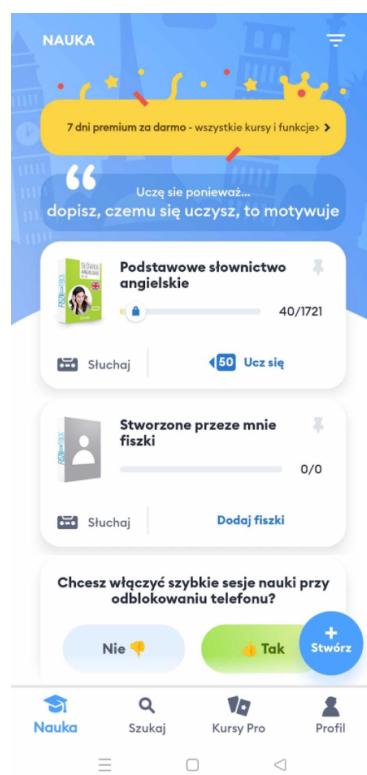
- Pozwala na wyszukiwanie talii innych użytkowników.

Wady:

- Reklamy w aplikacji.
- Pełny dostęp do wszystkich funkcjonalności tylko za opłatą.
- Brak rankingu popularności talii użytkowników, co pozwalałoby na łatwy dostęp do innych talii.
- Brak trybu sterowania talii głosem.

3.4.2 Fiszkontakte

Polski portal do nauki metodą fiszek ukierunkowany na naukę języków obcych. Oprócz strony internetowej aplikacja dostępna na system Android lub iOS.



Rysunek 3.2: Aplikacja mobilna Fiszoteka.

Zalety:

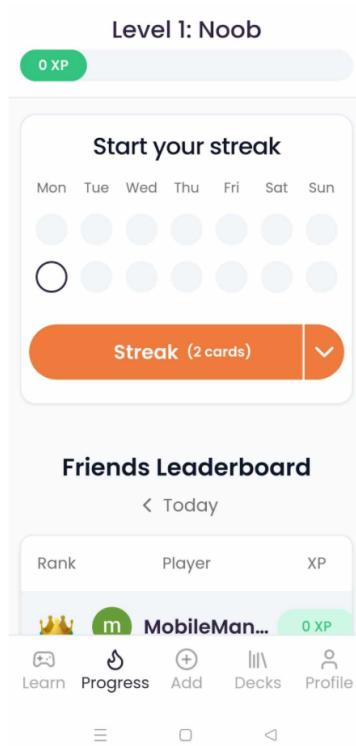
- Dostępne talie językowe o różnych poziomach trudności.
- Odczytanie przez lektora treści fiszek po zobaczeniu odpowiedzi.
- Tryb słuchania, tytuł i treść fiszek są odczytywane po kolej po lektora.
- Dostęp do publicznych talii.
- Można pobrać dokument PDF lub Mp3 z fiszek.

Wady:

- Pełny dostęp do wszystkich funkcjonalności tylko za opłatą.
- Dla bezpłatnej wersji ograniczenie liczby utworzonych fiszek do 300.
- Reklamy w aplikacji.
- Lektor w bezpłatnej wersji ograniczony do 20 dźwięków dziennie.
- Brak trybu sterowania talii głosem.
- Aplikacja ukierunkowana na uczenie się słówek.

3.4.3 Gizmo

Aplikacja wykorzystuje narzędzia sztucznej inteligencji, które pozwalają na tworzenie zestawów fiszek na podstawie plików PDF, CSV lub filmów na platformie YouTube. Gizmo posiada zarówno stronę internetową, jak i aplikację mobilną na Androida i iOS.



Rysunek 3.3: Aplikacja mobilna Gizmo.

Zalety:

- Brak reklam, brak opłat.
- Zaawansowane zastosowanie AI do tworzenia fiszek.
- Automatyczne tłumaczenie skanu na język angielski.

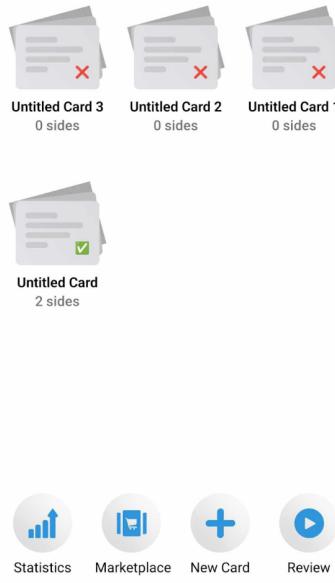
- Szybki i prosty import fiszek z różnych źródeł.
- Kalendarz passy (streak) zachęcający do nauki.
- Przejrzysty interfejs.
- Dostęp do publicznych talii.

Wady:

- Nieuumiejętnie użycie AI przy tworzeniu, prowadzi do utworzenia bezsensownych talii.
- Dużo niezagospodarowanej przestrzeni na ekranie w trybie uczenia.
- Brak rankingu popularności talii użytkowników.
- Domyslnie irytująca częstotliwość powiadomień.
- Brak trybu sterowania talii głosem.

3.4.4 Audio Flashcards

Aplikacja dostępna tylko w wersji mobilnej. Ukierunkowana na uczenie się ze słuchu i tworzenie talii z użyciem mowy. Nieintuicyjny interfejs sprawia, że aplikacja jest nieprzyjazna dla nowych użytkowników.



Rysunek 3.4: Aplikacja mobilna Flashcards.

Zalety:

- W trybie uczenia pozwala na korzystanie bez dotykania.
- Możliwość tworzenia fiszek poprzez dyktowanie.

- Brak reklam, brak opłat.

Wady:

- Bardzo archaiczny interfejs.
- Nieintuicyjny interfejs.
- Skromna instrukcja korzystania.
- W trybie uczenia można tylko słuchać i czekać na timer, nie ma nasłuchiwanego hasła/komend.
- Brak publicznych talii.

3.5 Analiza szans oraz zalet względem konkurencyjnych rozwiązań

Podrozdział zawiera spis najważniejszych funkcjonalności, które odróżniają aplikację fiszki od innych produktów dostępnych na rynku.

3.5.1 Szanse/Zalety względem konkurencyjnych rozwiązań

Zalety:

- Sterowanie głosowe talii.
- Generowanie treści fiszki, wykorzystując ChatGPT.
- Ranking najpopularniejszych talii.

3.5.2 Wady względem konkurencyjnych rozwiązań

Wady:

- Aplikacja dostępna tylko w języku angielskim uniemożliwia uczenie się języków obcych.
- Brak narzędzi generowania zestawów talii z różnych formatów plików takich jak: CSV, mp4, PDF.
- Brak możliwości generowania testów wielokrotnego wyboru lub prawda/fałsz.
- Nie można eksportować talii do innych formatów plików takich jak PDF lub CSV.

3.6 Społeczny aspekt projektu

W tej sekcji omówiono społeczne aspekty projektu oraz wpływ, jaki potencjalnie może mieć korzystanie z aplikacji na użytkowników - tak w aspekcie jednostkowym, jak i społecznym. Przedstawiono, jak elementy grywalizacji mogą zwiększyć zaangażowanie w naukę oraz jakie ryzyka wiążą się z użyciem technologii w edukacji. Ponadto omówiono zagrożenia wynikające z automatyzacji tworzenia treści edukacyjnych, w szczególności w kontekście jakości i wiarygodności przyswajanych informacji oraz ich wpływu na rozwój umysłowy użytkowników.

3.6.1 Gamifikacja w kontekście produktu

Została przeprowadzona analiza związana z wpływem rywalizacji na aktywność użytkownika. Wynika z niej, że elementy rankingowe mają wpływ na zwiększoną aktywność użytkowników, z tego powodu aplikacja zawiera ranking popularności użytkowników i talii [3]. Pozycja w rankingu użytkownika jest wyliczana na podstawie sumy wszystkich pobrań talii upublicznionych dla innych odbiorców systemu. Ranking talii to zestawy fiszek o największej liczbie pobrań. System rankingowy ma na celu wzbudzenie rywalizacji u użytkowników, co ma ich sprowokować do większej aktywności. System punktów zachęca także odbiorców systemu do tworzenia nowych zestawów talii i dzielenia się z nimi w celu zwiększenia swojej szansy na poprawienie swojej pozycji rankingowej.

3.6.2 Ryzyko związane z nauką przy pomocy systemu Fishki

System fiszki ma możliwość generowania definicji na podstawie słowa, wykorzystując ChatGPT. Taka funkcjonalność niesie ze sobą różne zagrożenia związane z rozwojem umysłowym odbiorcy. Korzystanie z generowania treści może doprowadzić do rozleniwienia użytkownika. Odbiorca, wykorzystując ChatGPT do tworzenia definicji może nie podejmować żadnej refleksji, czy dostarczona treść ma jakikolwiek sens. W przypadku, w którym użytkownik nie jest zaznajomiony z tematem lub zagadnieniami może doprowadzić do sytuacji, w której wygenerowana treść będzie niepoprawna, ale odbiorca ze względu na brak wiedzy nie będzie tego świadomy. ChatGPT potrafi generować nieprawdziwe informacje co może spowodować nauczenie się przez użytkownika zmyślonych treści. Warto też podkreślić, że informacje przekazywane do czatu są wykorzystywane do uczenia modeli, jeżeli użytkownik przy generowaniu treści wykorzysta dane wrażliwe trafią one do twórców narzędzia [4]. Sytuacja, gdy użytkownik jest zaznajomiony z materiałem do nauki i jest w stanie ocenić jakość generowanych treści też niekoniecznie musi być lepsza, ponieważ odbiorca, tworząc samemu treści fiszek utrwała sobie zagadnienia. Bezmyślne zapamiętywanie informacji to kolejne zagrożenie, jakie może pojawić się w czasie nauki nowego materiału. Uczenie się na pamięć treści bez dogłębniego zrozumienia ich, sprawia, że zagadnienie jest krócej pamiętane przez użytkownika, a także doprowadza do sytuacji, w której użytkownik nie rozumie i nie wie jak zastosować przyswojone informacje. Powyższe problemy występują także w momencie, w którym użytkownik korzysta z zestawu fiszek utworzonych przez kogoś innego.

3.6.3 Podsumowanie

Podsumowując, projekt ma na celu stworzenie aplikacji edukacyjnej wykorzystującej nowoczesne technologie do efektywnej nauki za pomocą fiszek. Dzięki integracji z narzędziami AI i obsługą głosową, aplikacja oferuje użytkownikom wygodną i dostosowaną do ich potrzeb metodę nauki przez powtórki, dostępną zarówno z poziomu aplikacji mobilnej, jak i przeglądarki internetowej.

Rozdział 4

Organizacja pracy

4.1 Harmonogram pracy

Harmonogram pracy nie został określony na wstępie projektu. Na początku prac, koncentracja skupiła się głównie na etapie planowania. Początkowym priorytetem było wypełnienie dokumentacji projektowej, zebranie informacji oraz zdefiniowanie założeń i wymagań. Druga faza projektu poświęcona była implementacji przy równoczesnym redagowaniu książki dyplomowej. Postępy w produkcji oraz zarządzanie projektem monitorowane były za pomocą systemu Jira.

Październik 2023 - Styczeń 2024

Planowanie			
Jakub Żurawski	Daniel Klimowski	Oliwier Kossak	Wiktor Krieger
Analiza wymagań	Analiza wymagań	Analiza wymagań	Analiza wymagań
Dobór odpowiednich technologii i narzędzi pracy			
Wypełnienie dokumentacji techniczne	Wypełnienie dokumentacji techniczne	Wypełnienie dokumentacji techniczne	Wypełnienie dokumentacji techniczne
Określenie celów i zakresu projektu			

Utworzenie diagramów	-	Utworzenie mocków aplikacji	Utworzenie mocków i ikon aplikacji
----------------------	---	-----------------------------	------------------------------------

Tabela 4.1: Harmonogram planowania.

Luty 2024 - Czerwien 2024

Implementacja			
Jakub Żurawski	Daniel Klimowski	Oliwier Kossak	Wiktor Krieger
Implementacja backendu	Implementacja aplikacji mobilnej	Implementacja backendu	Implementacja aplikacji webowej
Implementacja aplikacji mobilnej	Konfiguracja serwera	Implementacja aplikacji webowej	Pisanie książki
Pisanie książki	Pisanie książki	Pisanie książki	-

Tabela 4.2: Harmonogram implementacji.

4.2 Wybrana metodyka

Zastosowaną w projekcie metodyką jest **model przyrostowo-ewolucyjny**. Stanowi on połączenie dwóch modeli: przyrostowego oraz ewolucyjnego. Proces tworzenia projektu został podzielony na fazę planowania oraz fazę implementacji. Faza planowania obejmowała zbieranie wymagań, wypełnianie dokumentacji oraz tworzenie mockupów aplikacji. Zrealizowano ją w podejściu ewolucyjnym, co pozwoliło na adaptację wymagań podczas tworzenia wspólnej wizji projektu [5]. Faza implementacji koncentrowała się na technicznej realizacji projektu, tj. na tworzeniu aplikacji. Odbywała się w trakcie sprintów, z których każdy był planowany po zakończeniu poprzedniego. Zalety takiego podejścia to przede wszystkim łatwość podejmowania działań nad rozwojem aplikacji, wczesna i stopniowa dystrybucja oprogramowania, stały wgląd w wygląd oraz funkcjonowanie aplikacji, a także możliwość nanoszenia poprawek w trakcie produkcji w związku ze zmianami wymagań.

4.3 Zespół i podział obowiązków

Daniel Klimowski:

- Budowanie aplikacji mobilnej;
- Konfiguracja Azure;

- Wypełnianie dokumentacji technicznej.

Oliwier Kossak:

- Budowa aplikacji webowej;
- Budowa aplikacji backendowej;
- Tworzenie modeli sql;
- Wypełnianie dokumentacji technicznej;
- Mockupy aplikacji mobilnej.

Wiktor Krieger:

- Budowa aplikacji webowej;
- Mockupy aplikacji webowej i mobilnej;
- Wypełnianie dokumentacji technicznej;
- Tworzenie ikon dla aplikacji.

Jakub Żurawski:

- Budowa aplikacji mobilnej;
- Budowa aplikacji backendowej;
- Wypełnianie dokumentacji technicznej;
- Tworzenie diagramów;
- Tworzenie modeli sql.

4.4 Analiza ryzyka

Podrozdział przedstawia analizę ryzyka, która ma na celu określenie i zrozumienie potencjalnych zagrożeń, które mogą wpływać na proces tworzenia projektu.

Zidentyfikowane ryzyko [20]	Symptomy	Środki / Działania zapobiegawcze i szacowany poziom trudności ich wdrożenia	Środki / Działania minimalizujące wpływ na projekt – już po jego wystąpieniu i szacowany poziom trudności ich wdrożenia (1-10)	Ranga ryzyka (im niższa, tym mniejszy negatywny wpływ na projekt)	Prawdopodobieństwo wystąpienia (1-100%)

Błędy w kodzie [O]	Błędy w komplikacji, wyniki testów nie pokrywają się z oczekiwanyym rezultatem	Szybka identyfikacja błędów, wykonywanie testów oprogramowania	Poprawa kodu, testowanie na bieżąco nowo implementowanych funkcjonalności (4)	10	80%
Awaria sprzętu deweloperskiego [S]	Sprzęt przestaje działać, brak możliwości odzyskania danych	Częste commits na repozytorium	Działanie na ostatniej wersji z repozytorium (1)	7	10%
Niedobór umiejętności w zespole [L]	Problemy jednostek w poszczególnych zadaniach	Uzupełnianie wiedzy	Pomoc innych członków zespołu (2)	7	70%
Niezgodność czasowa zespołu [C]	Osoba blokuje postęp nad projektem poprzez odpowiedzialność nad kluczowym elementem	Wcześniejszego zaplanowanie pracy	Wspólna praca nad kluczowym elementem zajęcie się zadaniami niepowiązanymi (5)	6	80%
Wybór nieodpowiedniej technologii [T]	Planowane rozwiązania nie są osiągalne przez wykorzystywane technologie	Dogłębna analiza potrzebnych technologii i ich możliwości/ograniczeń	Dopasowanie alternatywnych możliwych rozwiązań (8)	5	50%
Niedoszacowanie budżetu potrzebnego do utrzymania infrastruktury [B]	Budżet zbliża się do wyczerpania przed zaplanowanym terminem	Analiza cenników wykorzystywanych produktów	Próba pozykowania inwestorów (10)	10	80%
ChatGPT 3.5 generuje bezsensowną treść dotyczącą zagadnienia o której został zapytany przez użytkownika [F]	Generowane treści nie nawiązują do zagadnień, o które ChatGPT został zapytany	Próba sprawdziania zapytania, które zostało przesłane do ChatGPT w celu wygenerowania konkretniejszej treści	Użytkownik może zaakceptować lub odrzucić wygenerowaną treść. (4)	7	50%

Przeglądarka niepoprawnie wczytuje stronę internetową [S]	Style strony nie wyglądają tak samo jak na stronie uruchomionej lokalnie lub na innych przeglądarkach. Pozycje elementów na stronie są inaczej rozmieszczone.	Próba dostosowania stylów lub funkcjonalności do konkretnej przeglądarki.	Uruchomienie strony internetowej na innej przeglądarce (3).	10	70%
---	---	---	---	----	-----

Tabela 4.3: Analiza ryzyka.

Rozdział 5

Analiza wymagań

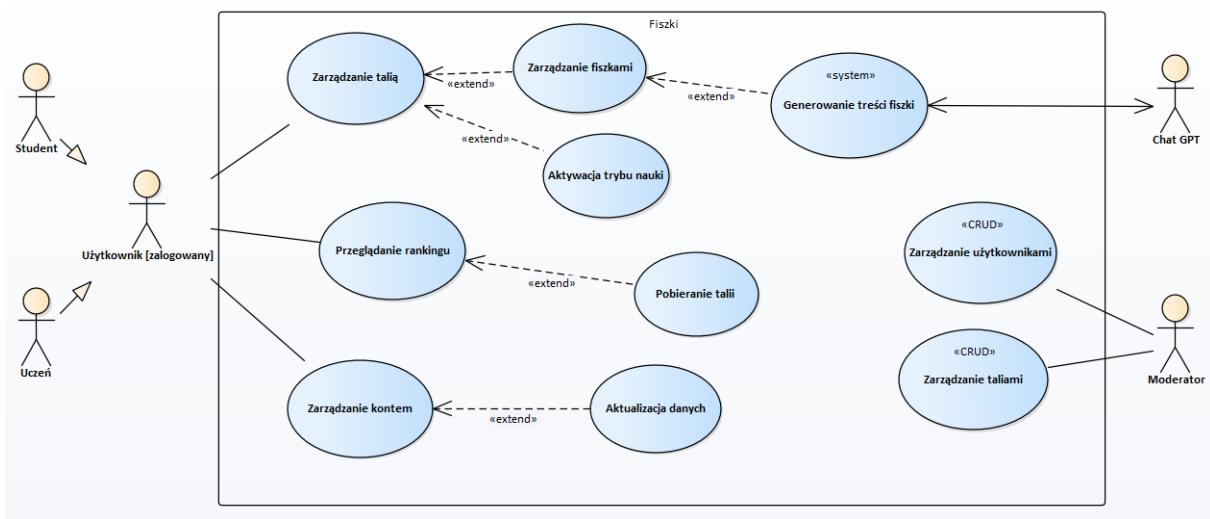
5.1 Diagramy

Diagram przypadków użycia jest narzędziem służącym do przedstawienia funkcjonalności systemu z punktu widzenia użytkownika.

Na diagramie z rysunku 5.1 przedstawiono:

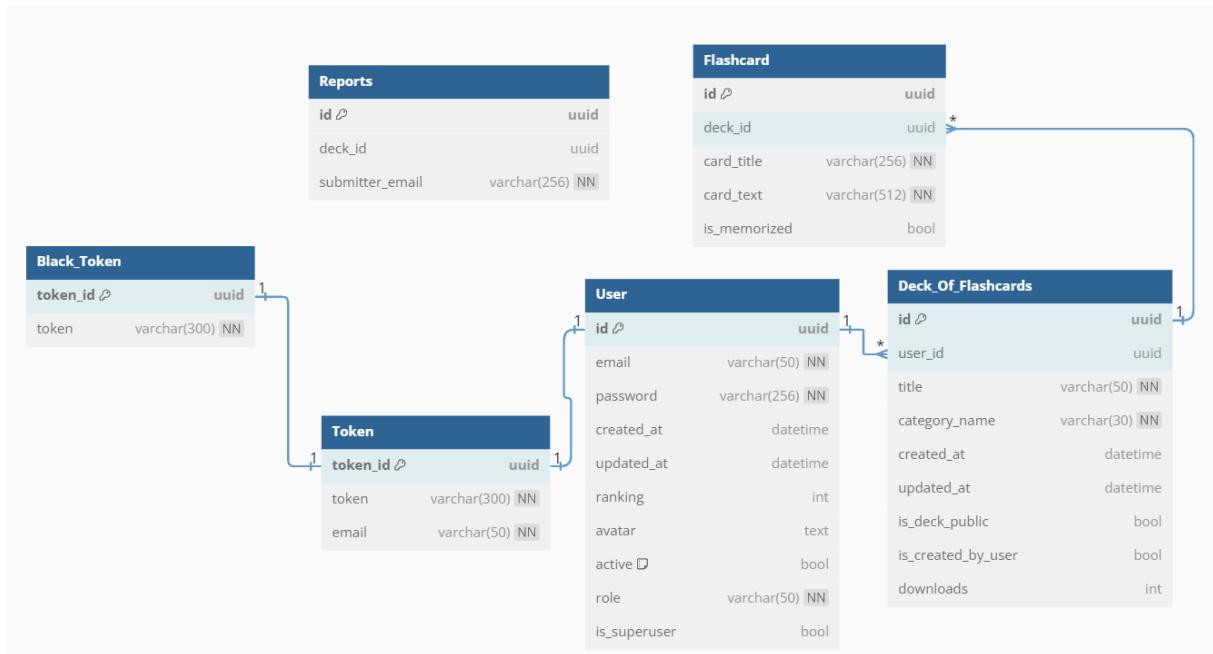
- Aktorzy:
 - Student / Uczeń - Docelowi użytkownicy systemu.
 - Użytkownik [zalogowany] - Ogólny użytkownik systemu, który jest zalogowany i ma dostęp do funkcji aplikacji.
 - Moderator - Użytkownik systemu z uprawnieniami do zarządzania użytkownikami oraz taliами fiszek.
 - ChatGPT - System odpowiedzialny za generowanie treści fiszek.
- Przypadki użycia:
 - Zarządzanie talią - Umożliwia użytkownikowi tworzenie, edytowanie oraz usuwanie talii fiszek.
 - Zarządzanie fiszkami - Podstawowy przypadek użycia, który obejmuje operacje na fiszkach w ramach talii. Użytkownik może dodawać, edytować oraz usuwać fiszki.
 - Generowanie treści fiszki - Funkcjonalność generowania treści fiszki na podstawie podanej definicji.
 - Aktywacja trybu nauki - Umożliwia użytkownikowi rozpoczęcie sesji nauki z fiszkami.
 - Przeglądanie rankingu - Użytkownik może przeglądać rankingi talii i użytkowników.
 - Pobieranie talii - Umożliwia użytkownikowi pobranie talii udostępnionej przez innego użytkownika.
 - Zarządzanie kontem - Użytkownik może zarządzać swoim kontem, zmieniać dane i ustawienia.
 - Aktualizacja danych - Umożliwia zmianę danych konta użytkownika.

- Zarządzanie użytkownikami - Dostępne dla moderatora, umożliwia przegląd i usuwanie kont użytkowników.
- Zarządzanie taliami - Dostępne dla moderatora, umożliwia usuwanie i przeglądanie talii użytkowników.
- Relacje między przypadkami użycia:
 - "extend" - Relacja rozszerzenia przypadku.
 - "CRUD" - Wskazuje na operacje tworzenia, odczytywania, aktualizowania oraz usuwania.



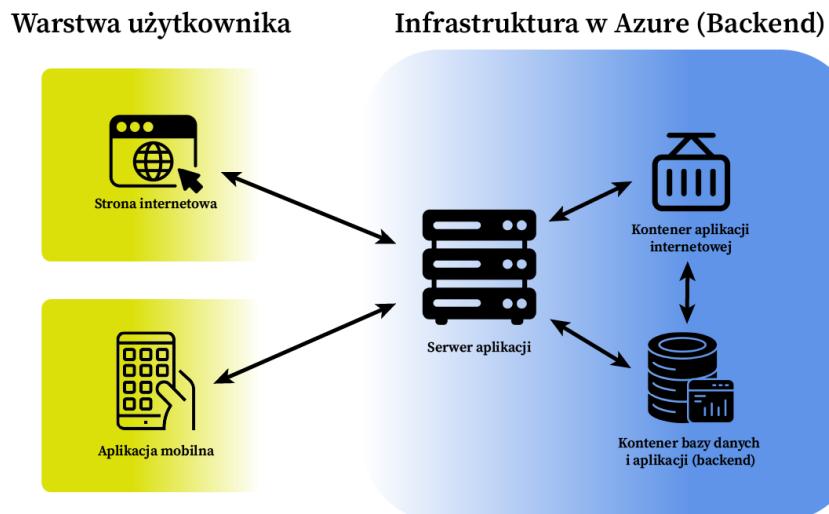
Rysunek 5.1: Diagram przypadków użycia.

Diagram ERD (Entity-Relationship Diagram) to graficzne przedstawienie struktury bazy danych używane w projektowaniu i modelowaniu baz danych.



Rysunek 5.2: Diagram ERD.

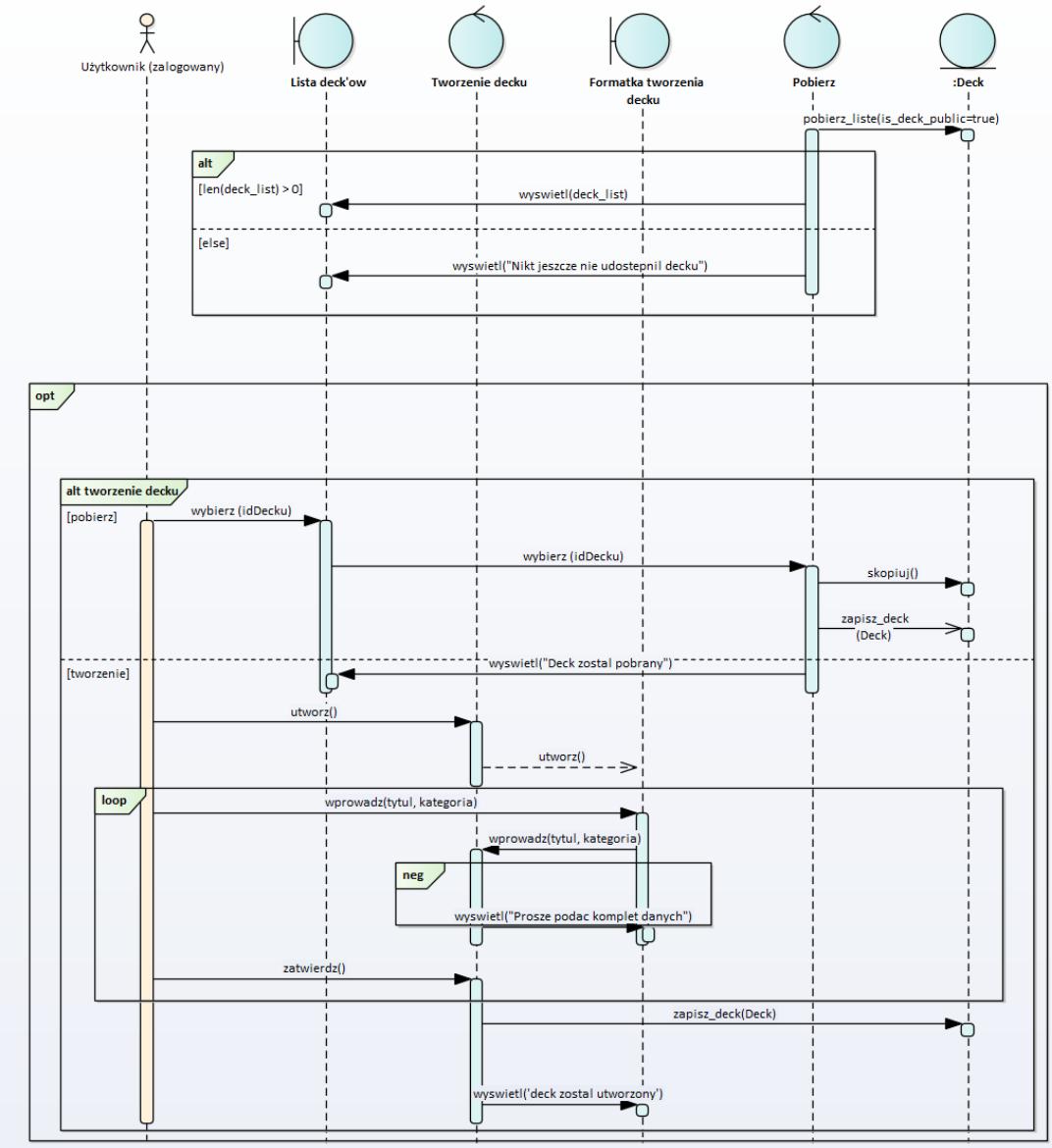
Diagram architektury stanowi reprezentację organizacji systemu informatycznego w oparciu o który został zrealizowany podjęty projekt.



Rysunek 5.3: Diagram architektury.

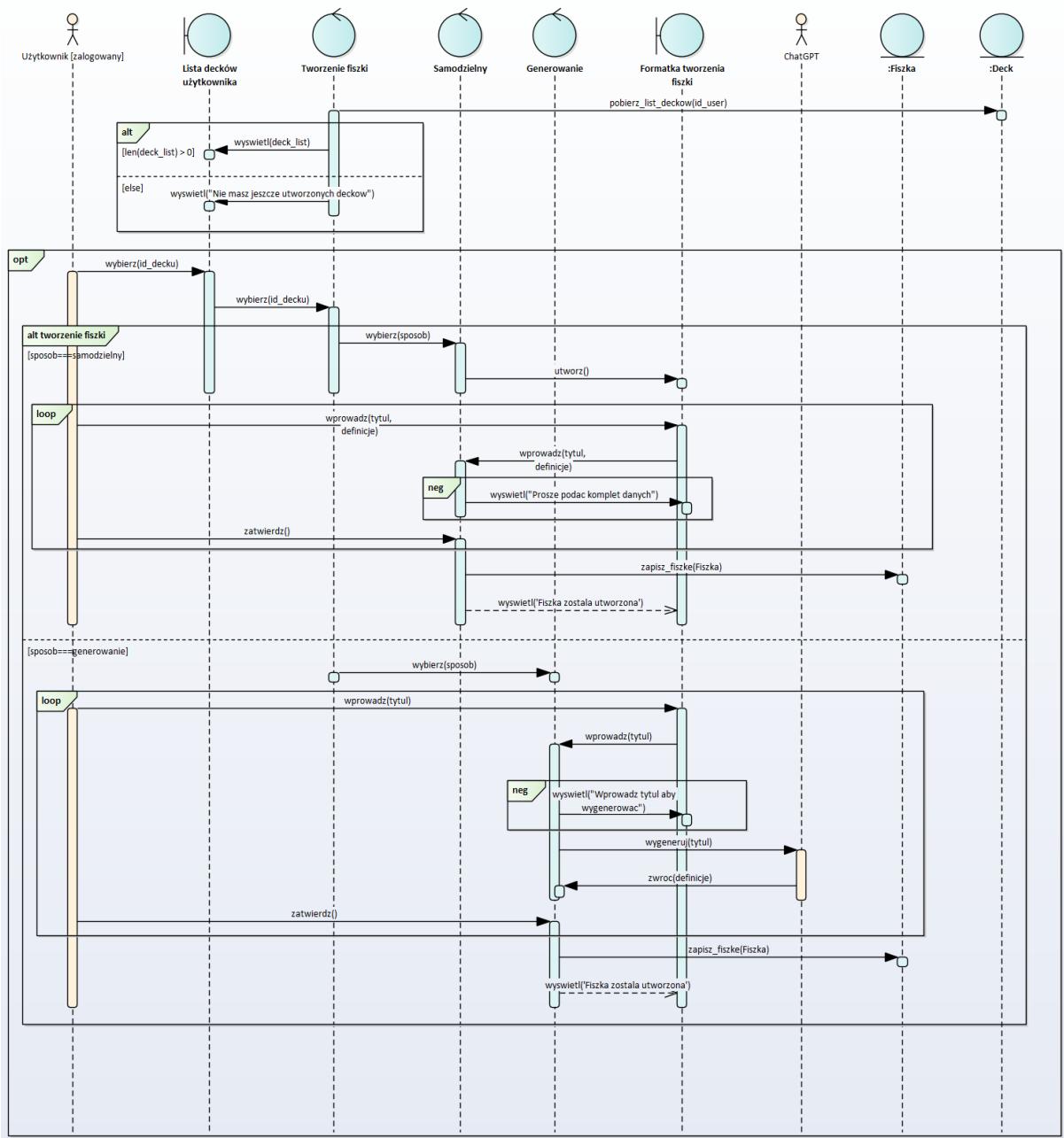
W diagramach sekwencji zostały przedstawione sposoby tworzenia deck'u oraz fiszk

Użytkownik ma możliwość stworzyć deck dla siebie na dwa sposoby. Pierwszym sposobem jest utworzenie go samodzielnie od podstaw, drugim jest pobranie talii udostępnionej przez innego użytkownika.



Rysunek 5.4: Diagram sekwencji przedstawiający tworzenie talii fiszek.

Tworzenie fiszek natomiast może przebiec poprzez zredagowanie definicji zagadnienia samodzielnie (wpisaniu lub dyktowaniu przez mikrofon) lub wygenerowaniu jej za pomocą ChatGPT.



Rysunek 5.5: Diagram sekwencji przedstawiający tworzenie fiszki.

5.2 Udziałowcy

KARTA UDZIAŁOWCA	
, Identyfikator:	UOB 01
Nazwa::	Zespół projektowy
Opis::	Zespół tworzący system
Typ udziałowca::	Ożywiony bezpośredni
Punkt widzenia::	Wytwarzanie aplikacji mobilnej, strony internetowej, zaplecza technicznego systemu; zarządzanie, utrzymanie i rozwój systemu
Ograniczenia::	Czynniki ludzkie organizacyjne, deadline i czas
Wymagania::	WF01, WF02, WF09

Tabela 5.1: Zespół projektowy jako udziałowiec.

Administrator - zarządza utrzymaniem systemu i jego środowiska technicznego, ma dostęp do kodu źródłowego oraz serwerów systemu, odpowiada za bazy danych oraz ciągłość funkcji.

KARTA UDZIAŁOWCA	
Identyfikator:	UOB 02
Nazwa::	Administrator systemu
Opis::	Administrator wytworzzonego systemu projektowego
Typ udziałowca::	Ożywiony bezpośredni
Punkt widzenia::	Nadzór i techniczne utrzymanie systemu, administracja środowiskiem systemowym
Ograniczenia::	Koszty utrzymaniowe - budżet chmury, parametry techniczne środowiska systemowego
Wymagania::	tu tylko symbole wymagań wyspecyfikowanych w rozdziale 3

Tabela 5.2: Administrator systemu jako udziałowiec.

Użytkownik systemu - osoba używająca systemu oraz wszystkich wytworzonych funkcjonalności użytkowych. Brak narzuconych ograniczeń ilościowych, prototyp projektu zakłada wstępную obsługę maksymalnie kilkuset użytkowników (200 - 300).

KARTA UDZIAŁOWCA	
Identyfikator:	UOB 03
Nazwa::	Użytkownik systemu
Opis::	Osoba korzystająca z systemu za pośrednictwem aplikacji mobilnej lub strony web
Typ udziałowca::	Ożywiony bezpośredni
Punkt widzenia::	Korzystanie z systemu: tworzenie talii fiszek, wykorzystywanie ich do nauki, udostępnianie talii innym
Ograniczenia::	Wytworzzone funkcjonalności
Wymagania::	WF03, WF04, WF05, WF06, WF07, WF08, WF10, WF11

Tabela 5.3: Użytkownik systemu jako udziałowiec.

KARTA UDZIAŁOWCA		
Identyfikator:	UOB 04	
Nazwa::	Dostawca technologii chmury	
Opis::	Firma świadcząca usługi chmurowe	
Typ udziałowca::	Ożywiony bezpośredni	
Punkt widzenia::	Dostarczanie infrastruktury i środowiska technicznego w którym działa system projektowy.	
Ograniczenia::	Budżet projektowy, parametry techniczne środowiska	
Wymagania::	tu tylko symbole wymagań wyspecyfikowanych w rozdziale 3	

Tabela 5.4: Dostawca technologii chmury jako udziałowiec.

5.3 Wymagania ogólne

5.3.1 Moduł autoryzacji

KARTA WYMAGANIA		
Identyfikator:	WO1	Priorytet: M – must
Nazwa:	Moduł autoryzacji	
Opis:	Rejestracja konta nowego użytkownika. Możliwość usunięcia konta użytkownika. Logowanie do systemu. Wylogowanie użytkownika z systemu. <u>Edycja danych użytkownika.</u>	
Udziałowiec:	Zespół projektowy (UOB 01) Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Logowanie do systemu (WF02) Wylogowanie z systemu (WF03) Edycja danych użytkownika (WF04) Usunięcie konta użytkownika (WF05) Weryfikacja konta użytkownika poprzez e-mail (WF14)	

Tabela 5.5: Przykładowe wymaganie ogólne lub dziedzinowe

5.3.2 Moduł zarządzania talią

KARTA WYMAGANIA		
Identyfikator:	WO2	Priorytet: M – must
Nazwa:	Moduł zarządzania talią	
Opis:	Tworzenie talii fiszek. Pobranie talii utworzonej przez innych użytkowników. Edycja utworzonych talii fiszek lub edycja talii pobranej przez innych użytkowników. Usunięcie talii fiszek.	
Udziałowiec:	Zespół projektowy (UOB 01) Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Tworzenie talii fiszek (WF06) Usunięcie talii fiszek (WF07) Edycja talii fiszek (WF08) Import talii innych użytkowników (WF11)	

Tabela 5.6: Wymaganie ogólne dla modułu zarządzania talią

5.3.3 Moduł uczenia

KARTA WYMAGANIA			
Identyfikator:	W03	Priorytet:	M – must
Nazwa:	Moduł uczenia		
Opis:	Tryb sterowania głosem pozwala na uruchomienie talii fiszek w specjalnym trybie, który pozwala na sterowanie talią fiszek przy wykorzystaniu komend głosowych. Zwykły tryb uczenia uruchamia talię w pełnym ekranie i pozwala na podzielenie talii na fiszki, które użytkownik zapamiętał i na te nie zapamiętane.		
Udziałowiec:	Zespół projektowy (UOB 01) Użytkownik systemu (UOB 03)		
Wymagania powiązane:	Tryb uczenia się z talii fiszek (WF09) Sterowanie talią przy użyciu mowy (WF10)		

Tabela 5.7: Wymaganie ogólne dla modułu uczenia

5.3.4 Moduł udogodnień

KARTA WYMAGANIA			
Identyfikator:	W04	Priorytet:	S – should)
Nazwa:	Moduł udogodnień		
Opis:	Tryb dark mode i light mode pozwala użytkownikowi na zmianę kolorystyki systemu w celu uniknięcia przemęczenia wzroku. Kalendarz oznacza dni, w których użytkownik korzysta z aplikacji, może to spowodować większą motywację u użytkownika aby regularnie korzystał z systemu.		
Udziałowiec:	Zespół projektowy (UOB 01) Użytkownik systemu (UOB 03)		
Wymagania powiązane:	Tryb dark mode i light mode (WF12) Kalendarz śledzący aktywność (WF13)		

Tabela 5.8: Wymaganie ogólne dla modułu udogodnień

5.3.5 Moduł wspomagania tworzenia talii

KARTA WYMAGANIA			
Identyfikator:	W05	Priorytet:	W – won’t
Nazwa:	Moduł wspomagania tworzenia talii		
Opis:	Analiza dokumentów poprzez wykorzystanie sztucznej inteligencji pozwala na wygenerowanie zestawu fiszek poprzez wgranie do systemu dokumentu w formacie csv lub pdf. Użytkownik może wygenerować treść fiszki, podając kilka słów kluczowych na podstawie których sztuczna inteligencja przeszukuje bazę definicji i na podstawie wyszukanych definicji generuje treść.		
Udziałowiec:	Użytkownik systemu (UOB 03)		
Wymagania powiązane:	Tworzenie talii fiszek przez analizę dokumentu (WF15) Generowanie treści fiszki na podstawie słów kluczowych (WF16)		

Tabela 5.9: Wymaganie ogólne dla modułu wspomagania tworzenia talii

5.4 Wymagania funkcjonalne

5.4.1 Rejestracja konta użytkownika

KARTA WYMAGANIA	
Identyfikator:	WF01 Priorytet: M – must
Nazwa:	Rejestracja nowego użytkownika w systemie
Opis:	Użytkownik musi utworzyć konto aby móc korzystać z aplikacji.
Kryteria akceptacji:	Nowy użytkownik dodany do systemu
Dane wejściowe:	nickname, e-mail, hasło
Warunki początkowe:	Użytkownik musi posiadać adres e-mail
Warunki końcowe:	Utworzenie konta użytkownika
Sytuacje wyjątkowe:	Brak łączności z bazą danych. Wprowadzenie dane przez użytkownika istnieją już w bazie danych.
Szczegóły implementacji:	Punkt 7.3.1
Udziałowiec:	Zespół projektowy (UOB 01) Użytkownik systemu (UOB 03)
Wymagania powiązane:	Logowanie do systemu (WF02) Wylogowanie z systemu (WF03) Edycja danych użytkownika (WF04) Usunięcie konta użytkownika (WF05) Weryfikacja konta użytkownika poprzez e-mail (WF14)

Tabela 5.10: Wymagania na interfejs z otoczeniem dla procesu rejestracji użytkownika

5.4.2 Logowanie do systemu

KARTA WYMAGANIA	
Identyfikator:	WF02 Priorytet: M – must
Nazwa:	Logowanie do systemu
Opis:	Użytkownik musi zalogować się do systemu, aby mieć dostęp do systemu.
Kryteria akceptacji:	Pomyślne zalogowanie do systemu
Dane wejściowe:	e-mail i hasło
Warunki początkowe:	Posiadane konto użytkownika, konto musi być aktywne.
Warunki końcowe:	Dostęp do systemu
Sytuacje wyjątkowe:	Brak łączności z bazą danych. Wprowadzenie niepoprawnych danych logowania, co uniemożliwia zalogowanie do systemu.
Szczegóły implementacji:	Punkt 7.3.2
Udziałowiec:	Zespół projektowy (UOB 01)
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Wylogowanie z systemu (WF03) Edycja danych użytkownika (WF04) Usunięcie konta użytkownika (WF05) Weryfikacja konta użytkownika poprzez e-mail (WF14)

Tabela 5.11: Wymagania na interfejs z otoczeniem dla procesu logowania do systemu

5.4.3 Wylogowanie z systemu

KARTA WYMAGANIA		
Identyfikator:	WF03	Priorytet: M – must
Nazwa:	Wylogowanie z systemu	
Opis:	Jako użytkownik chciałbym mieć możliwość wylogowania z systemu, aby inne osoby nie mogły korzystać z mojego konta.	
Kryteria akceptacji:	Wylogowanie użytkownika z systemu.	
Dane wejściowe:	Brak	
Warunki początkowe:	Użytkownik zalogowany do systemu	
Warunki końcowe:	Wylogowanie z systemu	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą.	
Szczegóły implementacji:	Punkt 7.3.3	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Logowanie do systemu (WF02) Edycja danych użytkownika (WF04) Usunięcie konta użytkownika (WF05) Weryfikacja konta użytkownika poprzez e-mail (WF14)	

Tabela 5.12: Wymagania na interfejs z otoczeniem dla procesu wylogowania z systemu

5.4.4 Edycja danych użytkownika

KARTA WYMAGANIA		
Identyfikator:	WF04	Priorytet: M – must
Nazwa:	Edycja danych użytkownika	
Opis:	Jako użytkownik muszę mieć możliwość zmiany hasła lub e-mail ponieważ mogę stracić dostęp do konta e-mail lub moje hasło z różnych powodów może stać się jawnie.	
Kryteria akceptacji:	Zmienione parametry logowania	
Dane wejściowe:	Użytkownik podaje nowy parametr wraz z hasłem, w celu edycji danych użytkownika	
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu.	
Warunki końcowe:	Zmienione parametry logowania	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą. Wprowadzenie niepoprawnych danych co uniemożliwia zmianę parametrów użytkownika.	
Szczegóły implementacji:	Punkt 7.3.4	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Logowanie do systemu (WF02) Wylogowanie z systemu (WF03) Usunięcie konta użytkownika (WF05) Weryfikacja konta użytkownika poprzez e-mail (WF14)	

Tabela 5.13: Wymagania na interfejs z otoczeniem dla procesu edycji danych użytkownika

5.4.5 Usunięcie konta użytkownika

KARTA WYMAGANIA			
Identyfikator:	WF05	Priorytet:	M – must
Nazwa:	Usunięcie konta użytkownika		
Opis:	Jako użytkownik chcę mieć możliwość usunięcia swojego konta, gdy przestanę korzystać z aplikacji.		
Kryteria akceptacji:	Usunięte konto użytkownika		
Dane wejściowe:	e-mail i hasło		
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu.		
Warunki końcowe:	Konto użytkownika usunięte z systemu		
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą. Wprowadzenie niepoprawnych danych, co uniemożliwia usunięcie konta użytkownika.		
Szczegóły implementacji:	Punkt 7.3.5		
Udziałowiec:	Użytkownik systemu (UOB 03)		
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Logowanie do systemu (WF02) Wylogowanie z systemu (WF03) Edycja danych użytkownika (WF04) Weryfikacja konta użytkownika poprzez e-mail (WF14)		

Tabela 5.14: Wymagania na interfejs z otoczeniem dla procesu usunięcia konta użytkownika

5.4.6 Tworzenie talii fiszek

KARTA WYMAGANIA			
Identyfikator:	WF06	Priorytet:	M – must
Nazwa:	Tworzenie talii fiszek		
Opis:	Jako użytkownik muszę mieć możliwość utworzenia własnej talii fiszek, w celu uczenia się zagadnień, które mają dla mnie znaczenie.		
Kryteria akceptacji:	Utworzona talia fiszek		
Dane wejściowe:	tytuł talii fiszek, kategoria talii, tytuł fiszki, treść fiszki		
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu.		
Warunki końcowe:	Utworzona nowa talia fiszek.		
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą. Puste pole z nazwą talii uniemożliwia tworzenie talii. Pusty tytuł fiszki lub pusta treść uniemożliwia utworzenie fiszki.		
Szczegóły implementacji:	Punkt 7.3.6		
Udziałowiec:	Użytkownik systemu (UOB 03)		
Wymagania powiązane:	Usunięcie talii fiszek (WF07) Edycja talii fiszek (WF08) Import talii innych użytkowników (WF11)		

Tabela 5.15: Wymagania na interfejs z otoczeniem dla procesu tworzenia talii fiszek

5.4.7 Usunięcie talii fiszek

KARTA WYMAGANIA		
Identyfikator:	WF07	Priorytet: M – must
Nazwa:	Usunięcie talii fiszek	
Opis:	Jako użytkownik chcę mieć możliwość usunięcia talii fiszek, z których nie będę już korzystał.	
Kryteria akceptacji:	Usunięta talia fiszek	
Dane wejściowe:	Brak	
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu. Utworzona talia fiszek, którą można usunąć.	
Warunki końcowe:	Talia fiszek zostaje usunięta.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą.	
Szczegóły implementacji:	Punkt 7.3.7	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Tworzenie talii fiszek (WF06) Edycja talii fiszek (WF08) Import talii innych użytkowników (WF11)	

Tabela 5.16: Wymagania na interfejs z otoczeniem dla procesu usunięcia talii fiszek

5.4.8 Edycja talii fiszek

KARTA WYMAGANIA		
Identyfikator:	WF08	Priorytet: M – must
Nazwa:	Edycja talii fiszek	
Opis:	Jako użytkownik chcę mieć możliwość edycji talii fiszek, aby zaktualizować informacje dotyczące talii.	
Kryteria akceptacji:	Zaktualizowana zawartość talii	
Dane wejściowe:	Nowy tytuł talii, kategoria lub zmieniona treść fiszki.	
Warunki początkowe:	Talia fiszek, która będzie edytowana.	
Warunki końcowe:	Talia fiszek zostaje zaktualizowana o nowe informacje.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą. Puste pole dotyczące tytułu talii uniemożliwia zapisanie zmian.	
Szczegóły implementacji:	Punkt 7.3.8	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Tworzenie talii fiszek (WF06) Usunięcie talii fiszek (WF07) Import talii innych użytkowników (WF11)	

Tabela 5.17: Wymagania na interfejs z otoczeniem dla procesu edycji talii fiszek

5.4.9 Tryb uczenia się z talii fiszek

KARTA WYMAGANIA			
Identyfikator:	WF09	Priorytet:	M – must
Nazwa:	Tryb uczenia się z talii fiszek		
Opis:	System musi posiadać tryb uczenia się, który pozwoli użytkownikowi na naukę i zapamiętywanie zagadnień znajdujących się w talii fiszek. W trakcie nauki użytkownik dzieli talię na zestaw z pojęciami które już opanował i na te, których jeszcze nie pamięta.		
Kryteria akceptacji:	Talia fiszek zostaje podzielona na dwa zbiory, fiszki zapamiętane i nie zapamiętane.		
Dane wejściowe:	Brak		
Warunki początkowe:	Utworzona talia fiszek lub pobrała talia fiszek, która zostanie wykorzystana do nauki.		
Warunki końcowe:	Talia fiszek zostaje podzielona na dwa zbiory, fiszki zapamiętane i nie zapamiętane.		
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą.		
Szczegóły implementacji:	Punkt 7.3.9		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:	Sterowanie talią przy użyciu mowy (WF10)		

Tabela 5.18: Wymagania na interfejs z otoczeniem dla trybu uczenia się z talii fiszek

5.4.10 Sterowanie talią przy użyciu mowy

KARTA WYMAGANIA			
Identyfikator:	WF10	Priorytet:	M – must
Nazwa:	Sterowanie talią przy użyciu mowy		
Opis:	Jako użytkownik chcę mieć możliwość uczenia się w warunkach, w których odczytywanie treści fiszki jest utrudnione, na przykład w trakcie prowadzenia samochodu. Sterowanie talią z przy użyciu mowy i odczytanie zawartości fiszki przez sztuczną inteligencję pozwala na naukę podczas prowadzenia pojazdu.		
Kryteria akceptacji:	Użytkownik steruje talią fiszek przy użyciu komend głosowych		
Dane wejściowe:	Brak		
Warunki początkowe:	Utworzona talia fiszek		
Warunki końcowe:	Talia fiszek pozostaje niezmieniona		
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych. Hasłas, który aplikacja może przechwytywać, przez co komendy głosowe mogą działać niepoprawnie. Fiszki utworzone w innym języku niż angielski co może spowodować trudności w odczytaniu ich zawartości przez syntezator mowy.		
Szczegóły implementacji:	Punkt 7.3.10		
Udziałowiec:	Użytkownik systemu (UOB 03)		
Wymagania powiązane:	Tryb uczenia się z talii fiszek (WF09)		

Tabela 5.19: Wymagania na interfejs z otoczeniem dla sterowania talią przy użyciu mowy

5.4.11 Import talii innych użytkowników

KARTA WYMAGANIA		
Identyfikator:	WF11	Priorytet: M – must
Nazwa:	Import talii innych użytkowników	
Opis:	Jako użytkownik chcę mieć możliwość pobrania talii od innych użytkowników, aby mieć łatwy dostęp do treści i zagadnień, które mnie interesują.	
Kryteria akceptacji:	Talia dodana do zakładki talii pobranych od innych użytkowników.	
Dane wejściowe:	Brak	
Warunki początkowe:	Talia którą użytkownik może pobrać.	
Warunki końcowe:	Talia dodana do zakładki talii pobranych od innych użytkowników.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych.	
Szczegóły implementacji:	Punkt 7.3.11	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Tworzenie talii fiszek (WF06) Usunięcie talii fiszek (WF07) Edycja talii fiszek (WF08)	

Tabela 5.20: Wymagania na interfejs z otoczeniem dla procesu importowania talii fiszek od innych użytkowników

5.4.12 Tryb dark mode i light mode

KARTA WYMAGANIA		
Identyfikator:	WF12	Priorytet: S – should
Nazwa:	Tryb dark mode i light mode	
Opis:	Jako użytkownik chciałbym mieć możliwość zmiany kontrolowanego jasności systemu aby mój wzrok się nie przemęczał.	
Kryteria akceptacji:	Zmiana koloru interfejsu aplikacji.	
Dane wejściowe:	Brak	
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu.	
Warunki końcowe:	Zmiana koloru interfejsu aplikacji.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych.	
Szczegóły implementacji:	Punkt 7.3.12	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Kalendarz śledzący aktywność (WF13)	

Tabela 5.21: Wymagania na interfejs z otoczeniem dla trybu dark mode i light mode

5.4.13 Kalendarz śledzący aktywność

KARTA WYMAGANIA		
Identyfikator:	WF13	Priorytet: S – should
Nazwa:	Kalendarz śledzący aktywność	
Opis:	Kalendarz odznaczający dni, w których użytkownik korzystał z aplikacji, mogłyby zwiększyć motywację użytkownika do regularnego korzystania z aplikacji.	
Kryteria akceptacji:	Kalendarz oznacza dzień w momencie uruchomienia przez użytkownika aplikacji.	
Dane wejściowe:	Brak	
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu.	
Warunki końcowe:	Dzień oznaczony w kalendarzu.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych.	
Szczegóły implementacji:	Brak	
Udziałowiec:	Zespół projektowy (UOB 01)	
Wymagania powiązane:	Tryb dark mode i light mode (WF12)	

Tabela 5.22: Wymagania na interfejs z otoczeniem dla kalendarza śledzącego aktywność użytkownika

5.4.14 Weryfikacja konta użytkownika poprzez e-mail

KARTA WYMAGANIA		
Identyfikator:	WF14	Priorytet: C – could
Nazwa:	Weryfikacja konta użytkownika poprzez e-mail	
Opis:	Użytkownik po pomyślnej rejestracji, otrzyma na e-mail wiadomość z linkiem przekierowującym na stronę webową, gdzie nastąpi wysłanie żądania o aktywację konta użytkownika.	
Kryteria akceptacji:	Wiadomość dostarczona na wskazany e-mail, poprawnie działająca aktywacja konta, poprawne przekierowanie.	
Dane wejściowe:	Token autoryzujący	
Warunki początkowe:	Użytkownik musi posiadać istniejący adres e-mail, na który przyjdzie wiadomość z linkiem aktywującym konto.	
Warunki końcowe:	Użytkownik aktywuje konto.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych. Użytkownik nie ma dostępu do podanego przez siebie adresu e-mail lub podany adres jest błędny. Wiadomość nie dotarła do użytkownika.	
Szczegóły implementacji:	Brak	
Udziałowiec:	Zespół projektowy (UOB 01)	
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Logowanie do systemu (WF02)	

Tabela 5.23: Wymagania na interfejs z otoczeniem dla weryfikacji konta użytkownika poprzez e-mail

5.4.15 Tworzenie talii fiszek poprzez zeskanowanie dokumentu

KARTA WYMAGANIA		
Identyfikator:	WF15	Priorytet: W – won't
Nazwa:	Tworzenie talii fiszek poprzez zeskanowanie dokumentu	
Opis:	Jako użytkownik chciałbym mieć możliwość szybkiego utworzenia zestawu talii fiszek poprzez wgranie gotowego dokumentu w formacie csv lub pdf z którego została utworzona talia fiszek na podstawie zagadnień znajdujących się w dokumencie.	
Kryteria akceptacji:	Talia fiszek utworzona po analizie dokumentu.	
Dane wejściowe:	Dokument do analizy w formacie csv lub pdf.	
Warunki początkowe:	Posiadane konto użytkownika, użytkownik zalogowany do systemu, dokument do analizy.	
Warunki końcowe:	Utworzona talia fiszek.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych. Dokument nie zdany do odczytu.	
Szczegóły implementacji:	Brak	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Generowanie treści fiszki na podstawie słów kluczowych (WF16)	

Tabela 5.24: Wymagania na interfejs z otoczeniem dla tworzenia talii fiszek poprzez zeskanowanie dokumentu

5.4.16 Generowanie treści fiszki na podstawie słów kluczowych

KARTA WYMAGANIA		
Identyfikator:	WF16	Priorytet: M – must
Nazwa:	Generowanie treści fiszki na podstawie słów kluczowych	
Opis:	Jako użytkownik chcę, aby system potrafił generować definicje zagadnień, które wpisałem na pierwszą stronę karty, co ułatwi tworzenie talii.	
Kryteria akceptacji:	Zawartość fiszki wygenerowana przy pomocy zewnętrznego API na podstawie podanej treści na stronie polu dla przedniej strony fiszki.	
Dane wejściowe:	Brak	
Warunki początkowe:	Utworzone konto użytkownika.	
Warunki końcowe:	Treść fiszki wygenerowana na podstawie zawartości wpisanej w pole przedniej strony karty.	
Sytuacje wyjątkowe:	Awaria API lub połączenia z API. Podanie przez użytkownika zdania nie mającego sensu.	
Szczegóły implementacji:	Punkt 7.3.13	
Udziałowiec:	Użytkownik systemu (UOB 03)	
Wymagania powiązane:	Tworzenie talii fiszek przez analizę dokumentu (WF15)	

Tabela 5.25: Wymagania na interfejs z otoczeniem dla generowania treści fiszki na podstawie słów kluczowych

5.4.17 Resetowanie hasła na konta

KARTA WYMAGANIA		
Identyfikator:	WF017	Priorytet: S – should
Nazwa:	Resetowanie hasła na konta	
Opis:	Użytkownik po podaniu adresu mailowego połączonego z kontem, otrzyma na e-mail wiadomość z linkiem przekierowującym na stronę webową, gdzie będzie mógł podać nowe hasło.	
Kryteria akceptacji:	Pomyślna zmiana hasła.	
Dane wejściowe:	Token autoryzujący, nowe hasło	
Warunki początkowe:	Użytkownik musi posiadać istniejący adres e-mail, na który przyjdzie wiadomość z linkiem przekierowującym na widok strony internetowej.	
Warunki końcowe:	Użytkownik pomyślnie zmienia hasło.	
Sytuacje wyjątkowe:	Awaria bazy danych lub brak połączenia z bazą danych. Użytkownik nie ma dostępu do podanego przez siebie adresu e-mail lub podany adres jest błędny. Wiadomość nie dotarła do użytkownika.	
Szczegóły implementacji:	Brak	
Udziałowiec:	Zespół projektowy (UOB 01)	
Wymagania powiązane:	Logowanie do systemu (WF02)	

Tabela 5.26: Wymagania na interfejs z otoczeniem dla resetowania hasła

5.5 Interfejs z otoczeniem

KARTA WYMAGANIA		
Identyfikator:	I01	Priorytet: S – should
Nazwa:	ChatGPT 3.5	
Opis:	Aplikacja zostaje połączona z API ChatGPT, aby użytkownik miał możliwość wygenerowania treści fiszki na podstawie podanego słowa.	
Kryteria akceptacji:	Prawidłowo generuje definicje dla wysyłanych słów kluczowych.	
Dane wejściowe:	Słowo podane przez użytkownika.	
Warunki początkowe:	Aplikacja połączona z API ChatGPT.	
Warunki końcowe:	ChatGPT zwraca definicję podanego słowa.	
Sytuacje wyjątkowe:	Brak połączenia z ChatGPT.	
Szczegóły implementacji:	Punkt 7.3.13	
Udziałowiec:	Zespół projektowy - UOB 01	
Wymagania powiązane:	Generowanie treści fiszki na podstawie słów kluczowych (WF16)	

Tabela 5.27: Wymagania na interfejs z otoczeniem dla integracji z ChatGPT 3.5

5.6 Wymagania pozafunkcjonalne

5.6.1 Instrukcja korzystania z trybu sterowania głosem

KARTA WYMAGANIA		
Identyfikator:	WN01	Priorytet: M – must
Nazwa:	Instrukcja korzystania z trybu sterowania głosem	
Opis:	Instrukcja zawiera komendy, wraz z opisem, których użytkownik korzysta w momencie uruchomienia trybu sterowania głosem.	
Kryteria akceptacji:	Nowy użytkownik po zapoznaniu się z instrukcją, jest w stanie korzystać z trybu sterowania głosem.	
Udziałowiec:	Zespół projektowy (UOB 01)	
Wymagania powiązane:	Sterowanie talią przy użyciu mowy (WF10)	

Tabela 5.28: Wymagania na interfejs z otoczeniem dla instrukcji korzystania z trybu sterowania głosem

5.6.2 Limit błędów dotyczących logowania

KARTA WYMAGANIA		
Identyfikator:	WN02	Priorytet: S – should
Nazwa:	Limit błędów dotyczących logowania	
Opis:	Po wpisaniu 3 razy niepoprawnych danych logowania w aplikacji pojawi się okienko informujące o blokadzie aplikacji na 1 minutę, okienko zawiera przycisk zmiany hasła. Przycisk przekierowuje do podstrony zawierającej pole do podania e-mail na które przyjdzie wiadomość dotycząca zmiany hasła.	
Kryteria akceptacji:	Użytkownik otrzymuje wiadomość e-mail, która przekierowuje go do formularza zmiany hasła.	
Udziałowiec:	Zespół projektowy (UOB 01)	
Wymagania powiązane:	Resetowanie hasła (WF017)	

Tabela 5.29: Wymagania na interfejs z otoczeniem dla limitu błędów logowania

5.6.3 Dostępność systemu

KARTA WYMAGANIA		
Identyfikator:	WN03	Priorytet: M – must
Nazwa:	Dostępność systemu	
Opis:	System powinien być dostępny 7 dni w tygodniu, 24 godziny na dobę.	
Kryteria akceptacji:	System działa przez 7 dni bez żadnych awarii. Co godzinę sprawdzane jest połączenie z systemem.	
Udziałowiec:	Zespół projektowy (UOB 01)	
Wymagania powiązane:		

Tabela 5.30: Wymagania na interfejs z otoczeniem dla dostępności systemu

5.6.4 Responsywność

KARTA WYMAGANIA			
Identyfikator:	WN04	Priorytet:	M – must
Nazwa:	Responsywność		
Opis:	System musi być responsywnym, dostosowując się do wielkości okienka lub urządzenia.		
Kryteria akceptacji:	System w pełni dostosuje się do dostępnej wielkości okienka.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.31: Wymagania na interfejs z otoczeniem dla responsywności systemu

5.6.5 Kompatybilność

KARTA WYMAGANIA			
Identyfikator:	WN05	Priorytet:	M – must
Nazwa:	Kompatybilność		
Opis:	System musi być kompatybilny z różnymi przeglądarkami internetowymi.		
Kryteria akceptacji:	System w pełni działa i ukazuje się w wybranej przeglądarce.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.32: Wymagania na interfejs z otoczeniem dla kompatybilności z przeglądarkami internetowymi

5.6.6 Hashowanie haseł

KARTA WYMAGANIA			
Identyfikator:	WN06	Priorytet:	M – must
Nazwa:	Hashowanie haseł		
Opis:	Hasła użytkowników hashowane przy użyciu algorytmu sha256.		
Kryteria akceptacji:	Hasło w bazie danych zostanie zaszyfrowane.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:	Rejestracja konta użytkownika (WF01) Logowanie do systemu (WF02)		

Tabela 5.33: Wymagania na interfejs z otoczeniem dla hashowania haseł

5.7 Wymagania na środowisko docelowe

5.7.1 Przeglądarka

KARTA WYMAGANIA			
Identyfikator:	ŚD01	Priorytet:	M – must
Nazwa:	Przeglądarka		
Opis:	Chrome wersja 110.0.0.0, Firefox wersja 120.0, Microsoft Edge 115.0.0.0		
Kryteria akceptacji:	Strona internetowa uruchamia się.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.34: Wymagania na interfejs z otoczeniem dla przeglądarek internetowych

5.7.2 System Android 10 i iOS 16

KARTA WYMAGANIA			
Identyfikator:	ŚD02	Priorytet:	M – must
Nazwa:	System Android 10 i iOS 16		
Opis:	Aplikacja działa na urządzeniach mobilnych z systemem Android w wersji 10 i wyższych, w przypadku iOS w wersji 16 i wyższych.		
Kryteria akceptacji:	Aplikacja uruchamia się na urządzeniach Android w wersji 10 i wyższych, w przypadku iOS w wersji 16 i wyższych.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.35: Wymagania na interfejs z otoczeniem dla systemów mobilnych

5.7.3 Kontenery dockerowe

KARTA WYMAGANIA			
Identyfikator:	ŚD03	Priorytet:	M – must
Nazwa:	Kontenery dockerowe		
Opis:	Dwa kontenery: jeden zawierający backend wraz z bazą danych, drugi zawierający aplikację webową.		
Kryteria akceptacji:	Łączność pomiędzy kontenerami.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.36: Wymagania na interfejs z otoczeniem dla kontenerów dockerowych

5.7.4 Baza danych

KARTA WYMAGANIA			
Identyfikator:	ŚD04	Priorytet:	M – must
Nazwa:	Baza danych		
Opis:	Mariadb - relacyjna baza danych w wersji 11.0. Będzie przechowywana w kontenerze razem z backend'em.		
Kryteria akceptacji:	Prawidłowo tworzące się obiekty modeli, stabilna łączność z backendem.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.37: Wymagania na interfejs z otoczeniem dla bazy danych

5.7.5 Python

KARTA WYMAGANIA			
Identyfikator:	ŚD05	Priorytet:	M – must
Nazwa:	Python		
Opis:	Python w wersji 3.10 będzie odpowiedzialny za poprawne działanie backendu.		
Kryteria akceptacji:	Backend reaguje na otrzymywane requesty.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.38: Wymagania na interfejs z otoczeniem dla Pythona

5.7.6 Node.js

KARTA WYMAGANIA			
Identyfikator:	ŚD06	Priorytet:	M – must
Nazwa:	Node.js		
Opis:	Minimalna wersja Node.js dla aplikacji webowej oraz mobilnej to wersja 16. Będzie on odpowiedzialny za działania aplikacji webowej, jak i mobilnej.		
Kryteria akceptacji:	Poprawne uruchomienie aplikacji webowej i mobilnej.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:			

Tabela 5.39: Wymagania na interfejs z otoczeniem dla Node.js

5.7.7 System operacyjny

KARTA WYMAGANIA			
Identyfikator:	ŚD07	Priorytet:	M – must
Nazwa:	System operacyjny		
Opis:	System operacyjny Ubuntu w wersji 20, w którym zainicjowane będą kontenery dockerowe oraz środowisko z zapleczem technicznym projektu.		
Kryteria akceptacji:	System stabilnie utrzymuje połączenie między użytkownikami a aplikacją, obsługuje środowisko techniczne projektu z zoptymalizowanym zużyciem zasobów oraz spełnia podstawowe wymagania zabezpieczeń bezpieczeństwa sieciowego.		
Udziałowiec:	Zespół projektowy (UOB 01)		
Wymagania powiązane:	ŚD03 ŚD04 ŚD05 ŚD06		

Tabela 5.40: Wymagania na interfejs z otoczeniem dla systemu operacyjnego

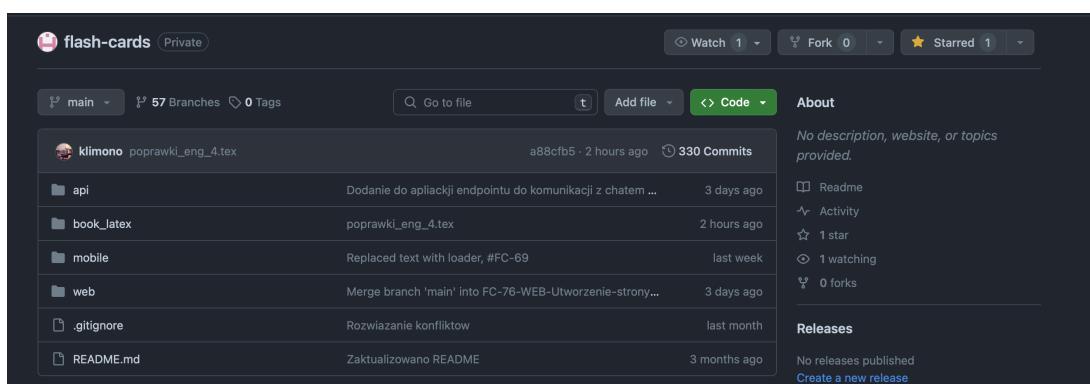
Rozdział 6

Architektura projektu i użyte technologie

Podjęty projekt cechuje się wysokim stopniem złożoności oraz potrzebą ciągłej komunikacji i współpracy w zespole. Efektywne zarządzanie infrastrukturą kodu, dokumentacji oraz organizacją prac nakłada obowiązek korzystania z rozwiązań, które umożliwiają wspólne planowanie, monitorowanie oraz realizację postępów. W tym rozdziale zostaną omówione kluczowe narzędzia użyte w projekcie w ujęciu zarówno narzędzi organizacji pracy - jak i wykorzystanych technologii oraz narzędzi programistycznych.

6.1 Narzędzia organizacji pracy

Github Platforma dostarczająca usługi zdalnego przechowywania oraz kontroli wersji projektów informatycznych. Na potrzeby projektu GitHub został wykorzystany w celu utworzenia repozytorium przechowującego kod źródłowy: aplikacji webowej, aplikacji mobilnej, backendu aplikacji oraz niniejszej pracy wraz ze wszystkimi niezbędnymi assetami i dokumentacją. Repozytorium odgrywa kluczową rolę stanowiąc ważne narzędzie przy organizacji projektu o zadanej złożoności nad którym cały zespół pracuje jednocześnie.



Rysunek 6.1: Repozytorium projektu umieszone na GitHub.

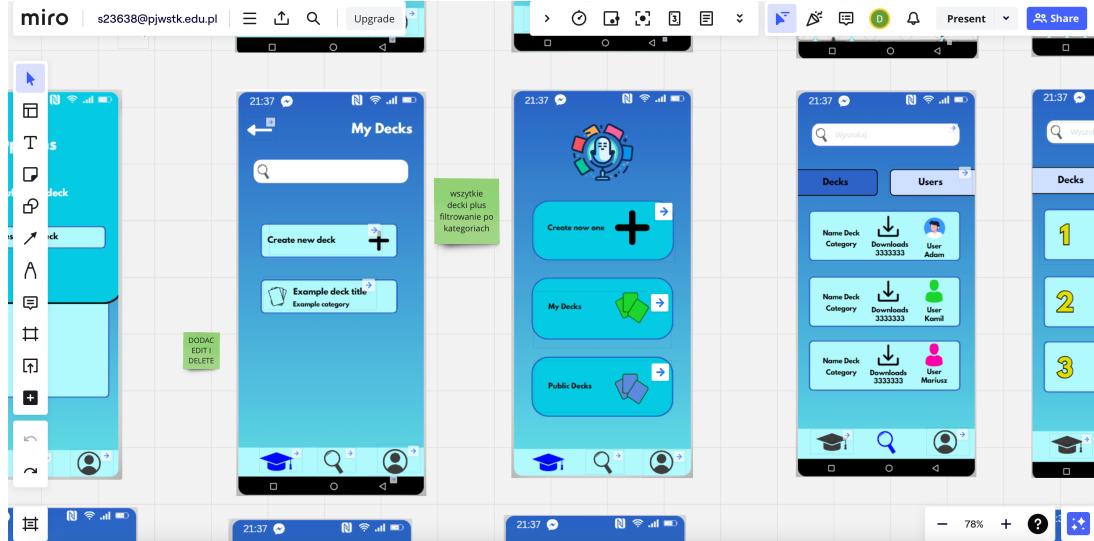
Jira Platforma służąca głównie do zarządzania projektami oraz do śledzenia powiązanych z nimi błędów. Jira została wykorzystana w celu dokumentacji, organizacji oraz planowania rozwoju systemu Fishki. Najważniejszą użytką w projekcie funkcją jest możliwość zarządzania sprintami i taskami, tj. zadaniami, bez których realizacja pracy w wybranym modelu byłaby niemożliwa.

The screenshot shows the Jira Backlog interface for a project named 'flash-cards'. At the top, there are filters for 'Wersja' (Version), 'Epic', and 'Etykieta' (Label). Below the header, a section titled 'ukonczenie aplikacji' (Application completion) is shown. A summary bar indicates 0 tasks in progress ('W TOKU') and 0 tasks completed ('GOTOWE'). The main list contains ten tasks, each with a checkmark, a title, and a status indicator:

- FC-35 [BACKEND] wstępne testy integracyjne (Status: DO ZROBienia)
- FC-50 [Web] Profil użytkownika (Status: W TOKU)
- FC-61 [WEB] Usunięcie konta użytkownika (Status: DO ZROBienia)
- FC-63 [WEB] Zmiana email użytkownika (Status: DO ZROBienia)
- FC-66 [WEB] Utworzenie strony public decks (Status: DO ZROBienia)
- FC-68 [WEB] Utworzenie widoku dla statystyk użytkownika (Status: DO ZROBienia)
- FC-64 [WEB] Zmiana nicku użytkownika (Status: W TOKU)
- FC-62 [WEB] Zmiana hasła użytkownika (Status: DO ZROBienia)
- FC-70 [WEB] Utworzenie strony do sterowania głosem talia fiszek (Status: GOTOWE)

Rysunek 6.2: Sprint rozpisany przy pomocy Jira.

Miro Wirtualna tablica umożliwiająca współpracę zespołową w czasie rzeczywistym. Miro zostało użyte do połączenia ze sobą mockupów aplikacji mobilnej utworzonych w canvie.

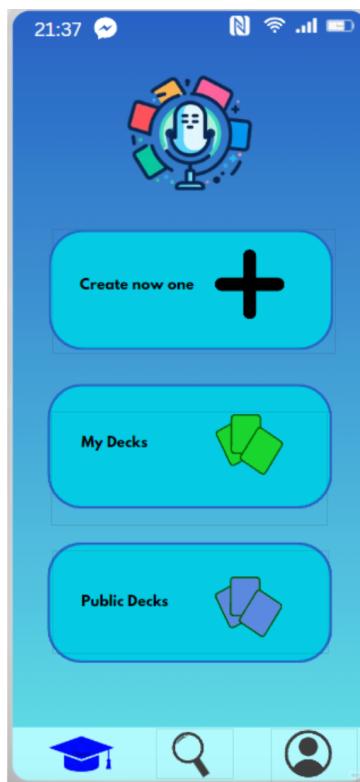


Rysunek 6.3: Tablica w Miro z interaktywnymi mockupami projektu.

Google Drive Usługa pozwalająca na przechowywanie plików wirtualnie w chmurze. Wykorzystana w projekcie do zarządzania i przechowywania niektórych dokumentów związanych z projektem.

Rysunek 6.4: Dokumenty przechowywane na Google Drive.

Canva Bezpłatne narzędzie służące do tworzenia grafik i stron internetowych. Program został użyty w celu utworzenia mockapów aplikacji moblinej oraz diagramu architektury.



Rysunek 6.5: Grafika mockupu wykonana w Canva.

Discord Bezpłatny program służący do komunikacji głosowej i tekstowej. Wykorzystywany do spotkań organizacyjnych i komunikacji członków zespołu o problemach związanych z projektem.

Enterprise Architect Program służący do modelowania diagramów. Został wykorzystany do przygotowania diagramu przypadków użycia.

dbdiagram.io Narzędzie online wykorzystywane przy projektowaniu diagramów.. W projekcie zostało użyte do stworzenia diagramu ERD.

LaTeX System składu tekstu. Wykorzystany w projekcie do organizacji oraz tworzenia książki projektu.

Git Rozproszony system kontroli wersji, wykorzystany w projekcie do zarządzania kodem.

6.2 Aplikacja webowa

React Biblioteka JavaScript wykorzystywana do tworzenia interfejsów użytkownika, pozwala ona na szybkie i łatwe kreowanie uniwersalnych komponentów, umożliwia to wykorzystanie jednego komponentu w wielu miejscach w kodzie. Biblioteka została wykorzystana do tworzenia strony internetowej.

TypeScript Rozszerza JavaScript o możliwość typowania zmiennych. TypeScript został wykorzystany w celu szybszego wykrywania błędów w kodzie i łatwiejszym zarządzaniu go.

Material UI Popularna biblioteka UI, zawiera zestaw komponentów do tworzenia interfejsów internetowych. Używana w połączeniu z biblioteką React.js.

Node JS Środowisko, które pozwala na uruchomienie javascriptu na serwerze jako samodzielnej aplikacji.

6.3 Aplikacja mobilna

React Native Framework pozwalający na tworzenie natywnych aplikacji mobilnych dla iOS i Android przy użyciu JavaScript i Reacta. Umożliwia programistom wykorzystanie jednego kodu źródłowego do budowy aplikacji na obie wymienione platformy.

Expo Framework i platforma służąca do tworzenia aplikacji na iOS i Android za pomocą React Native, która ułatwia rozwój aplikacji mobilnych

TypeScript Jw. w sekcji “Aplikacja webowa”.

6.4 Backend aplikacji

Python Wysokopoziomowy język programowania. Wykorzystany w projekcie ze względu na swoją prostotę, wszechstronność i wieloplatformowość.

FastAPI Framework do tworzenia interfejsów API w języku Python. Wykorzystany w projekcie ze względu na szybkość co pozwala na szybką komunikację z bazą danych. Framework wykorzystuje asynchroniczną obsługę żądań co pozwala aplikacji osiągnąć wysoką wydajność.

6.5 Baza danych

MariaDB Relacyjna baza danych, wykorzystana w projekcie ze względu na wysoką wydajność, a także otwartoźródłowy charakter systemu.

6.6 Infrastruktura serwerowa

Microsoft Azure Platforma chmurowa pozwalająca wdrożyć wirtualną infrastrukturę IT. Wykorzystana na rzecz utrzymania środowiska projektowego tj. kontenera z API, backend aplikacji webowej i bazę danych w oparciu o wirtualną maszynę z systemem operacyjnym Ubuntu server.

Serwer Nginx Zaawansowany serwer WWW i serwer PROXY. Wykorzystany w celu pełnej konfiguracji aplikacji webowej, HTTPS i przekierowania ruchu ze strony do API.

6.7 Narzędzia programistyczne

Jetbrains PyCharm Środowisko programistyczne zaprojektowane dla języka Python. Wykorzystane w projekcie podczas projektowania i tworzenia backendu aplikacji oraz do komplikacji książki w LaTeX.

Jetbrains WebStorm Środowisko programistyczne zaprojektowane głównie dla języka JavaScript oraz technologii wykorzystywanych w budowie aplikacji webowej/mobilnej. Użyte w projektowaniu oraz tworzeniu frontendu.

Docker Narzędzie programistyczne, które umożliwia tworzenie, wdrażanie i uruchamianie aplikacji w kontenerach. Docker wykorzystany został do umieszczenia aplikacji webowej, API i bazy danych w kontenerach, aby ułatwić łatwe uruchamiania aplikacji.

Visual Studio Code Środowisko programistyczne wszechstronnego przeznaczenia. Wykorzystane na etapie tworzenia aplikacji ze względu na dodatek Thunder Client, który zastąpił popularnego Postmana, do testowania API.

Android Studio Oficjalne zintegrowane środowisko programistyczne dla systemu operacyjnego Google Android. Umożliwia ściąganie różnych wersji emulatorów, ściągania wymaganych paczek do uruchomienia danego emulatora, a także do celów testowania wytwarzanej aplikacji mobilnej.

Xcode Oficjalne środowisko programistyczne Apple udostępniające narzędzia do tworzenia aplikacji oraz oprogramowania dla systemów z rodziny iOS i macOS. Wykorzystane w projekcie w celu testowania aplikacji mobilnej w symulatorach urządzeń zainstalowanym systemem iOS.

Adobe Illustrator Rozbudowany program graficzny przeznaczony do tworzenia i edycji grafiki wektorowej. Utworzone zostały za jego pomocą ikony oraz tło do naszych aplikacji.

Rozdział 7

Implementacja

Tworzenie projektu odbyło się metodą przyrostowo-ewolucyjną. Metoda ta pozwala na szybką adaptację i elastyczność, która jest kluczowa w przypadku projektów, które mogą ulegać zmianom w czasie realizacji. Tworzenie systemu odbyło się w dwóch etapach, fazie planowania i fazie implementacji. Faza planowania dotyczyła określenia celu projektu, funkcjonalności systemu i wypełnienia dokumentacji. Faza implementacji odbywała się w sprintach, które zazwyczaj trwały od 2 do 3 tygodni.

7.1 Faza planowania

Na samym początku przeprowadzona została burza mózgów w celu wymyślenia tematu naszego projektu [6]. Po analizie potencjalnych tematów zespół zdecydował się na aplikację do uczenia się z wykorzystaniem metody fiszek. Następnym krokiem było wypełnienie karty projektu, zespół musiał określić cele projektu, miary sukcesu i główne funkcjonalności. Po określeniu ogólnych założeń dotyczących projektu, zespół zajął się wypełnieniem dokumentu założeń wstępnych, który dotyczył opisu naszego problemu, analizy konkurencji i ogólnej wizji konstrukcyjnej. Kolejnym krokiem w fazie planowania było wypełnienie specyfikacji wymagań systemowych, która dotyczyła szczegółowego opisu wymagań dotyczących naszego projektu. Ostatnim podjętym działaniem było utworzenie mockupów aplikacji mobilnej i kilku mockupów aplikacji webowej, co pozwoliło na konsolidację informacji i na wytworzenie wspólnej wizji projektu. Po ukończeniu planowania zespół był gotowy do przejścia w fazę implementacji.

7.2 Faza implementacji

Implementacja projektu odbyła się w sprintach, które trwały od 2 do 4 tygodni. Członkowie zespołu w ramach każdego sprintu mieli do wykonania zadania, które były przypisane do nich w narzędziu do zarządzania projektem jira. Po implementacji nowych funkcjonalności lub komponentów odbywało się testowanie, aplikacja była uruchamiana w celu sprawdzenia czy zaimplementowane rozwiązania, działały i były zgodne z oczekiwaniami.

7.2.1 Sprint 1

Pierwszy przyrost dotyczył podstawowej konfiguracji projektu z wykorzystaniem framework'a FastAPI. Na początku został utworzony projekt wraz z podstawową strukturą katalogów, następnie powstał plik readme, który zawierał instrukcję uruchomienia projektu. Kolejnym wykonanym krokiem było utworzenie modeli karty fiszek i talii. Ostatnim zadaniem wykonanym w przyroście było opakowanie projektu w kontener dockerowy.

7.2.1.1 Wykonane zadania:

Zadanie	Wykonawca
Utworzenie kontenera backendu	Jakub
Aktualizacja Readme	Oliwier
[BACKEND] Dodanie 'connector' dla łączności z bazą danych	Jakub
[BACKEND] Utworzenie modelu autoryzacji i użytkownika	Jakub
[BACKEND] Utworzenie cli i funkcji do tworzenia modeli w bazie	Jakub
[BACKEND] Dodanie app_middleware'y	Jakub
[BACKEND] Utworzenie modelu tali oraz karty	Oliwier
[BACKEND] Naprawienie ścieżki importów projektu	Oliwier

Tabela 7.1: Zadania wykonane w sprintie 1.

7.2.2 Sprint 2

Drugi przyrost był skupiony na utworzeniu widoku logowania i rejestracji dla aplikacji mobilnej, a także na implementacji tokenu autoryzacji i przypisaniu ról użytkownikom systemu.

7.2.2.1 Wykonane zadania:

Zadanie	Wykonawca
[BACKEND] Poprawienie middleware dla jwt	Oliwier
[MOBILE] Utworzyć style scss i zimportować	Daniel
[MOBILE] Wstępny widok logowania i rejestracji bez funkcjonalności (mobilka)	Daniel
[MOBILE] Dodać serwis logowania i rejestracji	Jakub
[BACKEND] Utworzyć wstępne fixture	Jakub
[BACKEND] Utworzyć endpoint do resetowania hasła	Jakub
[MOBILE] Utworzyć wstępne pliki i konfiguracje dla aplikacji mobilnej	Jakub
[BACKEND] Utworzenie loggera	Jakub
[BACKEND] Poprawienie ścieżki dla api	Jakub
[BACKEND] Dodanie nowej dependencji dla roli	Jakub
[BACKEND] Dodanie tokenu na czarną listę	Jakub

Tabela 7.2: Zadania wykonane w sprincie 2.

7.2.3 Sprint 3

Trzeci przyrost był skupiony na zadaniach związanych z uporządkowaniem kodu aplikacji mobilnej. Ważnym krokiem dotyczącym strony webowej było utworzenie strony domowej. Backend został rozbudowywany o nowe endpointy związane z talią, został utworzony role checker do sprawdzania roli użytkownika aplikacji.

7.2.3.1 Wykonane zadania:

Zadanie	Wykonawca
[MOBILE] Dodać możliwość rejestracji	Jakub
[MOBILE] Dodać panel użytkownika	Jakub
[MOBILE] Zrobić porządek w kodzie	Jakub
[MOBILE] Dodać walidacje hasła	Jakub
[MOBILE] Zaktualizować serwisy z nową metodą request	Jakub
[WEB] Okno logowania	Wiktor
[BACKEND] Poprawić endpoint decs	Oliwier
[WEB] Okno rejestracji	Wiktor
[MOBILE] Dodać interface dla zwracanych danych w metodzie request	Jakub
[BACKEND] napisanie endpointów dla fiszek	Oliwier
[WEB] Utworzenie kontenera docker dla NodeJS	Oliwier
[BACKEND] Poprawić dependencje dla RoleCheckera	Jakub
[MOBILE] Utworzenie metody request	Jakub
[MOBILE] Utworzyc strone domowa po zalogowaniu	Daniel
[WEB] Utworzyc strone domowa po zalogowaniu	Oliwier
[MOBILE] Przerzucić regexy z do katalogu validator	Daniel
[MOBILE] Poprawić widok logowania i rejestracji	Daniel
[WEB] Utworzenie strony do tworzenia decku	Oliwier

Tabela 7.3: Zadania wykonane w sprintie 3.

7.2.4 Sprint 4

W czwartym przyroście aplikacja webowa została w dużym stopniu rozbudowana o nowe widoki, a także została połączona z warstwą backend, umożliwiło to komunikację strony internetowej z bazą danych w celu pobierania i tworzenia danych potrzebnych do rejestracji i logowania użytkownika. Rozbudowa aplikacji mobilnej była skupiona na profilu użytkownika, zostały dodane funkcjonalności związane ze zmianą i aktualizacją danych.

7.2.4.1 Wykonane zadania:

Zadanie	Wykonawca
[MOBILE] Utworzyć loader	Jakub
[MOBILE] Dodać modal aby potwierdzić hasłem	Jakub
[BACKEND] Poprawa modeli talii i fiszki	Oliwier
[MOBILE] Dodać walidacje hasła	Jakub
[MOBILE] Widok dla zmiany e-mail	Jakub
[MOBILE] Widok dla zmiany hasła	Jakub
[MOBILE] Widok dla zmiany nazwy użytkownika	Jakub
[BACKEND] Dodać endpointy na aktualizowanie danych użytkownika	Jakub
[MOBILE] Widok tworzenia decku	Daniel
[MOBILE] Bottom tab navigator	Daniel
[MOBILE] Widok my decks	Daniel
[WEB] Połączenie strony tworzenia fiszek z backendem	Oliwier
[WEB] Połączenie strony domowej z backendem	Oliwier
[WEB] Utworzenie strony my decks	Oliwier
[WEB] Wylogowanie użytkownika	Oliwier
[WEB] Utworzenie customowego okna dla alertów	Oliwier
[WEB] Utworzenie widoku strony do nauki z talii fiszek	Oliwier

Tabela 7.4: Zadania wykonane w sprintie 4.

7.2.5 Sprint 5

Do aplikacji został dodany endpoint, umożliwiający komunikację z czatem GPT. Pozwoliło to na dodanie do strony webowej funkcjonalności związanej z generowaniem treści. Zostały także zaimplementowany tryb uczenia się, który pozwala na dzielenie fiszek na zapamiętane i nie zapamiętane. Do aplikacji mobilnej zostało dodane usuwanie konta użytkownika oraz poprawiona została struktura kodu.

7.2.5.1 Wykonane zadania:

Zadanie	Wykonawca
[MOBILE] Dodać usuwanie konta	Jakub
[BACKEND] Dodał podstawowe fixture dla decków	Jakub
[WEB] Zmiana hasła użytkownika	Wiktor
[WEB] Utworzenie strony do sterowania głosem talia fiszek	Oliwier
[BACKEND] Dodanie endpointu do obsługi ChatGPTF	Oliwier
[WEB] Dodać generowanie treści przy użyciu ChatGPT	Oliwier
[BACKEND] Dodanie do usera kolumny dla avatara poprawa dodanie w flashcard kolumny is memorized	Oliwier
[WEB] Dodanie widoku dla not memorized flashcards	Oliwier
[BACKEND] Dodanie endpointów do flashcards które filują zapamiętane i nie zapamiętane karty	Oliwier
[WEB] Dodanie trybu uczenia	Oliwier
[WEB] Dodać opcje edycji fiszki i możliwość udostępnienia decku	Oliwier
[MOBILE] Poprawienie nazewnictwa w kodzie nawigacji	Daniel
[MOBILE] Naprawa struktury ekranów	Daniel

Tabela 7.5: Zadania wykonane w sprincie 5.

7.2.6 Sprint 6

Do aplikacji mobilnej został dodany ekran tworzenia talii fiszek, następne dodane widoki były związane z trybem uczenia się. Aplikacja webowa została rozbudowana o ranking talii i użytkowników. Na stronie webowej zostały dodane funkcjonalności związane z aktualizacją danych użytkownika. Utworzony został model nlp do rozumienia semantyki słów wykorzystany do sterowania talią fiszek przy użyciu komend głosowych. Dla backendu zostały napisany testy integracyjne uruchamiany przy użyciu biblioteki "pytest".

7.2.6.1 Wykonane zadania:

Zadanie	Wykonawca
[MOBILE] Dodać możliwość update'u avatara	Jakub
[MOBILE] Naprawić błąd z query w widoku decklist	Jakub
[WEB] Profil użytkownika	Wiktor
[BACKEND] Dodać celery do aplikacji	Jakub
[WEB] Usunięcie konta użytkownika	Wiktor
[WEB] Zmiana e-mail użytkownika	Wiktor
[WEB] Zmiana nicku użytkownika	Wiktor
[BACKEND] Utworzyć metodę do wysyłania e-maila	Jakub
[BACKEND] Utworzyć templatki dla maila	Jakub
[MOBILE] Detale usera z rankingu	Jakub
[MOBILE] Spiąć "My Decks" z API	Daniel
[MOBILE] Spiąć "Create Decks" z API	Daniel
[MOBILE] Dodać ekran podglądu fiszek	Daniel
[MOBILE] Dodać ekran podglądu decku	Daniel
[WEB] Ranking użytkowników	Oliwier
[MOBILE] Dodać listę udostępnionych talii	Jakub
[MOBILE] Spiąć podgląd decku z API	Daniel
[MOBILE] Dodać ekran tworzenia fiszki	Daniel
[MOBILE] Spiąć ekran tworzenia fiszki z API	Daniel
[MOBILE] Dodać service flashcards	Daniel
[MOBILE] Utworzyć component pobierania danych z API	Daniel
[MOBILE] Dodać edycję i usunięcie fiszki	Daniel
[MOBILE] Dodać ekran settings dla decku	Daniel
[WEB] Poprawa rankingów	Oliwier
[MOBILE] Dodać i spiąć memorized flashcards	Daniel
[BACKEND] Testy integracyjne	Oliwier
[WEB] Utworzenie strony public decks	Oliwier
[WEB] Dodanie obsługi złożonych komend głosowych	Oliwier
[BACKEND] Utworzenie modelu nlp do rozumienia semantyki słów	Oliwier

Tabela 7.6: Zadania wykonane w sprintie 6.

7.2.7 Sprint 7

Sprint 7 był ostatnim przyrostem w naszym projekcie. Zespół był skupiony na dopracowaniu ostatecznej wersji aplikacji i poprawie występujących błędów. Został także, uruchomiony serwer na azurze na, którym jest hostowana strona internetowa.

7.2.7.1 Wykonane zadania:

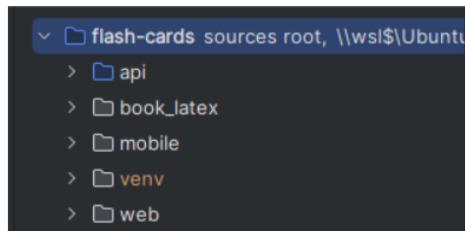
Zadanie	Wykonawca
[MOBILE] Dodać możliwość update'u avatara	Jakub
[MOBILE] Naprawić błąd z query w widoku decklist	Jakub
[WEB] Profil użytkownika	Wiktor
[BACKEND] Dodać celery do aplikacji	Jakub
[WEB] Usunięcie konta użytkownika	Wiktor
[WEB] Zmiana e-mail użytkownika	Wiktor
[WEB] Zmiana nicku użytkownika	Wiktor
[BACKEND] Utworzyć metodę do wysyłania e-maila	Jakub
[BACKEND] Utworzyć templatki dla maila	Jakub
[MOBILE] Detale usera z rankingu	Jakub
[MOBILE] Spiąć "My Decks" z API	Daniel
[MOBILE] Spiąć "Create Decks" z API	Daniel
[MOBILE] Dodać ekran podglądu fiszek	Daniel
[MOBILE] Dodać ekran podglądu decku	Daniel
[WEB] Ranking użytkowników	Oliwier
[MOBILE] Dodać listę udostępnionych talii	Jakub
[MOBILE] Spiąć podgląd decku z API	Daniel
[MOBILE] Dodać ekran tworzenia fiszki	Daniel
[MOBILE] Spiąć ekran tworzenia fiszki z API	Daniel
[MOBILE] Dodać service flashcards	Daniel
[MOBILE] Utworzyć component pobierania danych z API	Daniel
[MOBILE] Dodać edycję i usunięcie fiszki	Daniel
[MOBILE] Dodać ekran settings dla decku	Daniel
[WEB] Poprawa rankingów	Oliwier
[MOBILE] Dodać i spiąć memorized flashcards	Daniel
[BACKEND] Testy integracyjne	Oliwier
[WEB] Utworzenie strony public decks	Oliwier
[WEB] Dodanie obsługi złożonych komend głosowych	Oliwier
[BACKEND] Utworzenie modelu nlp do rozumienia semantyki słów	Oliwier
[MOBILE] Podczas wyszukiwania decku input traci focus	Jakub
[MOBILE] Użyć use refa w search public decks, users i reports	Jakub
[MOBILE] Dodać modal z instrukcją sterowania głosu	Jakub
[MOBILE] Dodać alerty po responsie	Jakub
[MOBILE] Poprawić animacje	Jakub
[MOBILE] Naprawić błąd nawigacji dla statystyk	Jakub

[MOBILE] Poprawić responsywność widoku public decks dla tabletu	Jakub
[MOBILE] Dodać panel dla moderatora	Jakub
[MOBILE] Dodać ikonę moderatora do paska nawigacji	Jakub
[MOBILE] Naprawić query search w deck list i user stats	Jakub
[MOBILE] Spiąć "Create Decks" z API	Jakub
[MOBILE] Dodać ekran podglądu fiszek	Jakub
[MOBILE] Dodać ekran podglądu decku	Jakub
[WEB] Utworzenie strony public decks	Oliwier
[WEB] Obsługa błędów (rejestracja/logowanie)	Wiktor
[MOBILE] Spiąć generowanie treści fiszki z AI	Daniel
[MOBILE] Utworzyć i spiąć tryb nauki	Daniel
[MOBILE] Spiąć ekran tworzenia fiszki z API	Daniel
[MOBILE] Dodać odsłuchanie fiszek w memorized i unmemorized flashcards	Daniel
[BACKEND] Dodanie tabeli i endpointów dla zgłoszonych talii	Oliwier
[WEB] Utworzenie panelu moderatora	Oliwier
[BACKEND] Usunięcie usera usuwa wszystkie powiązane rekordy, dodanie endpointu do usuwania innych userów	Jakub
[WEB] Poprawa okienka do edycji nazwy decku	Oliwier
[WEB] Poprawić responsywność strony	Oliwier
[SERWER] Zainicjować maszynę wirtualną w Azure	Daniel
[SERWER] Skonfigurować użytkowników	Daniel
[SERWER] Skonfigurować sieć, środowisko API, bazę danych i aplikację web	Daniel
[MOBILE] Dodać sterowanie głosem	Jakub
[WEB] Dodać usuwanie użytkownika do panelu moderatora	Oliwier
[BACKEND] Poprawa modelu do sterowania głosem	Oliwier
[MOBILE] Dodać podgląd i pobranie decku publicznego	Daniel
[WEB] Dodanie instrukcji sterowania głosem	Oliwier
[MOBILE] Dodać wyszukiwanie swoich decków	Daniel
[BACKEND] Dodać biblioteki do requirements	Oliwier
[MOBILE] Poprawić udostępnianie decku	Daniel
[SERWER] Skonfigurować połączenie HTTPS i domenę	Daniel
[MOBILE] Dodać potwierdzenie czy użytkownik akceptuje treść wygenerowaną przez ChatGPT	Daniel
[MOBILE] Spiąć z API zgłaszanie decku	Daniel

Tabela 7.7: Zadania wykonane w sprincie 7.

7.3 Szczegóły implementacji

Wykorzystanym podejściem prowadzenia pracy było monorepo. Cały kod projektu trzymany był w jednym repozytorium, takie podejście wymagało przemyślenia struktury katalogów i odpowiedniej konwencji nazewnictwa [monorepo]. Repozytorium zostało podzielone na 5 katalogów. W katalogu api zawarty jest kod projektu odpowiedzialny za warstwę backendu i bazy danych. Book₁atex zawiązało rozdziały prac dyplomowych pisanej przy uż



Rysunek 7.1: Zdjęcie przedstawia strukturę katalogów w projekcie.

7.3.1 Rejestracja konta użytkownika

Funkcjonalność umożliwiająca rejestrację w aplikacji.

7.3.1.1 Backend

Api przetwarza dane rejestracyjne po odebraniu zapytania z aplikacji webowej lub mobilnej. Do weryfikacji należy:

- Czy użytkownik z wprowadzonym e-mail istnieje już w bazie;
- Czy użytkownik z wprowadzoną nazwą użytkownika istnieje już w bazie;
- Czy hasła są równe (poprzez porównanie sumy hashu).

W przypadku błędnej weryfikacji zwraca odpowiednią wiadomość z błędem dla użytkownika np. "Wprowadzony e-mail jest już zajęty". Natomiast jeśli weryfikacja przebiegnie pomyślnie, użytkownik zostanie dodany do bazy, a system wyśle mu wiadomość e-mail z prośbą o aktywację konta.

```
  ↳ Jusiek +2
@router.post( path: "/register/", status_code=201)
async def register(
    payload: RegisterPayloadScheme,
    db: Session = Depends(get_db)
):
    payload = payload.dict()
    if get_user(payload['email'], db):
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Email already taken"
        )

    if get_user_by_username(payload['username'], db):
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Username already taken"
        )

    if payload['password'] != payload['re_password']:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Passwords do not match"
        )

    db.add(
        User(
            email=payload['email'],
            username=payload['username'],
            password=get_password_hash(payload['password']),
            role=UserRoles.get_default_roles()
        )
    )
    db.commit()

    token = create_access_token(
        data={ "sub": payload["email"] },
        expires_delta=timedelta(minutes=RESET_ACCESS_TOKEN_EXPIRE_MINUTES),
    )
    db.add(
        Token(access_token=token, user_email=payload["email"])
    )
    db.commit()

    send_active_account_email(payload["email"], token, db)
    return {'detail': 'User added successfully'}, 201
```

Rysunek 7.2: Implementacja kodu rejestracji użytkownika w backendzie.

7.3.1.2 Aplikacja moblina i aplikacja webowa

Aby założyć konto, użytkownik musi wypełnić formularz rejestracyjny wymagający podania nazwy użytkownika, adresu e-mail oraz hasła wpisanego jednakowego do dwóch pól w celach jego walidacji. Aplikacja pozwoli na utworzenie konta jedynie jeżeli:

- Pseudonim zawiera od 3 do 20 znaków i składa się wyłącznie z liter, cyfr, lub podkreśników;
- Adres e-mail jest dostarczony w prawidłowym dla niego formacie;
- Hasło posiada co najmniej 8 znaków;
- Hasło wpisane oba pola walidacyjne jest w nich takie samo;

- Adres e-mail i nazwa użytkownika są unikalne (czyli w bazie danych nie są przypisane do żadnego innego konta).

Po zatwierdzeniu prawidłowych danych aplikacja informuje komunikatem o udanym utworzeniu konta. Użytkownik zostaje przeniesiony do strony logowania. W przypadku podania nieprawidłowych danych, użytkownik nie zostanie zarejestrowany i zostanie powiadomiony komunikatem w aplikacji o błędzie.

```
public async register(body: object, navigation: NavigationProp<any>) : Promise<{res: Response, data: ...}> {
  return await request( {url, query, headers, method, body, formData, skipRedirect, navigation}: {
    url: AUTH_ENDPOINTS.register,
    method: "POST",
    body,
    navigation,
  });
}
```

Rysunek 7.3: Metoda „register” w serwisie Auth zastosowana w aplikacji mobilnej.

```
31  ↗ public async register(body: object) {
32    // @ts-ignore
33    ↗ return await request( {url, query, headers, method, body, formData, skipRedirect}: {
34      url: AUTH_ENDPOINTS.register,
35      method: 'POST',
36      body
37      // @ts-ignore
38    }).then(response => response.data)
39    ↗ .catch(error => {
40      throw error;
41    });
42 }
```

Rysunek 7.4: Metoda „register” w serwisie AuthService zastosowana w aplikacji webowej.

Funkcje z rysunków 7.3 i 7.2 wysyłają asynchroniczne żądanie POST do serwera w celu rejestracji nowego użytkownika. Metoda przekazuje dane rejestracyjne w treści żądania, a następnie zwraca czy żądanie zostało obsłużone.

7.3.2 Logowanie do systemu

Funkcjonalność umożliwiająca logowanie do aplikacji

7.3.2.1 Backend

Po wysłaniu requesta na przez aplikację webową lub mobilną api wyciąga przesłane dane, weryfikuje je. Weryfikacja odbywa się za pomocą funkcji `authenticate_user` zimportowaną z `utils`. Ta najpierw próbuje wyciągnąć z użytkownika z bazy, który posiada wysłany e-mail. Potem, jeżeli użytkownik istnieje, sprawdzamy hasło metodą klasową `verify_password`. Jeśli wszystko przebiegło pomyślnie, użytkownikowi zostanie zwrócony token autoryzacji i dane użytkownika. W przeciwnym razie zwrócona wiadomość i błędnie wprowadzonych danych.

```
Jusiek +1
@router.post( path: "/login/", response_model=LoginResponse)
async def login(
    payload: LoginPayloadScheme,
    db: Session = Depends(get_db)
):
    payload = payload.dict()
    user = authenticate_user(payload['email'], payload['password'], db)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Could not validate credentials"
        )

    token = create_access_token(
        data: {"sub": user.email},
        expires_delta=timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES),
    )

    db.add(
        Token(access_token=token, user_email=user.email)
    )
    db.commit()

    res_token = db.query(Token).filter(Token.access_token == token).first()

    return {
        "user_data": user,
        "token_data": res_token
    }
```

Rysunek 7.5: Implementacja kodu logowania użytkownika w backendzie.

7.3.2.2 Aplikacja mobilna i aplikacja webowa

Na stronie logowania, użytkownik musi wypełnić formularz wymagający uzupełnienia pól adres e-mail oraz hasło. Po zatwierdzeniu danych i ich weryfikacji pojawia się komunikat o zalogowaniu użytkownika, a następnie użytkownik zostanie przeniesiony na stronę główną. W przypadku podania nieprawidłowych danych, użytkownik nie zostanie zalogowany i zostanie poinformowany komunikatem o błędzie.

```
public async login(body: object, navigation: NavigationProp<any>): Promise<...> {
    return await request({url, query, headers, method, body, formData, skipRedirect, navigation}: {
        url: AUTH_ENDPOINTS.login,
        method: "POST",
        body,
        navigation,
    });
}
```

Rysunek 7.6: Metoda "login" zastosowana w aplikacji mobilnej.

```
public async login(body: object) {
  // @ts-ignore
  const {data} = await request({url, query, headers, method, body, formData, skipRedirect}: {
    url: AUTH_ENDPOINTS.login,
    method: 'POST',
    body
  });
  ActiveUser.set(data);
}
```

Rysunek 7.7: Metoda "login" zastosowana w aplikacji webowej.

Metoda `ActiveUser.set(data)` użyta w funkcjach z rysunków 7.5 i 7.6 zapisuje dane zalogowanego użytkownika w lokalnym stanie aplikacji. Dzięki temu, informacje o użytkowniku, takie jak token autoryzacyjny, mogą być łatwo dostępne w całej aplikacji. Pozwala to na:

- Personalizację interfejsu użytkownika (np. wyświetlanie nazwy użytkownika);
- Umożliwienie dostępu do zasobów, które wymagają uwierzytelnienia;
- Przechowywanie sesji użytkownika, aby mógł pozostać zalogowany pomiędzy różnymi wizytami na stronie.

7.3.3 Wylogowanie z systemu

Funkcjonalność pozwalająca na wylogowanie użytkownika.

7.3.3.1 Backend

Po kliknięciu wyloguj w aplikacji mobilnej czy w webowej, zostaje wysłany request na api, który dodaje aktualnie używany token do listy nieaktualnych tokenów.

```
  Jusiiek
@router.post("/logout/")
async def logout(
    token: Annotated[str, Depends(oauth2_scheme)],
    db: Session = Depends(get_db)
):
    db.add(
        Blacklist_Tokens(
            token=token
        )
    )
    db.commit()
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Token is no longer valid",
        headers={"WWW-Authenticate": "Bearer"}
)
```

Rysunek 7.8: Implementacja kodu wylogowania użytkownika w backendzie.

7.3.3.2 Aplikacja mobilna i aplikacja webowa

Aby wylogować się z aplikacji, użytkownik musi kliknąć w nawigacji "Logout". Po kliknięciu przycisku, dane użytkownika zostaną usunięte z pamięci lokalnej. Token autoryzacyjny użytkownika utworzony przy logowaniu zostanie dodany w bazie danych do tabeli zawierającej nieaktywne tokeny.

```
public async logout(navigation: NavigationProp<any>) : Promise<{res: Response, data: ...}> {
    return await request({
        url: AUTH_ENDPOINTS.logout,
        method: "POST",
        navigation,
    });
}
```

Rysunek 7.9: Metoda "logout" zastosowana w aplikacji mobilnej.

```
public async logout(token: string) : Promise<void> {
  try {
    await request( {url, query, headers, method, body, formData, skipRedirect}: {
      url: AUTH_ENDPOINTS.logout,
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`
      }
    });
    ActiveUser.clean();
  } catch (error) {
    console.error('Error during log out :', error);
  }
}
```

Rysunek 7.10: Metoda ”logout” zastosowana w aplikacji webowej.

Powyższe fragmenty serwisu odpowiadają za usunięcie danych użytkownika z pamięci lokalnej wykorzystujący endpoint dezaktywujący token autoryzacji.

7.3.4 Edycja danych użytkownika

Funkcjonalność umożliwiająca edycję danych użytkownika.

7.3.4.1 Backend

Aplikacje frontendowe wysyłają dane do aktualizacji np. e-mail. Jedynym obowiązkowym atrybutem przy aktualizacji jest `current_password`. Po zweryfikowaniu zgodności hasła, system przystępuje do zmiany danych wprowadzonych przez użytkownika. Na koniec zostają zwrócone użytkownika z poprawionym polami.

```
2 usages  ✨ Jakub Zurawski
class UserUpdateModel(BaseModel):
    email: Optional[str] = ''
    username: Optional[str] = ''

1 usage  ✨ Jakub Zurawski +1
class UpdateMe(UserUpdateModel):
    current_password: str
    password: Optional[str] = ''
    re_password: Optional[str] = ''
```

Rysunek 7.11: Implementacja klas modeli ”UserUpdateModel” i ”UpdateMe” w backendzie.

```
✉ Jakub Zurawski +2
@router.put(path: "/me/", response_model=UserResponse, status_code=200)
async def update_me(
    payload: UpdateMe,
    user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    payload = payload.dict()
    user_details: User = db.query(User).filter(User.id == user.id).first()

    if user_details:
        if not user_details.verify_password(payload['current_password']):
            raise HTTPException(
                status_code=status.HTTP_400_BAD_REQUEST,
                detail="Invalid password"
            )

        if payload.get('password') and payload.get('re_password'):
            if payload['password'] != payload['re_password']:
                raise HTTPException(
                    status_code=status.HTTP_400_BAD_REQUEST,
                    detail="Passwords do not match"
                )
            user_details.password = get_password_hash(payload['password'])

        if payload.get('username'):
            if get_user_by_username(payload['username'], db):
                raise HTTPException(
                    status_code=status.HTTP_400_BAD_REQUEST,
                    detail="Username already taken"
                )
            user_details.username = payload['username']

        if payload.get('email'):
            if get_user(payload['email'], db):
                raise HTTPException(
                    status_code=status.HTTP_400_BAD_REQUEST,
                    detail="Email already taken"
                )
            user_details.email = payload['email']

        user_details.updated_at = datetime.today()
        db.commit()
        db.refresh(user_details)

    return user_details

raise HTTPException(status_code=404, detail="User not found")
```

Rysunek 7.12: Implementacja kodu zmiany danych użytkownika w backendzie.

7.3.4.2 Aplikacja mobilna i aplikacja webowa

W celu zmiany danych użytkownika, takich jak avatar, nazwa użytkownika, adres e-mail czy hasło, użytkownik może skorzystać z odpowiednich formularzy dostępnych na stronie profilu użytkownika. Każda zmiana (za wyjątkiem zmiany avatara) wymaga uwierzytelnienia przez podanie aktualnego hasła. Poniżej opisane są poszczególne procesy.

```
Show usages ⚡ Jakub Zurawski
const handleUpdateAvatar = async(avatar: string) : Promise<void> => {
    await UsersService.updateUserAvatar(userData.id, avatar, navigation);
    const getMeData = await UsersService.getMe(navigation)
    await ActiveUser.updateUserData(getMeData);
    await getUserData();
    setShowAvatarModal( value: false);
}
```

Rysunek 7.13: Funkcja ”handleAvatarSelect” zastosowany w aplikacji mobilnej.

```
const handleAvatarSelect = async (selectedAvatar: string) => {
    setIsLoading( value: true);
    try {
        await AuthService.updateAvatar(userId, body: { avatar: selectedAvatar });
        // @ts-ignore
        setAvatar(AVATAR_MAPPING[selectedAvatar]);
        handleClose( modalType: 'avatar');
        setIsLoading( value: false);
    } catch (error) {
        // @ts-ignore
        setPasswordError( value: "Failed to update avatar: " + error.message);
        setIsLoading( value: false);
    }
};
```

Rysunek 7.14: Funkcja ”handleAvatarSelect” zastosowany w aplikacji webowej.

Zmiana avatara

Aby zmienić avatar, użytkownik musi wybrać nowy avatar z dostępnych opcji. Po dokonaniu wyboru aplikacja wysyła żądanie do serwera w celu aktualizacji avatara. Po pomyślnej aktualizacji użytkownik zobaczy nowy avatar na stronie profilu oraz w widoku statystyk.

Zmiana pseudonimu

Aby zmienić pseudonim, użytkownik musi wprowadzić nowy pseudonim oraz aktualne hasło w odpowiednim formularzu. Po zatwierdzeniu formularza dane są walidowane:

- Pseudonim musi zawierać 3-20 znaków i składać się wyłącznie z liter, cyfr lub podkreśników;
- Aktualne hasło musi być poprawne.

Jeśli dane są poprawne, aplikacja wysyła żądanie do serwera w celu aktualizacji pseudonimu. Po pomyślnej aktualizacji nowy pseudonim będzie wyświetlany na stronie profilu.

Zmiana adresu e-mail

Aby zmienić adres e-mail, użytkownik musi wprowadzić nowy adres e-mail oraz aktualne hasło w odpowiednim formularzu. Po zatwierdzeniu formularza dane są walidowane:

- Adres e-mail musi mieć poprawny format;
- Aktualne hasło musi być poprawne.

Jeśli dane są poprawne, aplikacja wysyła żądanie do serwera w celu aktualizacji adresu e-mail. Po pomyślnej aktualizacji użytkownik będzie logował się nowym adresem e-mail.

Zmiana hasła

Aby zmienić hasło, użytkownik musi wprowadzić aktualne hasło, nowe hasło oraz potwierdzenie nowego hasła w odpowiednim formularzu. Po zatwierdzeniu formularza dane są walidowane:

- Nowe hasło musi mieć co najmniej 8 znaków;
- Potwierdzenie nowego hasła musi być zgodne z nowym hasłem;
- Aktualne hasło musi być poprawne.

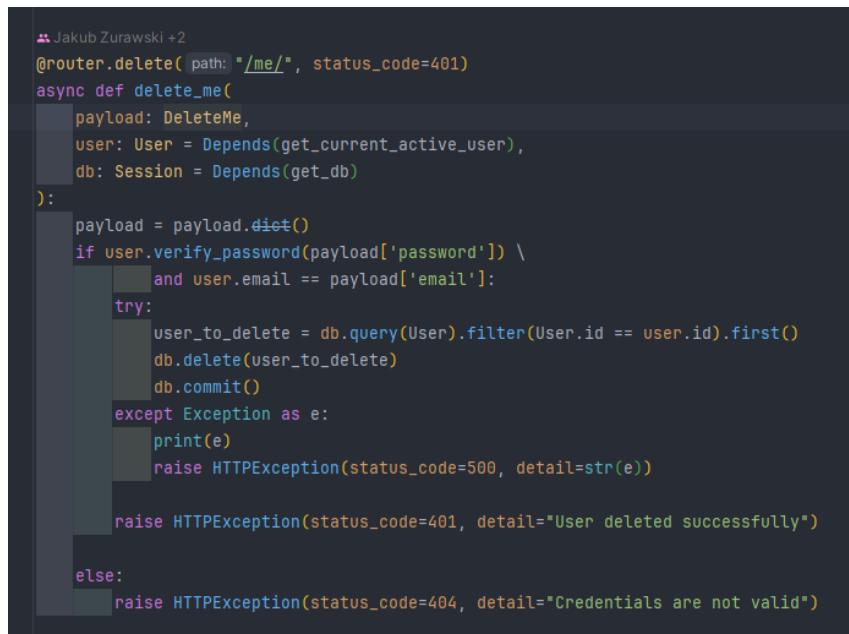
Jeśli dane są poprawne, aplikacja wysyła żądanie do serwera w celu aktualizacji hasła.

7.3.5 Usunięcie konta użytkownika

Funkcjonalność umożliwiająca usunięcie konta użytkownika.

7.3.5.1 Backend

Aplikacje frontendowe wysyłają na api e-mail użytkownika i hasło. Api sprawdza poprawność danych, następnie usuwa konto użytkownika.

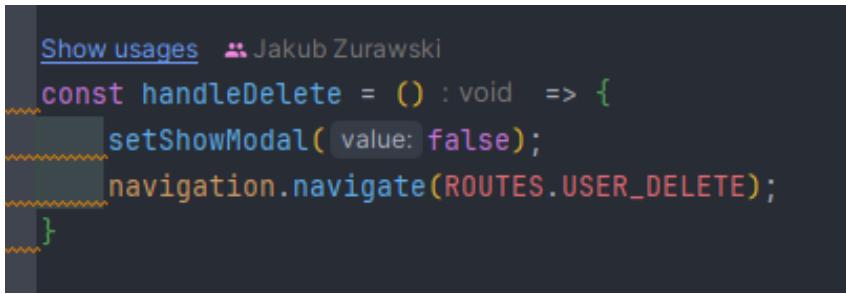


```
Jakub Zurawski +2
@router.delete(path: "/me/", status_code=401)
async def delete_me(
    payload: DeleteMe,
    user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    payload = payload.dict()
    if user.verify_password(payload['password']) \
        and user.email == payload['email']:
        try:
            user_to_delete = db.query(User).filter(User.id == user.id).first()
            db.delete(user_to_delete)
            db.commit()
        except Exception as e:
            print(e)
            raise HTTPException(status_code=500, detail=str(e))
        raise HTTPException(status_code=401, detail="User deleted successfully")
    else:
        raise HTTPException(status_code=404, detail="Credentials are not valid")
```

Rysunek 7.15: Implementacja kodu usunięcia użytkownika w backendzie.

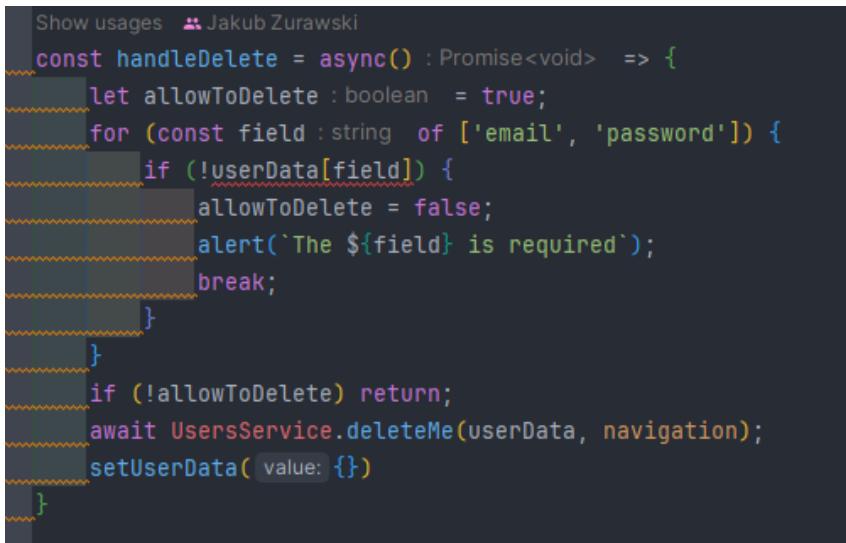
7.3.5.2 Aplikacja moblina

Użytkownik jest proszony o potwierdzenie czy aby na pewno chce usunąć konto. Po zatwierdzeniu zostaje przekierowany do formularza gdzie podaje e-mail oraz hasło, które po zatwierdzeniu zostają wysłane na api.



```
Show usages ↵ Jakub Zurawski
const handleDelete = () : void => {
  setShowModal( value: false);
  navigation.navigate(ROUTES.USER_DELETE);
}
```

Rysunek 7.16: Wywołanie funkcji "handleDelete" zastosowanego w aplikacji mobilnej.



```
Show usages ↵ Jakub Zurawski
const handleDelete = async() : Promise<void> => {
  let allowToDelete : boolean = true;
  for (const field : string of ['email', 'password']) {
    if (!userData[field]) {
      allowToDelete = false;
      alert(`The ${field} is required`);
      break;
    }
  }
  if (!allowToDelete) return;
  await UsersService.deleteMe(userData, navigation);
  setUserData( value: {})
```

Rysunek 7.17: Funkcja "handleDelete" zastosowany w aplikacji mobilnej.

7.3.5.3 Aplikacja webowa

Jeżeli po potwierdzeniu wyboru prawidłowym hasłem, funkcja wykryje błąd "error 401 (Unauthorized)", oznaczać to będzie że konto zostało usunięte. Następnie pojawia się komunikat o usunięciu który przekierowuje do strony logowania



```
const handleDeleteAccount = async () => {
  try {
    await AuthService.deleteAccount( body: { email: userEmail, password: currentPassword });
  } catch (error: any) {
    if (error.response?.status === 401) {
      setAlertMessage( value: "Account deleted");
      setOpenDeleteAccount( value: false);
    } else {
      setPasswordError( value: "Incorrect password");
      setCurrentPassword( value: "");
    }
  }
};
```

Rysunek 7.18: Funkcja "handleDelete" zastosowany w aplikacji webowej.

7.3.6 Tworzenie talii fiszek

Funkcjonalność umożliwiająca tworzenie talii fiszek.

7.3.6.1 Backend

Utworzono metodę POST, która umożliwia tworzenie talii o podanej nazwie i kategorii. Do tworzenia fiszki używana jest osobna metoda POST, która przyjmuje treść dla przedniej i tylnej strony karty, a także identyfikator talii do, której utworzona fiszka zostanie przypisana.

```
@router.post(path: "/create_deck", status_code=status.HTTP_201_CREATED)
async def create_deck(
    deck: DeckCreate,
    db: Session = Depends(get_db)
):
    """Create a new deck"""
    deck_model = Deck(**deck.dict())
    db.add(deck_model)
    db.commit()
    db.refresh(deck_model)
    return deck_model
```

Rysunek 7.19: Logika tworzenia decku po stronie serwera.

```
@router.post(path: "/create_flash_card", status_code=status.HTTP_201_CREATED)
async def create_flash_card(
    flash_card: FlashCardCreate,
    db: Session = Depends(get_db)
):
    """Create a new flash card"""
    flash_card_model = FlashCard(**flash_card.dict())
    db.add(flash_card_model)
    db.commit()
    db.refresh(flash_card_model)
    return flash_card_model
```

Rysunek 7.20: Logika tworzenia fiszek po stronie serwera.

7.3.6.2 Aplikacja mobilna

Dodanie fiszki jest możliwe po przejściu do widoku jej tworzenia z podglądu wybranej talii. Wymagane jest uzupełnienie pól przedniej i tylnej strony fiszki. Tworzenie odbywa się przez wysłanie danych do odpowiedniego endpointu.

```

const handleCreate = async () => {
  const trimmedSideA :string  = sideA.substring(0, 254);
  const trimmedSideB :string  = sideB.substring(0, 510);

  if (
    InputValidator( type: "deck", trimmedSideA ) &&
    InputValidator( type: "deck", trimmedSideB )
  ) {
    const flashcardData :{deck_id: any, card_title: str...  = {
      deck_id: deck.id,
      card_title: trimmedSideA,
      card_text: trimmedSideB,
    };

    try {
      await FlashCardsService.createFlashcard(flashcardData, navigation);
      navigation.goBack();
    } catch (error) {
      console.error("Failed to create flashcard:", error);
      alert("Failed to create flashcard.");
    }
  } else {
    alert("Flashcards fields must not be empty.");
  }
};

```

Rysunek 7.21: Funkcja obsługująca tworzenie nowej fiszki.

7.3.6.3 Aplikacja webowa

Użytkownik aby utworzyć talię musi wypełnić pole tekstowe dla nazwy i kategorii talii, następnie musi uzupełnić treść dla przedniej i tylnej strony fiszki. Po kliknięciu "Create deck" dane z formularzy zostają przekazane do funkcji, która zebrane dane przekazuje do serwisu odpowiedzialnego za komunikację z endpointem do tworzenia talii i endpointem do tworzenia fiszki.

```

public async create_deck(body: object) : Promise<{res: Response, data: ...} {
  // @ts-ignore
  return await request( {url, query, headers, method, body, formData, skipRedirect}: {
    url: AUTH_ENDPOINTS.create_deck,
    method: 'POST', body,
    headers: {
      'Authorization': `${token}`
    },
  });
}

```

Rysunek 7.22: Funkcja odpowiedzialna za komunikację z endpointem do tworzenia decku.

```
public async create_flash_card(body: object) : Promise<{res: Response, data: ...} > {
    // @ts-ignore
    return await request( {url, query, headers, method, body, formData, skipRedirect}: {
        url: AUTH_ENDPOINTS.create_flash_card,
        method: 'POST',
        body,
        headers: {
            'Authorization': `${token}`
        },
    });
}
```

Rysunek 7.23: Funkcja serwisu odpowiedzialna za komunikację z endpointem do tworzenia fiszki.

7.3.7 Usunięcie talii fiszek

Funkcjonalność umożliwiająca usunięcie talii fiszek.

7.3.7.1 Backend

Został utworzony endpoint `delete_deck`, która pozwala na usunięcie talii o podanym identyfikatorze wraz z wszystkimi powiązanymi fiszkami.

```
@router.delete("/delete_deck/{delete_id}")
async def delete_deck(
    delete_id: uuid.UUID,
    db: Session = Depends(get_db)
):
    """Delete deck and all flash cards associated with it"""
    deck = db.query(Deck).filter(Deck.id == delete_id).first()
    if deck is None:
        raise HTTPException(status_code=404, detail="Deck not found")

    db.query(FlashCard).filter(FlashCard.deck_id == delete_id).delete()

    db.query(Deck).filter(Deck.id == delete_id).delete()
    db.commit()
```

Rysunek 7.24: Logika usuwająca talie i wszystkie należące do niej fiszki.

7.3.7.2 Aplikacja moblina

Usunięcie talii jest możliwe z poziomu jej opcji. Po wybraniu tej opcji do api zostaje wysłane zapytanie o usunięcie talii. W zapytaniu przekazywany jest jej identyfikator.

```

const deleteDeckFromApi = async (deck) => {
  const deck_id = deck.id;

  try {
    await DecksService.delete_deck(deck_id, navigation);
    alert("Deck deleted successfully.");
    navigation.goBack();
    navigation.goBack();
  } catch (error) {
    console.error("Failed to delete deck:", error);
    alert("Failed to delete deck.");
  }
};

```

Rysunek 7.25: Funkcja odpowiedzialna za obsługę usunięcia decku.

7.3.7.3 Aplikacja webowa

Użytkownik aby usunąć talię kliką w opcje, następnie wybiera "Delete deck". Po kliknięciu, identyfikator aktualnej talii zostaje pobrany z pamięci lokalnej i przekazane do serwisu, który łączy się z bąkcentowym endpointem do usuwania talii. Endpoint usuwa z bazy danych talię o podanym identyfikatorze i wszystkie należące do niego fiszki.

```

public async deleteDeck(deck_id: string) : Promise<{res: Response, data: ...}> {
  try {
    const url : string = `${BASE_API}/decks/delete_deck/${deck_id}`;
    return await request({url, query, headers, method, body, formData, skipRedirect}: {
      url,
      method: 'DELETE',
      headers: {
        'Authorization': `${token}`
      },
    });
  } catch (error) {
    // @ts-ignore
    console.error(error.message);
    throw error;
  }
}

```

Rysunek 7.26: Funkcja serwisu odpowiedzialna za komunikację z endpointem do usuwania talii.

7.3.8 Edycja talii fiszek

Funkcjonalność umożliwiająca aktualizowanie zawartości talii fiszek.

7.3.8.1 Backend

Zostały utworzone metody PUT, pozwalające na komunikację z bazą danych w celu aktualizacji informacji o fiszkach i taliach.

```

@router.put("/update_deck/{deck_id}")
async def update_deck(
    deck_id: uuid.UUID,
    deck_data: DeckUpdate,
    db: Session = Depends(get_db)
):
    deck_model = db.query(Deck).filter(Deck.id == deck_id).first()
    if deck_model is None:
        raise HTTPException(status_code=404, detail="Deck not found")

    if deck_data.title is not None:
        deck_model.title = deck_data.title
    if deck_data.deck_category is not None:
        deck_model.deck_category = deck_data.deck_category
    if deck_data.is_deck_public is not None:
        deck_model.is_deck_public = deck_data.is_deck_public
    if deck_data.downloads is not None:
        deck_model.downloads = deck_data.downloads

    db.commit()

```

Rysunek 7.27: Metoda aktualizująca dane talii o podanym identyfikatorze.

```

@router.put("/update_flash_card_text/{flash_card_id}")
async def update_flash_card_text(
    flash_card_id: uuid.UUID,
    flash_card_data: FlashCardUpdateText,
    db: Session = Depends(get_db)
):
    """
    Update a single flash card by its ID.
    """
    flash_card_model = db.query(FlashCard).filter(FlashCard.id == flash_card_id).first()

    if flash_card_model is None:
        raise HTTPException(
            status_code=404,
            detail=f"Flash card with ID {flash_card_id} not found"
        )

    if flash_card_data.card_title is not None:
        flash_card_model.card_title = flash_card_data.card_title
    if flash_card_data.card_text is not None:
        flash_card_model.card_text = flash_card_data.card_text

    db.commit()

```

Rysunek 7.28: Metoda PUT aktualizująca dane fiszki.

7.3.8.2 Aplikacja moblina

Aby zmienić nazwę lub kategorię talii, użytkownik musi wybrać widok "Edit" w jej opcjach. Aktualizacja danych odbywa się za zmianą treści pól "nazwa" i "kategoria" po zaakceptowaniu zmian, te wysyłane są na endpoint.

```

const handleEdit = async () => {
  if (InputValidator(type: "deck", title) && InputValidator(type: "deck", category)) {
    const deck_data : {title: any, deck_category: any} = {
      title,
      deck_category: category,
    };
    try {
      await DecksService.update_deck_title_category(
        deck.id,
        deck_data,
        navigation,
      );
      navigation.goBack();
    } catch (error) {
      alert("Update deck failed");
    }
  } else {
    alert("Flashcards fields must not be empty.");
  }
};

```

Rysunek 7.29: Funkcja obsługująca aktualizację danych talii.

7.3.8.3 Aplikacja webowa

Użytkownik w celu zmiany nazwy talii lub kategorii musi otworzyć okno opcji następnie kliknąć przycisk zmiany danych. Po wykonaniu tych operacji pojawia się formularz do zmiany nazwy talii i kategorii. Użytkownik może podać nowe dane i kliknąć przycisk "Accept". Nowe dane zostają przekazane do funkcji zmiany nazwy i kategorii następnie przechwycone dane zostają przekazane do serwisu, który łączy się z backendowym endpointem. Następnie endpoint łączy się z bazą danych w celu aktualizacji talii o otrzymane dane. Tak samo wygląda proces aktualizacji danych fiszki, po kliknięciu ikonę ołówkiem znajdującym się na fiszce, pojawia się formularz do aktualizacji danych fiszki.

```

public async update_deck_title_category(deck_id: string, body: object) : Promise<{res: Response, data: ...}> {
  const url : string = `${BASE_API}/decks/update_deck/category_and_title/${deck_id}`;
  console.log(body)
  try {
    // @ts-ignore
    return await request({url, query, headers, method, body, formData, skipRedirect}: {
      url: url,
      method: 'PUT',
      body: body,
      headers: {
        'Authorization': `${token}`
      },
    });
  } catch (error) {
    // @ts-ignore
    console.error(error.message);
  }
}

```

Rysunek 7.30: Funkcja służąca do komunikacji z backendowym endpointem do aktualizacji danych talii.

```

public async update_single_flash_card(flashcard_id: string, body: object) {
  const url :string = `${BASE_API}/flash_card/update_flash_card_text/${flashcard_id}`;

  try {
    // @ts-ignore
    return await request( {url, query, headers, method, body, formData, skipRedirect}: {
      url,
      method: 'PUT',
      body,
      headers: {
        'Authorization': `${token}`
      },
    });
  } catch (error) {
    // @ts-ignore
    console.error(error.message);
  }
}

```

Rysunek 7.31: Funkcja służąca do komunikacji z backendowym endpointem do aktualizacji danych fiszki.

7.3.9 Tryb uczenia się z talii fiszek

7.3.9.1 Backend

Model fiszki zawiera kolumnę `is_memorized`, która przy tworzeniu fiszki jest domyślnie ustawiona na "false", wartość ta oznacza, że użytkownika jeszcze nie nauczył się danej treści fiszki. Jeżeli użytkownik uzna, że fiszka została przez niego zapamiętana, to wartość kolumny `is_memorized` po zakończonym uczeniu się zostanie zaktualizowana na wartość true. Do aktualizacji danych fiszek została wykorzystany endpoint, który przyjmuje listę fiszek i następnie aktualizuje dane wszystkich fiszek zawartych w przesłanej liście.

```

@router.put( path: "/update_flash_cards", status_code=status.HTTP_204_NO_CONTENT)
async def update_flash_cards(
    flash_cards_data: List[FlashCardUpdate],
    db: Session = Depends(get_db)
):
    """
    Update multiple flashcards
    """
    for flash_card in flash_cards_data:
        flash_card_model = db.query(FlashCard).filter(FlashCard.id == flash_card.id).first()
        if flash_card_model is None:
            raise HTTPException(status_code=404, detail=f"Flash card with ID {flash_card.id} not found")

        if flash_card.card_title is not None:
            flash_card_model.card_title = flash_card.card_title
        if flash_card.card_text is not None:
            flash_card_model.card_text = flash_card.card_text

        flash_card_model.is_memorized = flash_card.is_memorized

    db.commit()

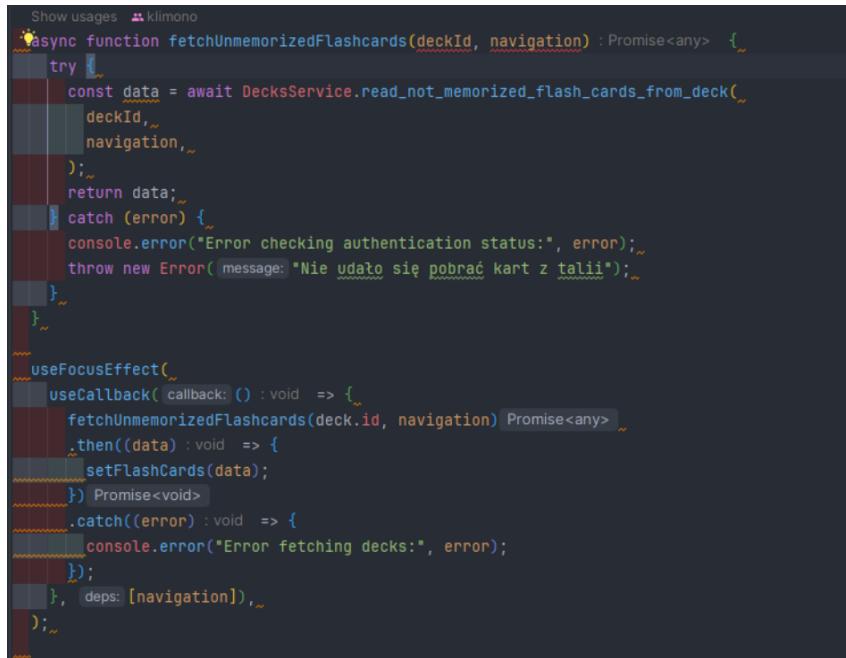
```

Rysunek 7.32: Logika aktualizacji przesłanych fiszek.

7.3.9.2 Aplikacja moblina

Po przejściu do widoku nauki, aplikacja ściąga fiszki użytkownika. W trakcie nauki użytkownik może wybrać dwie opcje "remember" lub "not remember". W momencie kliknięcia przycisku "remember" informacje o fiszce zostaną dodane do listy nauczonych fiszek, które potem zostaną zaktualizowane w bazie.

Po zakończeniu nauki, aplikacja ściąga na nowo listę fiszek i filtryuje je po polu `is_memorized`. Fiszki które mają ustawione `is_memorized` na "true", nie zostaną ponownie wyświetlane w trybie nauki.



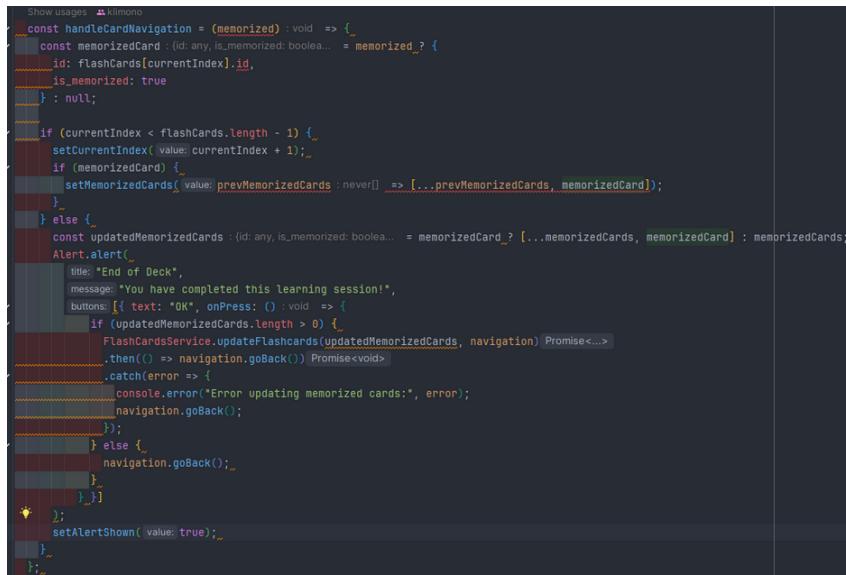
```

Show usages ⚡ klimono
async function fetchUnmemorizedFlashcards(deckId, navigation) : Promise<any> {
  try {
    const data = await DecksService.read_not_memorized_flash_cards_from_deck(
      deckId,
      navigation,
    );
    return data;
  } catch (error) {
    console.error("Error checking authentication status:", error);
    throw new Error(message: "Nie udało się pobrać kart z talii");
  }
}

useFocusEffect(
  useCallback(() : void => {
    fetchUnmemorizedFlashcards(deck.id, navigation).Promise<any>
      .then((data) : void => {
        setFlashCards(data);
      })
      .catch((error) : void => {
        console.error("Error fetching decks:", error);
      });
  }, [navigation]),
);

```

Rysunek 7.33: Funkcja ściągająca niezapamiętane fiszki.



```

Show usages ⚡ klimono
const handleCardNavigation = (memorized) : void => {
  const memorizedCard : {id: any, isMemorized: boolean} = memorized ? {
    id: flashCards[currentIndex].id,
    isMemorized: true
  } : null;

  if (currentIndex < flashCards.length - 1) {
    setCurrentIndex(value: currentIndex + 1);
    if (memorizedCard) {
      setMemorizedCards(value: prevMemorizedCards : never[] => [...prevMemorizedCards, memorizedCard]);
    }
  } else {
    const updatedMemorizedCards : {id: any, isMemorized: boolean} = memorizedCard ? [...memorizedCards, memorizedCard] : memorizedCards;
    Alert.alert(
      title: "End of Deck",
      message: "You have completed this learning session!",
      buttons: [{text: "OK", onPress: () : void => {
        if (updatedMemorizedCards.length > 0) {
          FlashCardsService.updateFlashcards(updatedMemorizedCards, navigation).Promise<any>
            .then(() => navigation.goBack())
            .catch(error => {
              console.error("Error updating memorized cards:", error);
              navigation.goBack();
            });
        } else {
          navigation.goBack();
        }
      }}]
    );
    setAlertShown(value: true);
  }
};

```

Rysunek 7.34: Funkcja aktualizacji zapamiętanych fiszek.

7.3.9.3 Aplikacja webowe

Po uruchomieniu trybu uczenia użytkownik widząc fiszkę może kliknąć przycisk "remember" lub "not remember". W przypadku kliknięcia przycisku "remember" informacje o fiszce zostaną dodane do listy fiszek, które zostaną zaktualizowane. Po zakończeniu trybu uczenia list z fiszkami, które zostały oznaczone jako zapamiętane zostaje przekazana do serwisu, który następnie przekazuje dane do bakkendowego

endpointu w celu aktualizacji danych fiszek w bazie danych. Kolumna `is_memorized` wszystkich przekazanych fiszek zostaje ustawiona na wartość true. Fiszki, których kolumna `is_memorized` jest ustawiona na true nie wyświetla się w trybie uczenia.

```
const fetchFlashCards = async () : Promise<void> => {
    try {
        let deck_id: string;

        const intervalId: NodeJS.Timeout = setInterval( callback: () : void => {
            const deckDataString: string | null = localStorage.getItem( key: "deckData" );
            const deckData = JSON.parse( text: deckDataString || "{}" );
            deck_id = deckData.id;
            setDeckTitle(deckData.title);
            if (deck_id) {
                clearInterval(intervalId);
                setTimeout( callback: async () : Promise<void> => {
                    const response: DeckData = await DeckService.get_not_memorized_flash_cards_from_deck(deck_id);
                    // @ts-ignore
                    setFlashcards(response);

                    setIsLoading( value: false )
                }, ms: 300 );
            }
        }, ms: 100 );
    } catch (error) {
        console.error(error);
    }
};

fetchFlashCards();
```

Rysunek 7.35: Funkcja ściągająca niezapamiętane fiszki.

```
Show usages ▾ kimono
const handleCardNavigation = (memorized) : void => {
    const memorizedCard : (id: any, is_memorized: boolean) = memorized ? {
        id: flashCards[currentIndex].id,
        is_memorized: true
    } : null;

    if (currentIndex < flashCards.length - 1) {
        setCurrentIndex( value: currentIndex + 1 );
        if (memorizedCard) {
            setMemorizedCards( value: prevMemorizedCards : never[] => [...prevMemorizedCards, memorizedCard] );
        }
    } else {
        const updatedMemorizedCards : (id: any, is_memorized: boolean) = memorizedCard ? [...memorizedCards, memorizedCard] : memorizedCards;
        Alert.alert(
            title: "End of Deck",
            message: "You have completed this learning session!",
            buttons: [ { text: "OK", onPress: () : void => {
                if (updatedMemorizedCards.length > 0) {
                    FlashCardsService.updateFlashcards(updatedMemorizedCards, navigation) Promise<...>
                    .then(() => navigation.goBack()) Promise<void>
                    .catch(error => {
                        console.error("Error updating memorized cards:", error);
                        navigation.goBack();
                    });
                }
            } ];
        );
    }
    setAlertShown( value: true );
};
```

Rysunek 7.36: Funkcja aktualizacji zapamiętyanych fiszek.

```

public async update_multiple_flash_card(body: object) : Promise<{res: Response, data: ...}> {
    const url :string = `${BASE_API}/flash_card/update_flash_cards`;
    try {
        // @ts-ignore
        return await request({url, query, headers, method, body, formData, skipRedirect}: {
            url: url,
            method: 'PUT',
            body: body,
            headers: {
                'Authorization': `${token}`
            },
        });
    } catch (error) {
        // @ts-ignore
        console.error(error.message);
    }
}

```

Rysunek 7.37: Funkcja aktualizacji zapamiętanych fiszek.

7.3.10 Sterowanie talią przy użyciu mowy

Funkcjonalność umożliwiająca sterowania talią przy użyciu komend głosowych.

7.3.10.1 Backend

Do wczytania modelu nlp została wykorzystana biblioteka "flair". Do zamiany tekstu na wektory został użyty model "BERT". Została utworzona lista wyrażeń i komend im odpowiadających. Lista ta zostanie wykorzystana do dopasowania tego co mówi użytkownik do odpowiednich komend głosowych. Funkcja `vectorize_text()` odpowiada za konwertowanie tekstu na wektor. Następnie przy użyciu pętli for i funkcji `vectorize_text()` zdania w liście są konwertowane na wektory. Następnie została utworzona funkcja `get_most_similar_answer()`, która przyjmuje zdanie, które wypowiedział użytkownik, i listę wyrażeń. Funkcja ta zmienia zdanie wypowiedzione przez użytkownika na wektor, po czym tworzona jest pusta listę do przechowywania wartości podobieństw pomiędzy wypowiedzią użytkownika a wyrażeniami zawartymi w liście. Pętla for przechodzi po wszystkich wektorach zdań w liście i liczona jest odległość cosinusowa pomiędzy wektorem zdania wypowiadzanego przez użytkownika a wektorem z zdania z listy wyrażeń. Odległość kosinusowa jest używana w analizie tekstu, jest to miara określenia różnicy między dwoma wektorami w przestrzeni n-wymiarowej [7]. Podobieństwo jest liczone ze wzoru 1 - odległość kosinusowa obliczona na podstawie wektora wypowiedzi użytkownika i wektora analizowanego zdania z listy wyrażeń. Obliczone podobieństwo dodawane jest do listy wyrażeń. Następnie przy użyciu `np.argmax()` zwracany jest indeks na, którym miejscu jest wartość z największym prawdopodobieństwem. Mając indeks funkcja zwraca komendę, która najbardziej odpowiada zdaniu wypowiadanemu przez użytkownika. Utworzona została metoda POST, która przyjmuje zdanie wypowiadane przez użytkownika i uruchamia analizę tekstu.

```
18     embedding = TransformerDocumentEmbeddings('bert-base-uncased')
19
20     qa_pairs = [
21         {"q": "next", "a": "next"},  

22         {"q": "next card", "a": "next"},  

23         {"q": "next flashcard", "a": "next"},  

24         {"q": "show next card", "a": "next"},  

25         {"q": "show me next card", "a": "next"},  

26         {"q": "show next flashcard", "a": "next"},  

27         {"q": "show me next flashcard", "a": "next"},  

28         {"q": "please show me next card", "a": "next"},  

29         {"q": "please show me forward card", "a": "next"},  

30         {"q": "please show me Subsequent card", "a": "next"},  

31         {"q": "previous", "a": "previous"},  

32         {"q": "previous card", "a": "previous"},  

33         {"q": "previous flashcard", "a": "previous"},  

34         {"q": "show me previous card", "a": "previous"},  

35         {"q": "please show me pior card", "a": "previous"},  

36         {"q": "please show me former card", "a": "previous"},  

37         {"q": "show me previous flashcard", "a": "previous"},  

38         {"q": "please show me previous card", "a": "previous"},  

39         {"q": "please show me preceding card", "a": "previous"},  

40         {"q": "turn", "a": "rotate"},  

41         {"q": "spin", "a": "rotate"},  

42         {"q": "rotate", "a": "rotate"},  

43         {"q": "turn card", "a": "rotate"},  

44         {"q": "spin card", "a": "rotate"},  

45         {"q": "rotate card", "a": "rotate"},  

46         {"q": "spin flashcard", "a": "rotate"},  

47         {"q": "rotate flashcard", "a": "rotate"},  

48         {"q": "please turn card", "a": "rotate"},  

49         {"q": "rotate flashcard", "a": "rotate"},  

50         {"q": "please rotate card", "a": "rotate"},  

51         {"q": "please spin card", "a": "rotate"},  

52         {"q": "read", "a": "read"},
```

Rysunek 7.38: Inicjalizacja modelu wraz z listą wyrażeń i odpowiadających im komend.

```

63     def vectorize_text(text):
64         sentence = Sentence(text)
65         embedding.embed(sentence)
66         return sentence.embedding.cpu().detach().numpy()
67
68
69     for pair in qa_pairs:
70         pair['q_vectorized'] = vectorize_text(pair['q'])
71
72     # usage: `OliwierKossak
73     def get_most_similar_answer(user_question, qa_pairs):
74
75         user_question_vectorized = vectorize_text(user_question)
76
77         similarities = []
78         for pair in qa_pairs:
79             similarity = 1 - cosine(user_question_vectorized, pair['q_vectorized'])
80             similarities.append(similarity)
81
82
83         most_similar_index = np.argmax(similarities)
84         return qa_pairs[most_similar_index]['a']
85
86
87     # OliwierKossak *
88     @router.post(path: "/calculate_similarity", status_code=status.HTTP_200_OK)
89     async def calculate_semantic_similarity(
90         text: SimilarityText
91     ):
92         try:
93             answer = get_most_similar_answer(text.text, qa_pairs)
94             return answer
95         except:
96             return "not found command"

```

Rysunek 7.39: Kod odpowiedzialny za analizę tekstu w celu znalezienia najlepiej dopasowanej komendy.

7.3.10.2 Aplikacja mobilna

Aby obsłużyć głosowy tryb nauki, aplikacja mobilna korzysta z biblioteki "expo-speech" i "expo-av". Po naciśnięciu przycisku z ikoną mikrofonu aplikacja zaczyna nasłuchiwać komend. Z powodu ograniczeń związanych ze środowiskiem Expo, głos nie jest strumieniowany bezpośrednio do modelu nlp który obsługuje backend. Aplikacja w trakcie nasłuchiwanego wykonuje krótkie nagrania, które są tymczasowo zapisywane i wysypane jako plik audio m4a do backendu. Po ich zinterpretowaniu backend odsyła komendę do której przypisuje znaczenie treści nagrania. Następnie komenda jest wykonywana w widoku nauki.

```
const startRecording = async () => {
  try {
    if (Platform.OS === "ios") {
      await Audio.setAudioModeAsync({ partialMode: {
        allowsRecordingIOS: true,
        playsInSilentModeIOS: true,
        staysActiveInBackground: true,
      }});
    } else {
      await Audio.setAudioModeAsync({ partialMode: {
        staysActiveInBackground: true,
      }});
    }
    console.log("Starting recording...");
    const recordingInstance : Audio.Recording = new Audio.Recording();
    await recordingInstance.prepareToRecordAsync(
      Audio.RecordingOptionsPresets.HIGH_QUALITY,
    );
    await recordingInstance.startAsync();
    recording.current = recordingInstance;
    console.log("Recording started");
  } catch (err) {
    console.error("Failed to start recording", err);
  }
};
```

Rysunek 7.40: Funkcja ”startRecording” odpowiedzialna za rozpoczęcie nasłuchiwania.

```

const stopRecording = async () => {
  try {
    console.log("Stopping recording...", recording);
    if (recording.current) {
      await recording.current.stopAndUnloadAsync();
      await Audio.setAudioModeAsync({ partialMode: {
        allowsRecordingIOS: false,
        playsInSilentModeIOS: false,
        staysActiveInBackground: false,
      }});
      const uri = recording.current.getURI();
      recording.current = null;
      setActiveVoiceControlMode(VOICE_CONTROL_STAGES.STOP);
      if (uri) {
        console.log(uri, typeof uri);
        await calculateSimilarity(uri);
      }
    } else {
      setActiveVoiceControlMode(VOICE_CONTROL_STAGES.STOP);
      console.error("No recording instance found");
    }
  } catch (err) {
    console.error("Failed to stop recording", err);
  }
};

```

Rysunek 7.41: Funkcja "stopRecording" odpowiedzialna za zakończenie nasłuchiwania.

```

const calculateSimilarity = async (recordingURI: string) => {
  if (!recordingURI) {
    console.error("No recording URI to upload");
    return;
  }
  try {
    const fileInfo: FileSystem.FileInfo = await FileSystem.getInfoAsync(recordingURI);
    if (!fileInfo.exists) {
      console.error("File does not exist at the given URI");
      return;
    }
    const fileUri: string = fileInfo.uri;
    const formData: FormData = new FormData();
    formData.append("name: "file", value: {
      uri: fileUri,
      name: "recording.m4a",
      type: "audio/m4a",});
    const { res, data } = await NlpService.calculateSimilarityByFile(
      formData,
      navigation,);
    await FileSystem.deleteAsync(recordingURI);
    if ([200, 201].includes(res.status)) {
      console.log("File uploaded successfully");
      await handleCommands(data.command);
      if (data.command || data.command !== "stop") {
        setActiveVoiceControlMode(VOICE_CONTROL_STAGES.START);}
    } else {
      await handleCommands( command: "Unknown");
    }
  } catch (err) {console.error("Something went wrong during the calculation", err);}
};

```

Rysunek 7.42: Funkcja "calculateSimilarity" obsługująca wysłanie nagrania głosowego do modelu nlp.

```

const handleCommands = async (command: string) => {
  switch (command) {
    case "previous":
      await handlePrevClick();
      break;
    case "next":
      await handleNextClick();
      break;
    case "rotate":
      handleRotateClick();
      break;
    case "read":
      handleSpeakerClick();
      break;
    case "stop":
      await handleStopControl();
      break;
    default:
      handleSpeech({ text: "Sorry, I didn't recognize the command" });
      console.warn({ message: "Command not found" });
      break;
  }
};

```

Rysunek 7.43: Funkcja ”handleCommands” obsługująca nawigację w trybie nauki głosowej.

7.3.10.3 Aplikacja webowa

Do zamiany mowy na tekst, który potem jest analizowany przez model nlp w celu wybrania komendy została wykorzystana biblioteka web speech api. Na początku została utworzona referencja recognition do przechowywania instancji SpeechRecognition. Po kliknięciu przycisku ”start listening” zmienna ”isListening” zostaje ustaliona na true rozpoczyna się nasłuchiwanie mowy przy użyciu `recognition.current.start()`. To co mówi użytkownik jest zapisywane w zmiennej ”textControl”. Następnie gdy wartość `textControl` zostaje wypełniona treścią wypowiadzaną przez użytkownika zostanie to przekazane do funkcji `nlpModelControl()`. Funkcja ta wysyła wypowiedź użytkownika do modelu nlp na backendzie, model nlp analizy treści i dopasowuje zdanie, które jest najbliższego semantycznie do wypowiedzi użytkownika. Następnie sprawdzana jest komenda, która przypisana jest do wybranego zdania. Komenda zostaje przesłana na frontend i przekazana do funkcji `voiceControl()`, która zawiera switch case, przesłana komenda uruchamia odpowiednią akcję na stronie internetowej.

```

const recognition: React.MutableRefObject<null> = useRef({ initialValue: null });

```

Rysunek 7.44: Referencja ”recognition”.

```

useEffect( effect: () : void  => {
    if (isListening) {
        // @ts-ignore
        recognition.current.start();
    } else {
        window.speechSynthesis.cancel()
        // @ts-ignore
        recognition.current.stop();
    }
},  deps: [isListening]);

```

Rysunek 7.45: Rozpoczęcie nasłuchiwania.

```

62  useEffect( effect: () => {
63      if ("SpeechRecognition" in window || "webkitSpeechRecognition" in window) {
64          // @ts-ignore
65          recognition.current = new (window.SpeechRecognition || window.webkitSpeechRecognition());
66          // @ts-ignore
67          recognition.current.lang = 'en-US';
68          // @ts-ignore
69          recognition.current.continuous = true;
70          // @ts-ignore
71          recognition.current.onresult = (event) : void  => {
72              let finalTranscriptText : string  = "";
73              for (let i = event.resultIndex; i < event.results.length; ++i) {
74                  if (event.results[i].isFinal) {
75                      finalTranscriptText += event.results[i][0].transcript + " ";
76                  }
77              }
78              let trimmedText : string  = finalTranscriptText.trim();
79              setTextControl( value:isSpeakingBigCard ? '' : trimmedText);
80              if (isSpeakingBigCard) {
81                  trimmedText = '';
82              }
83          };
84          // @ts-ignore
85          recognition.current.addEventListener('end', () : void  => {
86              if (isListening) {
87                  // @ts-ignore
88                  recognition.current.start();
89              }
90          });
91          return () : void  => {
92              if (recognition.current) {
93                  // @ts-ignore
94                  recognition.current.stop();
95              }
96          };
97      } else {
98          alert("Browser not support (Speech Recognition API).");
99      }
}

```

Rysunek 7.46: Zapisywanie tego co mówi użytkownik do zmiennej "textControl".

```
useEffect( effect: () : void  => {
    if (textControl.length > 2) {
        nlpModelControl(textControl);
    }
}, [deps: [textControl]]);
```

Rysunek 7.47: Przekazanie zawartości textControl do ”nlpModelControl”.

```
+ usages - 2 OliwierKossak +1
const nlpModelControl = async (text: string) : Promise<void>  => {

    try {
        let body : {text: string}  = {
            text : text
        }

        const nlp_answer : {res: Response, data: any, hea...  = await NlpService.sent_message(body)
        // @ts-ignore
        voiceControl(nlp_answer?.data);

    } catch (error) {
        console.error("Error:", error);
    }
};
```

Rysunek 7.48: Komenda wybrana przez model nlp zostaje przekazana do ”voiceControl()”.

```
public async sent_message(body: object) : Promise<{res: Response, data: ...  {
    const url : string  = `${BASE_API}/nlp/calculate_similarity`;
    try {
        // @ts-ignore
        return await request( {url, query, headers, method, formData, skipRedirect}: {
            url: url,
            method: 'POST', body
        });

    } catch (error) {
        // @ts-ignore
        console.error(error.message);
    }
}
```

Rysunek 7.49: Funkcja ”sent_message()” przekazuje wypowiedź użytkownika do modelu nlp na backen-
dzie.

```
const voiceControl = (text: string) : void => {

    const command : {previous: number, next: number...} = {
        "previous": 0,
        "next": 1,
        "rotate": 2,
        'read': 3,
        'stop': 4,
    }
    // @ts-ignore
    let number = command[text]
    if (!isSpeakingBigCard) {
        switch (number) {
            case 0:
                handlePrevClick();
                break;
            case 1:
                handleNextClick();
                break;
            case 2:
                handleRotateClick();
                break;
            case 3:
                handleSpeakerBigCardClick();
                break;
            case 4:
                handleStopControl();
                break;
            default:
                handleSpeakerNotRecognized();
                console.log('command not found');
                break;
        }
    }
};
```

Rysunek 7.50: Na podstawie otrzymanej komendy aplikacja wykonuje odpowiednią akcję.

7.3.11 Import talii innych użytkowników

Funkcjonalność pozwalająca użytkownikom na zapisanie talii udostępnionej publicznie.

7.3.11.1 Backend

Utworzona została metoda POST, która służy do kopiowania talii innych użytkowników. Przyjmuje ona identyfikator talii do skopiowania i identyfikator użytkownika, który tę talię chce zainportować. Metoda tworzy nową niezależną talią, która zostaje przypisana do użytkownika o podanym identyfikatorze. Pozycje w rankingu talii są przyznawane na podstawie liczby pobrań, natomiast pozycja w rankingu jest zależna od sumy pobrań wszystkich jego udostępnionych talii. Użytkownik ma teraz możliwość importowania talii fiszek. Pobrała talia zostaje dodana do widoku public decs.

```

@router.post(path: "/copy_deck/{deck_id}/{user_id}", status_code=status.HTTP_201_CREATED)
async def copy_deck(
    deck_id: uuid.UUID,
    user_id: uuid.UUID,
    db: Session = Depends(get_db)
):
    """Copy deck without modifying the original"""
    original_deck = db.query(Deck).filter(Deck.id == deck_id).first()
    if original_deck is None:
        raise HTTPException(status_code=404, detail="Original deck not found")

    original_deck.downloads += 1

    copied_deck = Deck(
        user_id=user_id,
        title=original_deck.title,
        deck_category=original_deck.deck_category,
        is_deck_public=False,
        is_created_by_user=False
    )
    db.add(copied_deck)
    db.commit()
    db.refresh(copied_deck)

    for original_flash_card in original_deck.flash_card_relationship:
        copied_flash_card = FlashCard(
            deck_id=copied_deck.id,
            card_title=original_flash_card.card_title,
            card_text=original_flash_card.card_text,
            is_memorized=False
        )
        db.add(copied_flash_card)

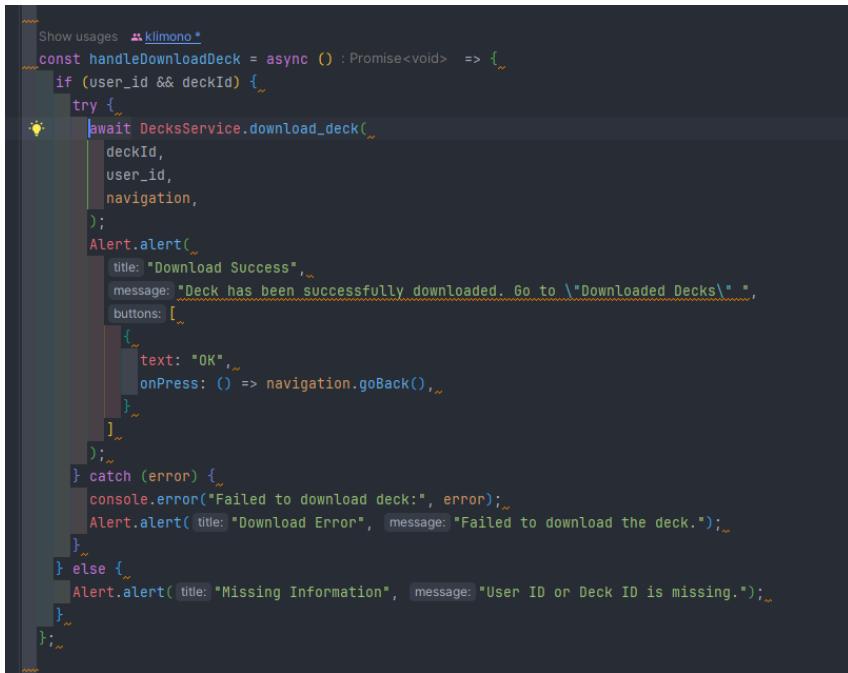
    db.commit()

```

Rysunek 7.51: Metoda POST do kopowania talii.

7.3.11.2 Aplikacja mobilna

Funkcja przekazuje do serwisu dane, takie jak identyfikator wybranej talii oraz identyfikator użytkownika, następnie wysyła request na api, aby dodać wybraną talię do listy użytkownika.



```
const handleDownloadDeck = async () : Promise<void> => {
  if (user_id && deckId) {
    try {
      await DecksService.download_deck(
        deckId,
        user_id,
        navigation,
      );
      Alert.alert({
        title: "Download Success",
        message: "Deck has been successfully downloaded. Go to \"Downloaded Decks\".",
        buttons: [
          {
            text: "OK",
            onPress: () => navigation.goBack(),
          },
        ],
      });
    } catch (error) {
      console.error("Failed to download deck:", error);
      Alert.alert({ title: "Download Error", message: "Failed to download the deck." });
    }
  } else {
    Alert.alert({ title: "Missing Information", message: "User ID or Deck ID is missing." });
  }
};
```

Rysunek 7.52: Funkcja obsługująca pobranie publicznej talii.



```
public async download_deck(deck_id: any, user_id: any, navigation: NavigationProp<any>) : Promise<{res: Response, data: any}> {
  return await request({
    url: DECKS_ENDPOINTS.download_deck(deck_id, user_id),
    method: "POST",
    navigation,
  });
}
```

Rysunek 7.53: Funkcja odpowiedzialna za pobieranie / kopiowanie talii.

7.3.11.3 Aplikacja webowa

Użytkownik żeby zaimportować talie musi kliknąć w przycisk "Import deck". Po kliknięciu w przycisk z pamięci lokalnej zostaje pobrany identyfikator talii i użytkownika. Te informacje zostają przekazane do serwisu, który łączy się z endpointem na backendzie i przekazuje dane talii i użytkownika potrzebne do skopiowania talii fiszek.



```
const handleImportPublicDeck = () : void => {
  const deckDataString : string | null = localStorage.getItem(key: "deckData");
  const deckData = JSON.parse(text: deckDataString || "{}");
  let deck_id = deckData.id;
  const userDataString : string | null = localStorage.getItem(key: "userData");
  const userData = JSON.parse(text: userDataString || "{}");
  let user_id = userData.id;

  DeckService.copy_public_deck(deck_id, user_id)
  navigate(to: '/decks_ranking')
}
```

Rysunek 7.54: Funkcja odpowiedzialna za aktywację serwisu do kopiowania talii.

```

public async copy_public_deck(deck_id: string, user_id: string) : Promise<(res: Response, data: ... > {
  const url : string = `${BASE_API}/decks/copy_deck/${deck_id}/${user_id}`

  return await request( {url, query, headers, method, body, formData, skipRedirect} ) {
    url,
    method: 'POST',
    headers: {
      'Authorization': `${token}`
    },
  );
}

```

Rysunek 7.55: Service do łączenia się z metodą "copy_deck" na backendzie.

7.3.12 Tryb dark mode i light mode

Funkcjonalność umożliwiająca zmianę stylu aplikacji na jasny lub ciemny. Zaimplementowana tylko w aplikacji mobilnej bez konieczności nawiązywania połączenia z backendem.

7.3.12.1 Aplikacja moblina

Przycisk do zmiany motywu został dodany do panelu użytkownika w sekcji ustawienia. Działanie jest proste, po kliknięciu następuje zmiana trybu na przeciwny. W zależności od aktywnego motywu, zmienia się ikona przycisku.

```

<Row className='w-full'/>
<Row className='w-full mt-1 mb-1'>
  <DarkMode/>
</Row>
<Row className='w-full mt-1 mb-1'>

Show usages ⚡ klimono
const DarkMode = () => {
  const { colorSchemeName, toggleColorScheme } = useColorScheme();
  return (
    <Button className='w-full p-2' onPress={toggleColorScheme}>
      <Text className='mx-5 font-bold text-xl'>
        {'Change mode! ' + (colorSchemeName === 'dark' ? '🌙' : '☀️')}
      </Text>
    </Button>
  );
}
Show usages ⚡ klimono
export default DarkMode;

```

Rysunek 7.56: Implementacja i wywołanie przycisku zmieniającego motyw aplikacji na ciemny.

7.3.13 Generowanie treści fiszki na podstawie słów kluczowych

Funkcjonalność, która pozwala na wygenerowanie treści fiszki używając ChatGPT.

7.3.13.1 Backend

Do generowania treści zostało wykorzystane api ChatGPT. Została utworzona metoda POST, która pozwala na wysłanie tekstu do endpointu. Na podstawie otrzymanej treści ChatGPT generuje treść i odsyła ją do aplikacji. Klucz api trzymany jest w zmiennych środowiskowych.

```
@router.post( path: "/chat", status_code=status.HTTP_201_CREATED)
async def create_flash_card(body: RequestBody):
    chat_api_key = os.environ.get('KEY_GPT')
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {chat_api_key}',
    }

    url = 'https://api.openai.com/v1/chat/completions'
    try:
        response = requests.post(url, json=body.dict(), headers=headers)
        response.raise_for_status()

        response_data = response.json()
        chat_answer = response_data['choices'][0]['message']['content']
        return chat_answer
    except requests.RequestException as e:
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                            detail=f"Failed to connect with chatGPT: {e}")
```

Rysunek 7.57: Endpoint służący do komunikacji z ChatGPT.

7.3.13.2 Aplikacja mobilna

Aby wygenerować definicje fiszki z ChatGPT, użytkownik musi mieć wypełniony tytuł fiszki. Następnie funkcja wysyła na wstakany endpoint tytuł fiszki, API kontaktuje się z ChatGPT w celu wygenerowania definicji. Bo wygenerowaniu definicji, zostaje ona zwrócona na front, gdzie wartość definicji zostaje nadpisana. Następnie po zatwierdzeniu fiszki przez użytkownika, aplikacja wysyła dane na api gdzie zostaje utworzona fiszka dla edytowanej talii.

```
Show usages  ✎ klimono *
const handleGenerate = async () : Promise<void> => {
  if (sideA.length >= 0) {
    alert("Please note that the response from the AI chat may not be accurate.");
    const messageToSend : string = `Please limit the response to 500 characters: ${sideA}`;
    try {
      const response = await ChatService.send_message(messageToSend);
      setSideB(response);
    } catch (error) {
      console.error("Failed to generate content with AI:", error);
      alert("Failed to generate content with AI. Please try again.");
    }
  } else {
    alert("Flashcard front side field must not be empty.");
  }
}
```

Rysunek 7.58: Funkcja obsługująca generowanie treści za pomocą ChatGPT.

7.3.13.3 Aplikacja webowa

Został utworzony service, który pozwala na połączenie z bakkendowym endpointem do komunikacji z api ChatGPT. Aby treść fiszki została wygenerowana użytkownik musi na przedniej stronie fiszki napisać wiadomość i kliknąć przycisk generate. Po kliknięciu treści z przedniej strony fiszki zostaje przekazana do funkcji, która wysyła pobraną treść do backendowego endpointu, następnie ChatGPT na podstawie otrzymanego kontentu generuje odpowiedź. Po otrzymaniu odpowiedzi na stronie internetowej wyskakuje okienko z informacją, że treści generowane przez ChatGPT mogą być nieprawdziwe, poniżej znajduje się wygenerowana wiadomość, użytkownik może zaakceptować otrzymana treść lub nie. W przypadku jeżeli zaakceptuje wiadomość tylnej strony fiszki zostanie zapełniona wygenerowaną wiadomością w przeciwnym wypadku fiszka zostaje pusta.

```
const handleGenerateText = async (id: number) : Promise<void> => {
    const frontSideText = texts[`front-${id}`] || '';
    if (frontSideText.length > 2) {
        setIsChatGenerating({ value: true });
        try {
            let chat_answer = await ChatService.sent_message(frontSideText)
            const maxLength: 511 = 511;
            const sliced_message = chat_answer.slice(0, maxLength);
            setboxContent(sliced_message);
            setboxOpen({ value: true });
        } catch (error) {
            console.error('Failed to generate text from chat:', error);
            setAlertMessage({ value: 'Failed to generate text from chat.' });
            setShowAlert({ value: true });
        } finally {
            setIsChatGenerating({ value: false });
        }
    } else {
        setAlertMessage({ value: 'Front side cannot be empty.' });
        setShowAlert({ value: true });
    }
};
```

Rysunek 7.59: Funkcja, która pobiera treść fiszki z przedniej strony karty i przekazuje wiadomość do service odpowiedzialnego za komunikację z backendem.

```
public async sent_message(message: string) : Promise<any> {
    const url : string = `${BASE_API}/chat_gpt/chat`;
    const body : string = JSON.stringify({ value: {
        "model": "gpt-3.5-turbo",
        "messages": [
            {"role": "user", "content": message}
        ]
    });
    try {
        console.log('sending message to chat');
        const response : Response = await fetch(url, { init: {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: body
        });
        if (!response.ok) {
            throw new Error(`Failed to send message: ${response.statusText}`);
        }
        const data = await response.json();
        return data;
    } catch (error) {
        console.error("Failed connect with ChatGPT", error);
        throw error;
    }
}
```

Rysunek 7.60: Funkcja służąca do komunikacji backendowym endpointem do wysyłania wiadomości do ChatGPT.

Rozdział 8

Testy

Testowanie aplikacji odbywało się po dodaniu nowej funkcjonalności, co pozwoliło zmniejszyć liczbę sytuacji, w których aplikacja nie działała zgodnie z oczekiwaniami. Testy integracyjne były przeprowadzane w celu sprawdzenia poprawności komunikacji endpointów z bazą danych. Opracowane zostały scenariusze testowe dla testów akceptacyjnych, które umożliwiły weryfikację działania kluczowych funkcji systemu. Po uzyskaniu pozytywnych wyników testów, zmiany mogły zostać dodane do repozytorium.

8.1 Testy integracyjne

Testy integracyjne mają na celu sprawdzenie, czy interfejsy pomiędzy komponentami współdziałają ze sobą poprawnie [8]. Do przeprowadzenia testów została użyta biblioteka "pytest", która pozwala na szybkie i proste uruchomienie testów [9]. Do wykonywania zapytań HTTP została wykorzystana biblioteka "requests". Testy miały na celu sprawdzenie, czy status zwrócony przez wykonane żądanie pokrywa się z oczekiwany rezultatem.

Przetestowane zostały poniższe funkcjonalności systemu:

- Utworzenie fiszki;
- Zwrócenie danych fiszki na podstawie identyfikatora;
- Aktualizacja danych fiszki;
- Usunięcie fiszki;
- Otrzymanie odpowiedzi od ChatGPT;
- Tworzenie talii;
- Zwrócenie danych talii na podstawie identyfikatora;
- Zwrócenie danych wszystkich talii należących do użytkownika;
- Aktualizacja danych talii;
- Usunięcie talii;

- Zwrócenie rankingu talii;
- Utworzenie konta użytkownika;
- Logowanie do systemu;
- Zwrócenie danych użytkownika.

```
def test_delete_deck():
    body = {
        "user_id": f"{user['id']}",
        "title": "string",
        "deck_category": "string"
    }
    headers = {
        "Authorization": f"Bearer {token}"
    }

    deck = requests.post(ENDPOINT + "decks/create_deck", json=body , headers=headers)
    deck_id = deck.json()['id']
    response = requests.delete(ENDPOINT + f"decks/delete_deck/{deck_id}" , json=body , headers=headers)
    assert response.status_code == 200
```

Rysunek 8.1: Test sprawdzający działanie endpointu do usuwania talii użytkownika.

8.2 Testy akceptacyjne

Testy akceptacyjne pozwalają ocenić gotowość systemu do wdrożenia. Głównym ich założeniem jest sprawdzenie, czy system nadaje się do użytkowania [10]. Do przeprowadzenia testów zostały sporządzone scenariusze testowe. Oliwier przechodził przez scenariusze testowe w celu sprawdzenia czy aplikacja działa zgodnie z oczekiwaniemi.

Opis przypadku:	Rejestracja użytkownika na stronie internetowej
Cel przypadku:	Użytkownik rejestruje się aby utworzyć konto, potrzebne do logowania.
Testowane wymagania:	3.2.1 Tworzenie konta użytkownika
Warunki początkowe:	Strona z formularzem, która pozwala wprowadzić dane użytkownika w celu zarejestrowania go.
Warunki końcowe:	Utworzony konto użytkownika
Dane wejściowe:	E-mail użytkownika, hasło i nickname
Oczekiwany wynik:	Użytkownik po zarejestrowaniu zostanie dodany do bazy danych a utworzone konto pozwoli mu na logowanie do strony.
Kroki procedury:	<ol style="list-style-type: none"> Uruchomienie przeglądarki - Pozytywny Wejście na stronę rejestracji - Pozytywny Wypełnienie formularza rejestracji - Pozytywny Kliknięcie w przycisk rejestracji - Pozytywny Nowe konto dodane do bazy danych - Pozytywny
Wynik:	Pozytywny

Tabela 8.1: Test akceptacyjny procesu rejestracji nowego użytkownika.

Opis przypadku:	Logowanie na stronę
Cel przypadku:	Sprawdzenie czy użytkownik posiadający konto ma dostęp do strony po poprawnym wypełnieniu formularza logowania.
Testowane wymagania:	3.2.2 Logowanie do systemu
Warunki początkowe:	Posiadane konto użytkownika
Warunki końcowe:	dostęp do strony domowej
Dane wejściowe:	e-mail, hasło
Oczekiwany wynik:	Użytkownik po poprawnym zalogowaniu przechodzi do strony domowej.
Kroki procedury:	<ol style="list-style-type: none"> Uruchomienie przeglądarki - Pozytywny Wejście na stronę logowania - Pozytywny Wypełnienie formularza logowania - Pozytywny Użytkownik zostaje przekierowany do strony domowej - Pozytywny
Wynik:	Pozytywny

Tabela 8.2: Test akceptacyjny procesu logowania użytkownika.

Opis przypadku:	Wylogowanie z systemu
Cel przypadku:	Sprawdzenie czy po wylogowaniu użytkownik, nie ma dostępu do strony domowej bez ponownego zalogowania
Testowane wymagania:	3.2.3 Wylogowanie z systemu
Warunki początkowe:	Posiadane konto użytkownika
Warunki końcowe:	Użytkownik po kliknięciu przycisku wylogowania zostaje przekierowany do strony logowania i nie ma dostępu do strony domowej bez ponownego zalogowania
Dane wejściowe:	e-mail, hasło
Oczekiwany wynik:	Brak dostępu do strony domowej
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w logout na nawigacji - Pozytywny 6. Przekierowanie do strony logowania - Pozytywny
Wynik:	Pozytywny

Tabela 8.3: Test akceptacyjny procesu wylogowania użytkownika.

Opis przypadku:	Edycja adresu e-mail
Cel przypadku:	Sprawdzenie czy użytkownik może zmienić e-mail
Testowane wymagania:	3.2.4 Edycja danych użytkownika
Warunki początkowe:	Posiadane konto użytkownika
Warunki końcowe:	Dane użytkownika zostaną zaktualizowane
Dane wejściowe:	e-mail, hasło
Oczekiwany wynik:	Użytkownik po zmianie e-mail zostaje wylogowany, a e-mail zostanie zaktualizowany.
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w profil użytkownika - Pozytywny 6. Kliknięcie w przycisk zmiany e-mail - Pozytywny 7. Wypełnienie formularza i kliknięcie zapisu - Pozytywny 8. Przekierowanie do strony logowania - Pozytywny 9. Zalogowanie się przy użyciu nowych danych - Pozytywny
Wynik:	Pozytywny

Tabela 8.4: Test zmiany adresu e-mail użytkownika.

Opis przypadku:	Usunięcie konta użytkownika
Cel przypadku:	Sprawdzenie czy użytkownik może usunąć swoje konto.
Testowane wymagania:	3.2.5 Usunięcie konta użytkownika
Warunki początkowe:	Posiadane konto użytkownika
Warunki końcowe:	usunięte konto użytkownika, brak możliwości zalogowania
Dane wejściowe:	e-mail, hasło
Oczekiwany wynik:	Użytkownik po usunięciu konta zostaje wylogowany z aplikacji.
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w profil użytkownika - Pozytywny 6. Kliknięcie w usunięcie konta - Pozytywny 7. Wypełnienie formularza i kliknięcie zapisu - Pozytywny 8. Przekierowanie do strony logowania - Pozytywny 9. Użytkownik nie jest w stanie zalogować się do strony - Pozytywny
Wynik:	Pozytywny

Tabela 8.5: Test procesu usuwania konta użytkownika.

Opis przypadku:	Tworzenie talii fiszek
Cel przypadku:	Sprawdzenie czy użytkownik może utworzyć nową talię fiszek.
Testowane wymagania:	3.2.6 Tworzenie talii fiszek
Warunki początkowe:	Posiadane konto użytkownika
Warunki końcowe:	Utworzona talia fiszek
Dane wejściowe:	nazwa i kategoria talii, treść fiszki
Oczekiwany wynik:	Po utworzeniu talii jest ona dostępna na podstronie my decks.
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w create decks - Pozytywny 6. Użytkownik wypełnia pole z nazwą i kategorią talii - Pozytywny 7. Wypełnienie przedniej i tylnej strony fiszki - Pozytywny 8. Kliknięcie w przycisk create deck - Pozytywny 9. Przejście do strony domowej - Pozytywny 10. Przejście do strony my decks w celu sprawdzenia czy została dodana utworzona talia - Pozytywny
Wynik:	Pozytywny

Tabela 8.6: Test procesu tworzenia nowej talii fiszek.

Opis przypadku:	Usunięcie talii fiszek
Cel przypadku:	Sprawdzenie czy użytkownik, może usunąć talie, która nie jest mu już potrzebna
Testowane wymagania:	3.2.7 Usunięcie talii fiszek
Warunki początkowe:	Posiadane konto użytkownika, utworzona talia
Warunki końcowe:	Talia przestaje istnieć
Dane wejściowe:	Brak
Oczekiwany wynik:	Po usunięciu talia znika z podstrony my decks
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w my decks - Pozytywny 6. Kliknięcie w talie - Pozytywny 7. Kliknięcie w opcję - Pozytywny 8. Kliknięcie w delete deck i potwierdzenie usunięcia decku - Pozytywny 9. Przekierowanie do strony my decks. Brak widocznej talii na stronie my decks - Pozytywny
Wynik:	Pozytywny

Tabela 8.7: Test procesu usuwania talii fiszek.

Opis przypadku:	Edycja talii fiszek
Cel przypadku:	Użytkownik musi mieć możliwość zmiany nazwy talii lub kategorii, jeżeli popełni jakiś błąd w tytule lub kategorii.
Testowane wymagania:	3.2.8 Edycja talii fiszek
Warunki początkowe:	Utworzona talia fiszek
Warunki końcowe:	Talia fiszek zaktualizowana o nowe wartości.
Dane wejściowe:	Brak
Oczekiwany wynik:	Zostaje zmieniona nazwa talii i kategoria.
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w my decks - Pozytywny 6. Kliknięcie w talie - Pozytywny 7. Kliknięcie w opcje - Pozytywny 8. Kliknięcie w zmianę kategorii lub nazwy - Pozytywny 9. Podanie w formularzu nowych wartości dla kategorii i nazwy - Pozytywny 10. Kliknięcie w przycisk save - Pozytywny 11. Informacje talii zostaną zaktualizowane o nowe wartości - Pozytywny
Wynik:	Pozytywny

Tabela 8.8: Test procesu edycji talii fiszek.

Opis przypadku:	Tryb uczenia się z talii fiszek służy do podziału fiszek na te, które użytkownik zna i te nie zapamiętane
Cel przypadku:	Sprawdzenie czy tryb uczenia poprawnie rozdziela fiszki należące do talii
Testowane wymagania:	3.2.9 Tryb uczenia się z talii fiszek
Warunki początkowe:	Utworzona talia fiszek, posiadane konto użytkownika
Warunki końcowe:	Fiszki podzielone na zapamiętane i nie zapamiętane
Dane wejściowe:	Brak
Oczekiwany wynik:	Fiszki w talii zostają podzielone przez użytkownika na te, które zna i te których nie pamięta
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w my decks - Pozytywny 6. Kliknięcie w talie - Pozytywny 7. Kliknięcie w tryb uczenia się - Pozytywny 8. Użytkownik dzieli talię na fiszki, na zapamiętane i nie zapamiętane - Pozytywny 9. Fiszki w talii zostają podzielone na zapamiętane i nie zapamiętane - Pozytywny
Wynik:	Pozytywny

Tabela 8.9: Test trybu uczenia się z talii fiszek.

Opis przypadku:	Po uruchomieniu trybu sterowania głosowego, aplikacja powinna reagować na komendy głosowe
Cel przypadku:	Sprawdzenie czy aplikacja reaguje na komendy głosowe
Testowane wymagania:	3.2.10 Sterowanie talią przy użyciu mowy
Warunki początkowe:	Utworzona talia fiszek, posiadane konto użytkownika
Warunki końcowe:	Brak
Dane wejściowe:	Brak
Oczekiwany wynik:	Aplikacja wykonuje akcje zgodnie z wypowiedzianymi przez użytkownika komendami
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w my decks - Pozytywny 6. Kliknięcie w talie - Pozytywny 7. Kliknięcie w voice control - Pozytywny 8. Kliknięcie w start listening - Pozytywny 9. Pojawia się popup box z instrukcją, zamknięcie instrukcji powoduje rozpoczęcie nashuchiwanego przez mikrofon - Pozytywny 10. Komenda "next flashcard" przenosi do następnej fiszki - Pozytywny 11. Komenda "read text" uruchamia syntezator odczytujący treść fiszki - Pozytywny 12. Komenda "rotate" obraca fiszkę - Pozytywny 13. Komenda "previous flashcard" przenosi do poprzedniej fiszki - Pozytywny 14. Komenda "stop control" wyłącza tryb głosowy - Pozytywny
Wynik:	Pozytywny

Tabela 8.10: Test sterowania talią przy użyciu mowy.

Opis przypadku:	Import talii pozwala na dodanie talii znalezionej w rankingu do talii zawartych w public decks
Cel przypadku:	Sprawdzenie czy talia zimportowana z rankingu dodaje się do public decks
Testowane wymagania:	3.2.11 Import talii innych użytkowników
Warunki początkowe:	Utworzono konto użytkownika
Warunki końcowe:	Talia dodana do public decks
Dane wejściowe:	Brak
Oczekiwany wynik:	Po kliknięciu przycisku importuj talia fiszek zostaje dodana do public decks
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w ranking na nawigacji - Pozytywny 6. Kliknięcie w decks ranking - Pozytywny 7. Kliknięcie w dowolną talię - Pozytywny 8. Kliknięcie w przycisk import deck - Pozytywny 9. Przeniesienie użytkownika do public decks - Pozytywny 10. Zimportowana talia dodana do public decks - Pozytywny
Wynik:	Pozytywny

Tabela 8.11: Test procesu importowania talii fiszek z rankingu.

Opis przypadku:	Użytkownik może wykorzystać przycisk generowania treści w celu wygenerowania przez ChatGPT zawartości fiszki.
Cel przypadku:	Sprawdzenie czy treść fiszki zostaje wygenerowana.
Testowane wymagania:	3.2.16 Generowanie treści fiszki na podstawie słów kluczowych
Warunki początkowe:	Utworzono konta użytkownika
Warunki końcowe:	Wygenerowana treść fiszki
Dane wejściowe:	Wpisanie treści w przednią część fiszki
Oczekiwany wynik:	Po uzupełnieniu przedniej strony fiszki i kliknięciu generate zostanie wyświetlona treść wygenerowana przez ChatGPT.
Kroki procedury:	1. Uruchomienie przeglądarki - Pozytywny 2. Wejście na stronę logowania - Pozytywny 3. Wypełnienie formularza logowania - Pozytywny 4. Użytkownik zostaje przekierowany do strony domowej - Pozytywny 5. Kliknięcie w create decks - Pozytywny 6. Uzupełnienie przedniej strony fiszki treścią - Pozytywny 7. Kliknięcie w generate - Pozytywny 8. Pojawienie się okienka z wygenerowaną treścią - Pozytywny 9. Kliknięcie accept - Pozytywny 10. Tylna część fiszki zostaje uzupełniona wygenerowaną treścią - Pozytywny
Wynik:	Pozytywny

Tabela 8.12: Test generowania treści fiszki przy użyciu funkcji ChatGPT.

Rozdział 9

Prezentacja projektu

Rozdział zawiera zdjęcie i opisy przedstawiające działanie aplikacji mobilnej i strony internetowej.

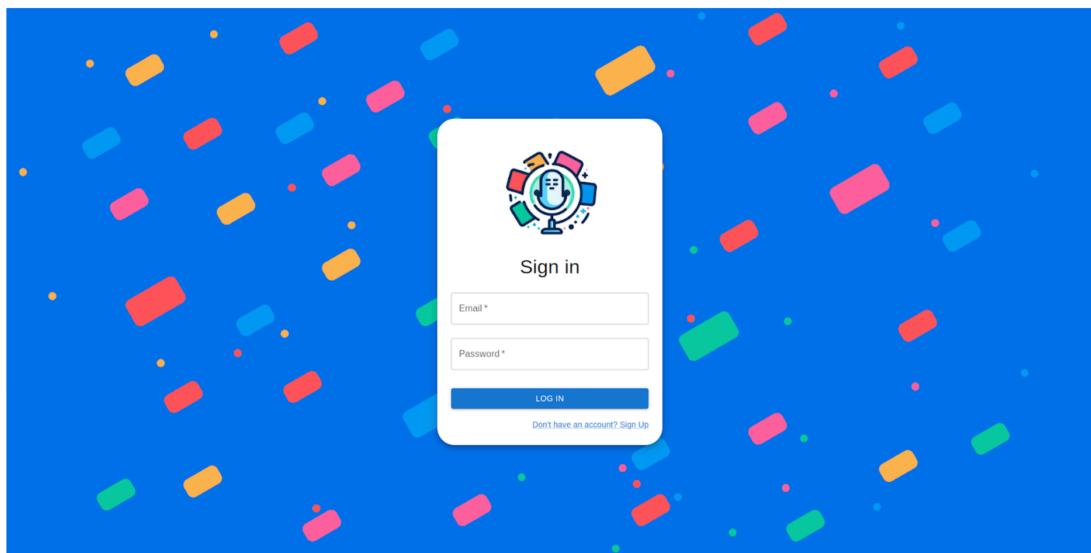
9.1 Strona internetowa

Rozdział przedstawia widoki i działanie strony internetowej.

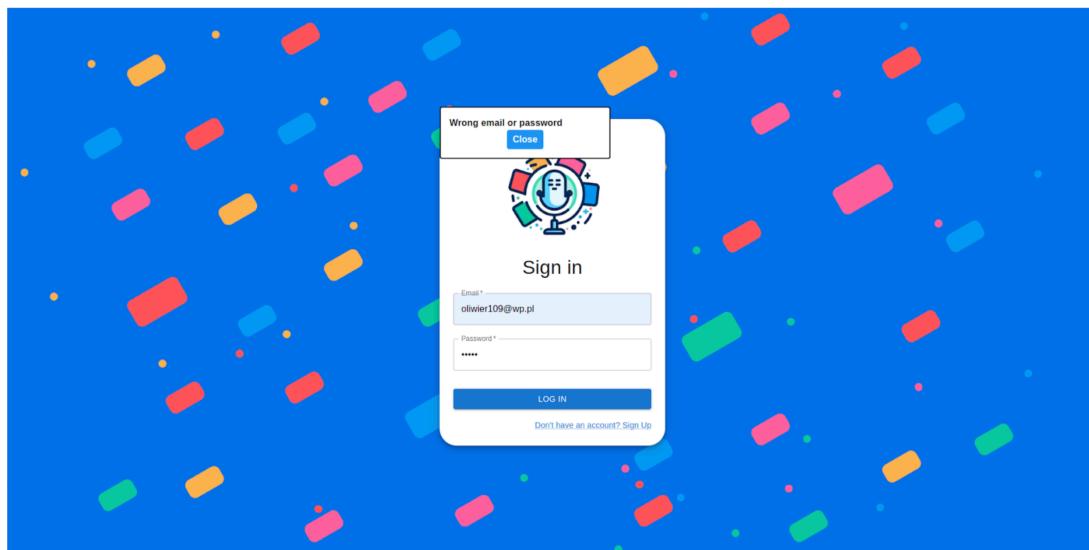
9.1.1 Logowanie

Po uruchomieniu strony ukazuje się widok zawierający formularz logowania do strony internetowej.

Po pomyślnym zalogowaniu użytkownik zostaje przekierowany do strony domowej.



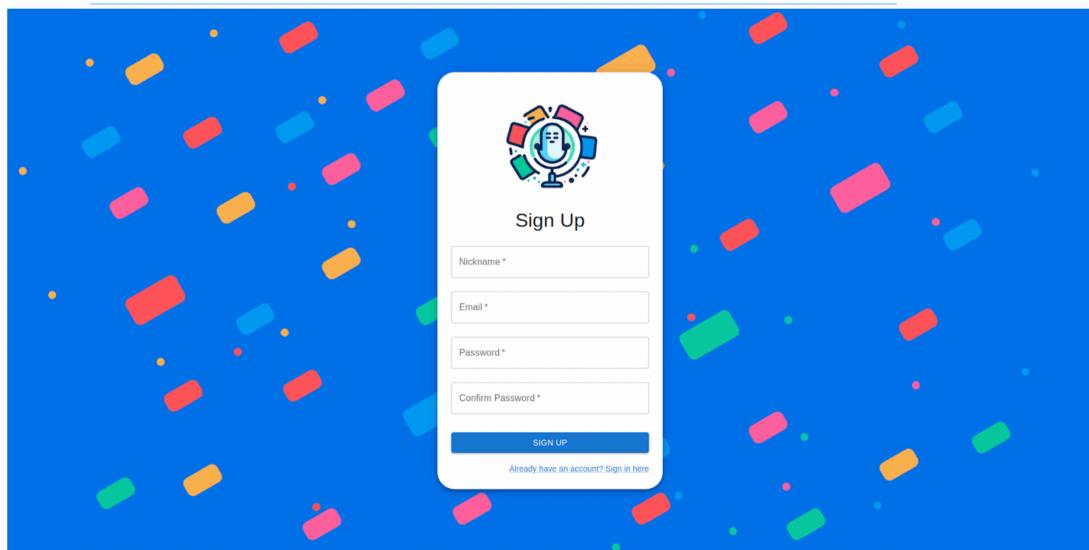
Rysunek 9.1: Widok logowania.



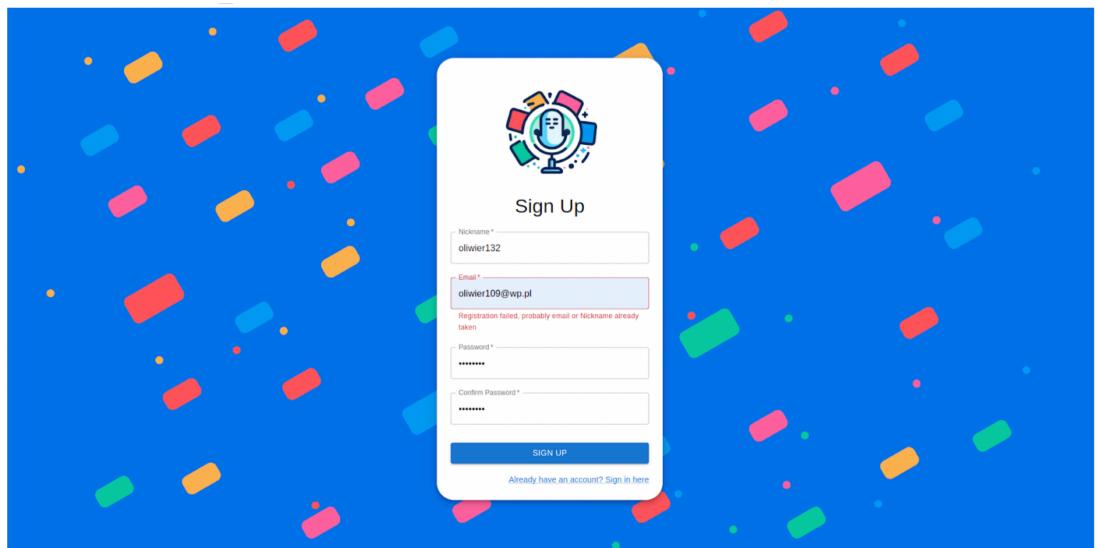
Rysunek 9.2: Widok logowania po wpisaniu nieprawidłowych danych logowania.

9.1.2 Rejestracja

Rejestracja pozwala na utworzenie konta potrzebnego do logowania.



Rysunek 9.3: Formularz rejestracji.

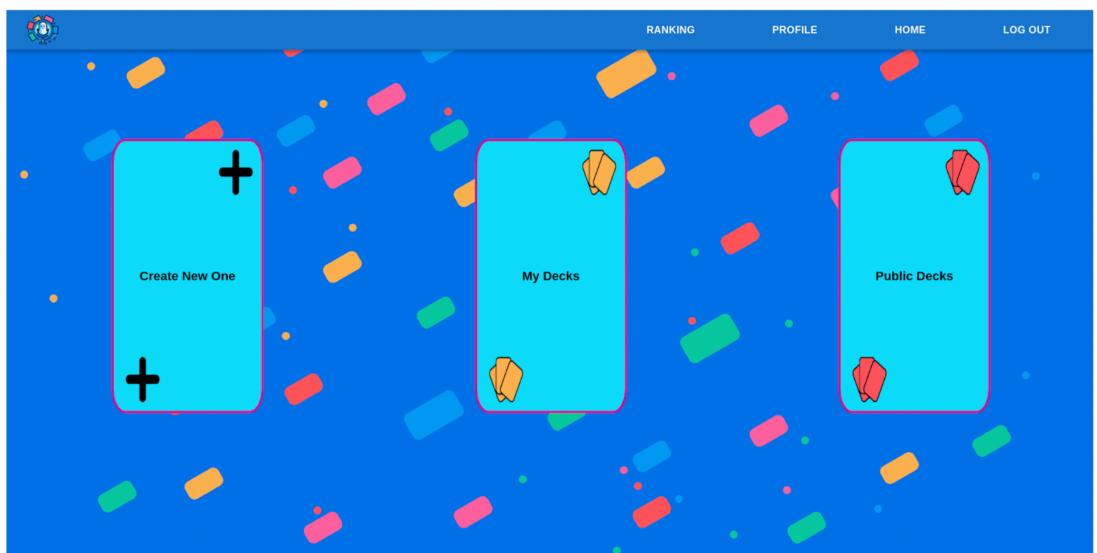


Rysunek 9.4: Błąd po nieudanej rejestracji konta.

9.1.3 Strona domowa

Po zalogowaniu użytkownik zostaje przekierowany do strony domowej. Na stronie domowej użytkownika możliwość wylogowania się lub może przejść do któregoś z podanych widoków:

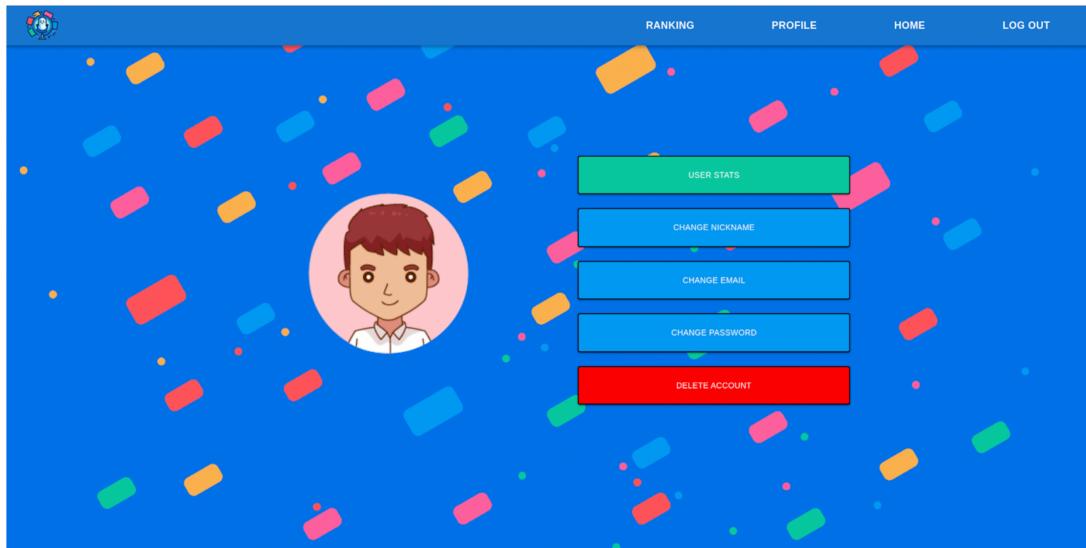
- "Create Decks";
- "My Decks";
- "Public Decks";
- "Profile";
- "Ranking".



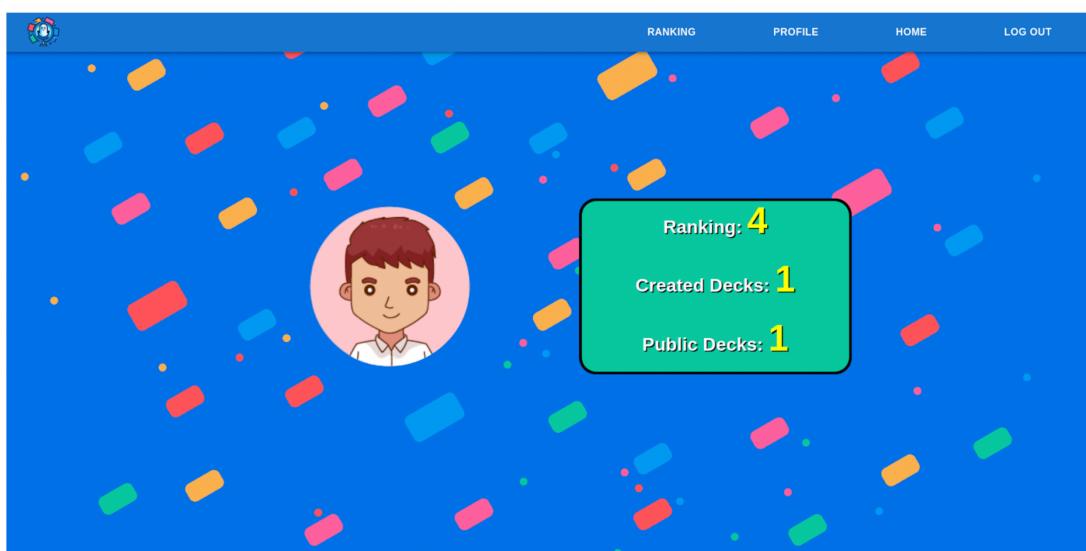
Rysunek 9.5: Strona domowa.

9.1.4 Profil użytkownika

Strona przedstawia profil użytkownika. Użytkownik ma możliwość zmiany swoich danych, usunięcia konta oraz przejścia do widoku ze swoimi statystykami.



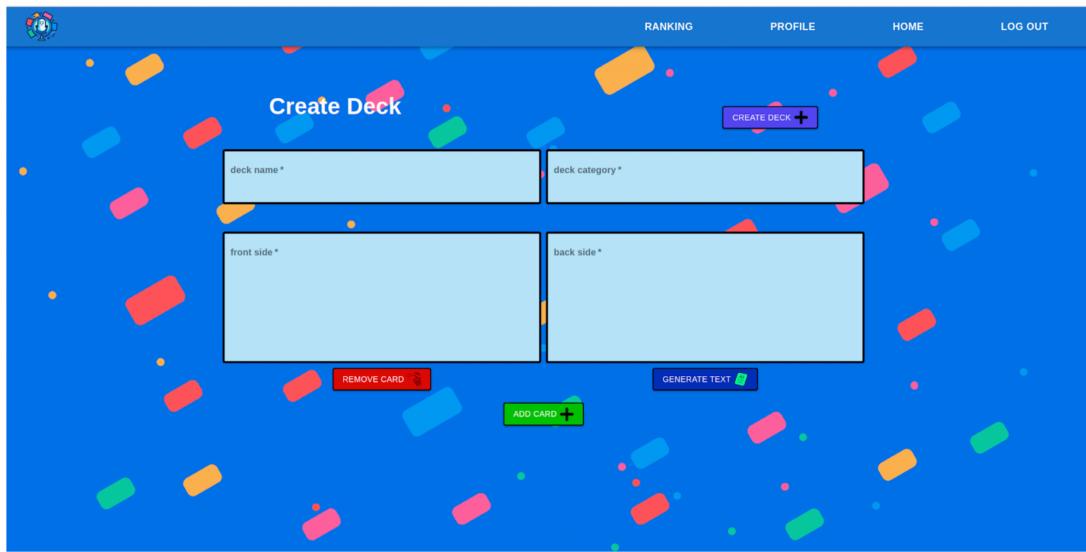
Rysunek 9.6: Profil użytkownika.



Rysunek 9.7: Statystyki użytkownika.

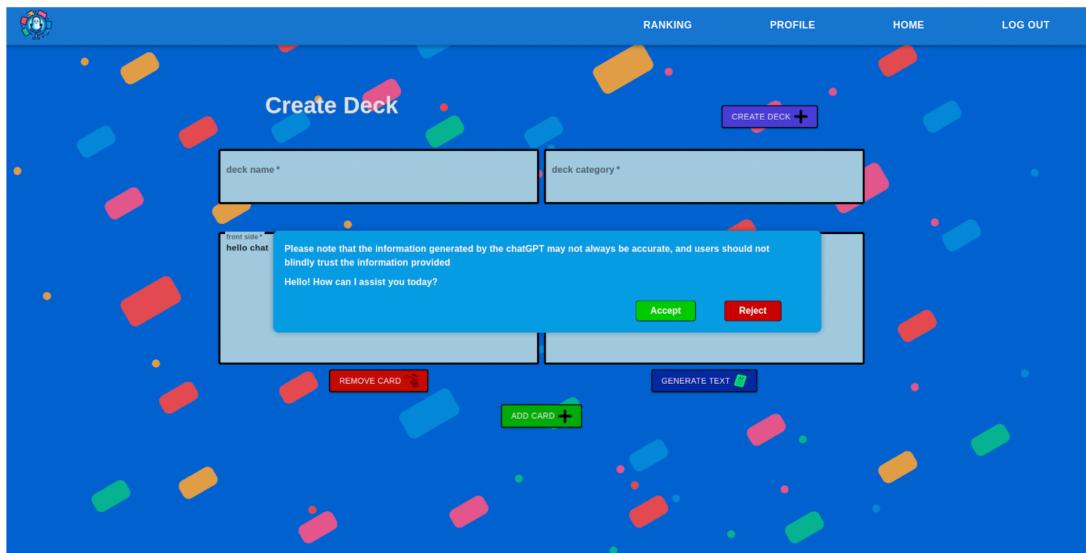
9.1.5 "Create deck"

Strona umożliwia tworzenie talii fiszek. Przycisk "Generate" pozwala na wygenerowanie treści fiszki używając ChatGPT. W celu utworzenia talii fiszek, użytkownik musi wypełnić nazwę talii, kategorię i mieć przynajmniej jedna wypełnione pole dla fiszki. Przycisk "Add card" dodaje następne pola do utworzenia fiszki. Użytkownik po wypełnieniu formularza musi kliknąć "Create deck" w celu utworzenia talii.



Rysunek 9.8: Widok tworzenia talii.

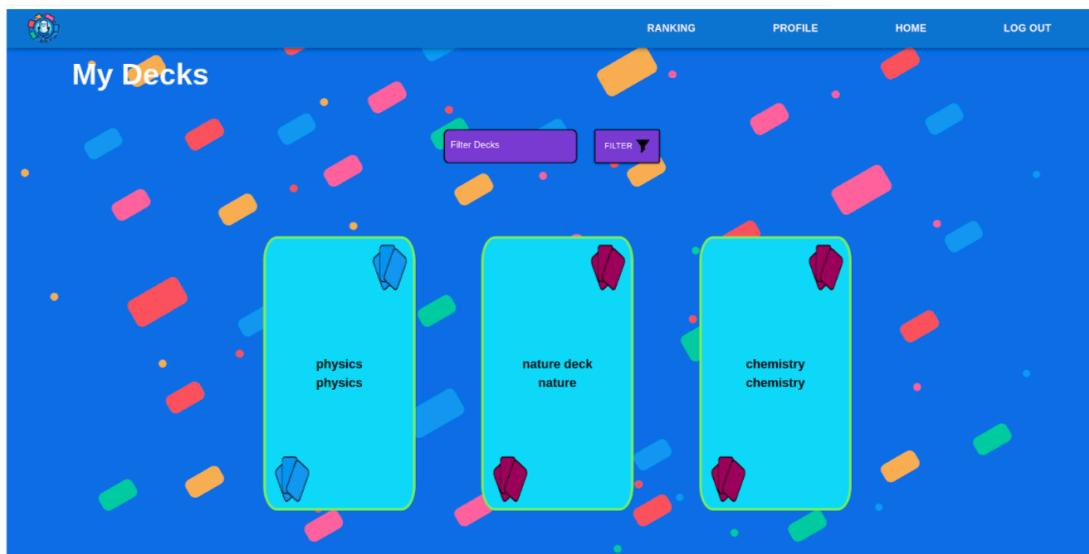
Użytkownik może wypełnić przednią stronę fiszki, następnie kliknięcie przycisku "Generate" pozwoli na wygenerowanie przez ChatGPT tekstu na podstawie zawartości przedniej strony fiszki. Po wygenerowaniu treść pojawi się w okienku i użytkownik ma możliwość zaakceptowania treści lub jej odrzucenia. W przypadku zaakceptowania tylna strona fiszki zostanie wypełniona wygenerowaną treścią.



Rysunek 9.9: Generowanie treści przez ChatGPT.

9.1.6 "My Decks"

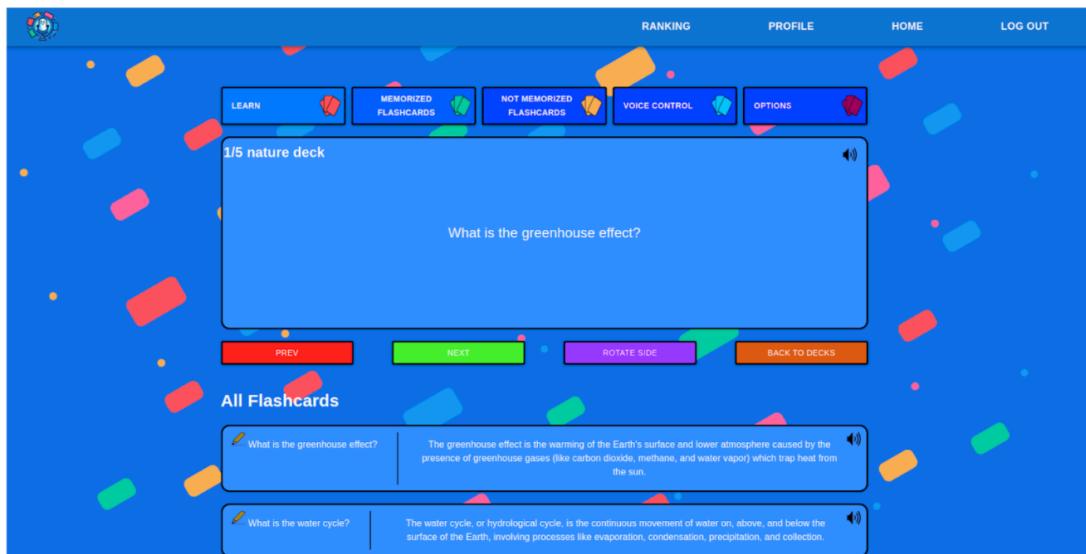
W "My decks" trzymane są wszystkie talie utworzone przez użytkownika. Filter pozwala na wyszukiwanie talii jednocześnie po kategorii i nazwie talii.



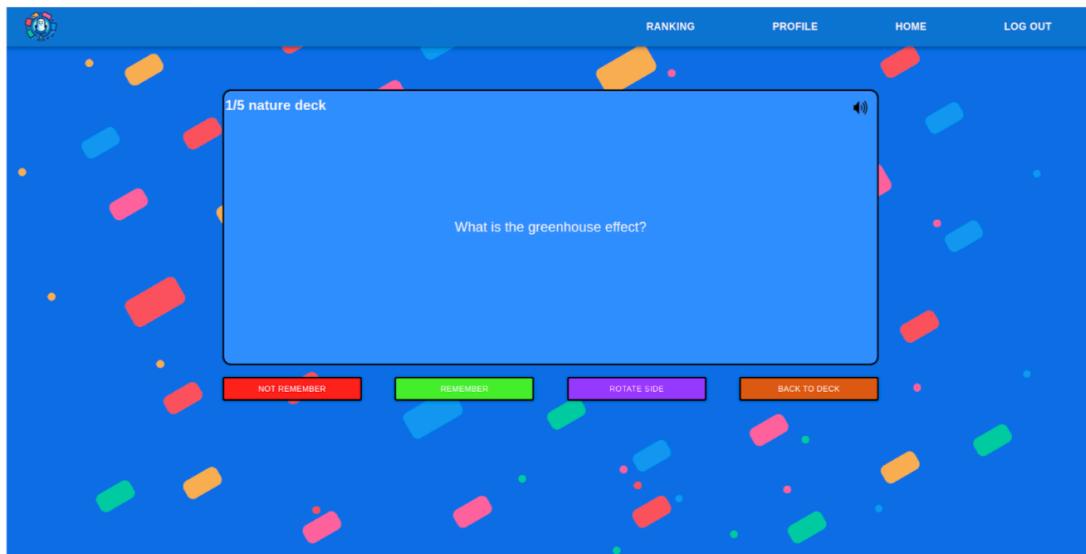
Rysunek 9.10: Widok talii utworzonych przez użytkownika.

Po przejściu ukazują się wszystkie fiszki w talii. Użytkownik po kliknięciu "Learn" uruchamia trybu uczenia, który pozwala na podzielenie talii na fiszki na zapamiętane i niezapamiętane. "Memorized" to widok w którym widać wszystkie zapamiętane fiszki, a w "Not memorized" znajdują się te niezapamiętane. W "Voice control" są dostępne wszystkie fiszki do podziału, co umożliwia użytkownikowi uruchomienie trybu sterowania na całej dostępnej talii. Opcje oferują użytkownikowi:

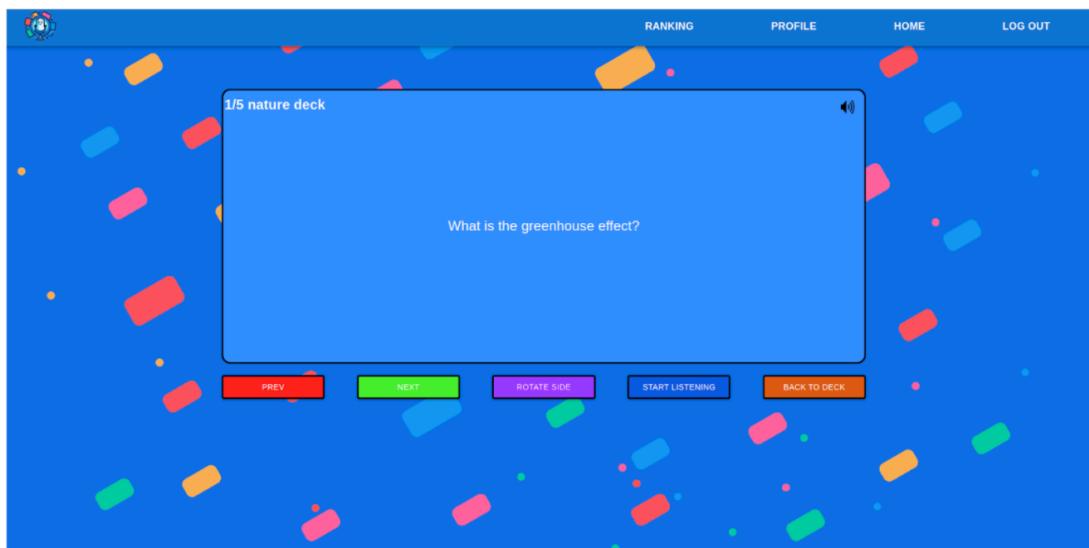
- Udostępnienie talii;
- Reset talii w celu ustawienia wszystkich fiszek jako niezapamiętane;
- Usunięcie talii;
- Dodanie nowej fiszki;
- Zmianę nazwy lub kategorii talii.



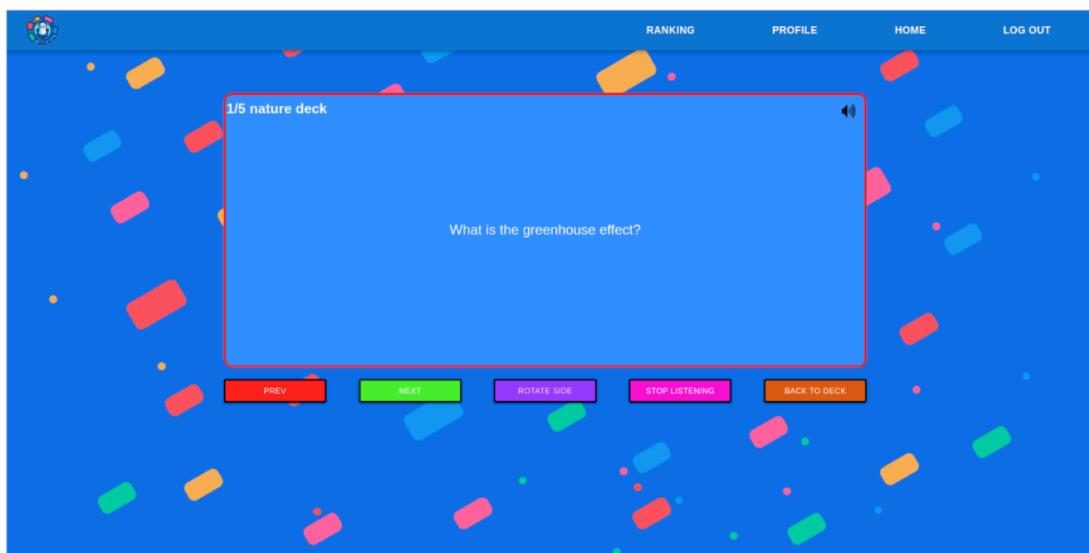
Rysunek 9.11: Widok talii po jej otworzeniu.



Rysunek 9.12: Widok podstawowego trybu nauki.



Rysunek 9.13: Widok trybu kontroli głosowej.



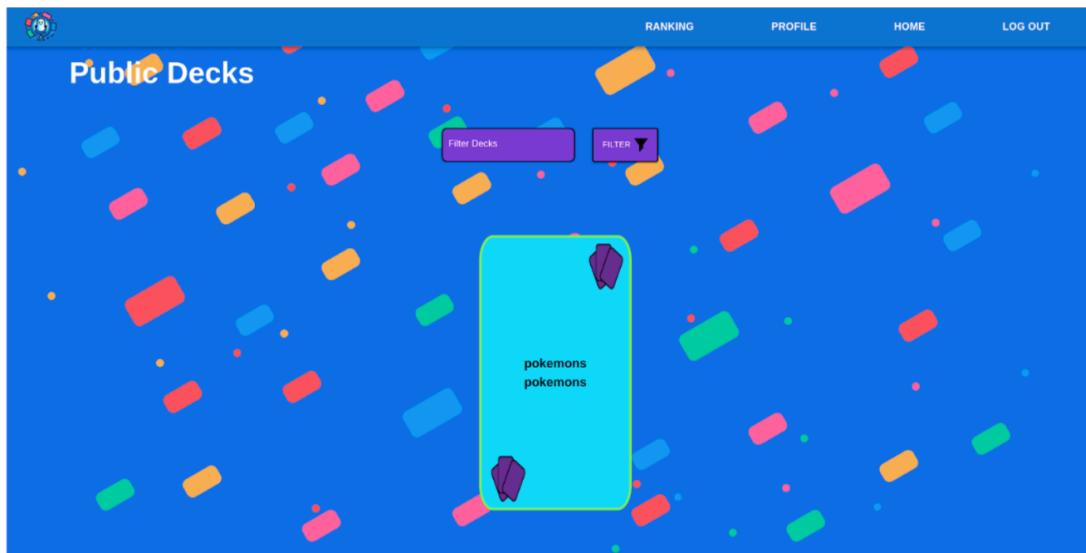
Rysunek 9.14: Widok trybu kontroli głosowej po uruchomieniu.



Rysunek 9.15: Opcje talii.

9.1.7 "Public decks"

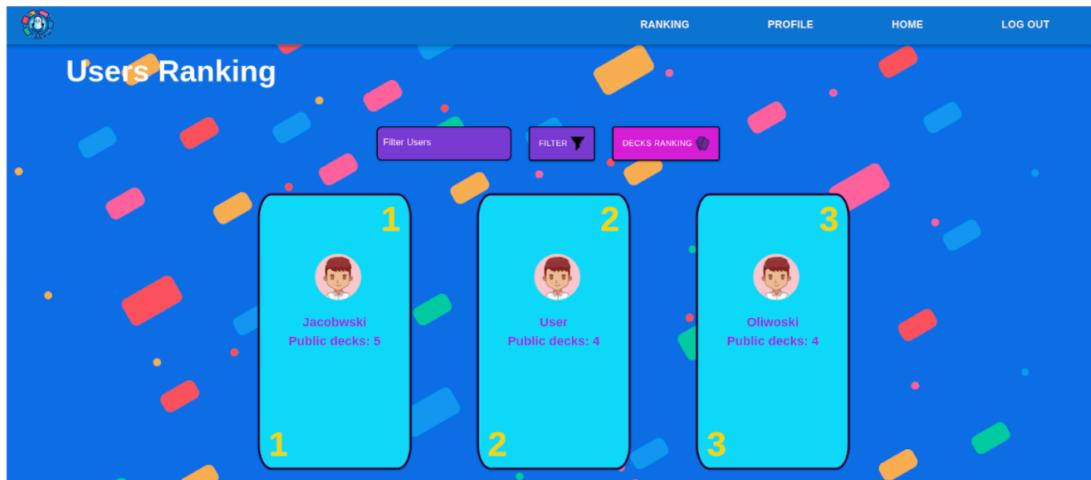
"Public decks" zawiera wszystkie talie, które zostały zainportowane z rankingu.



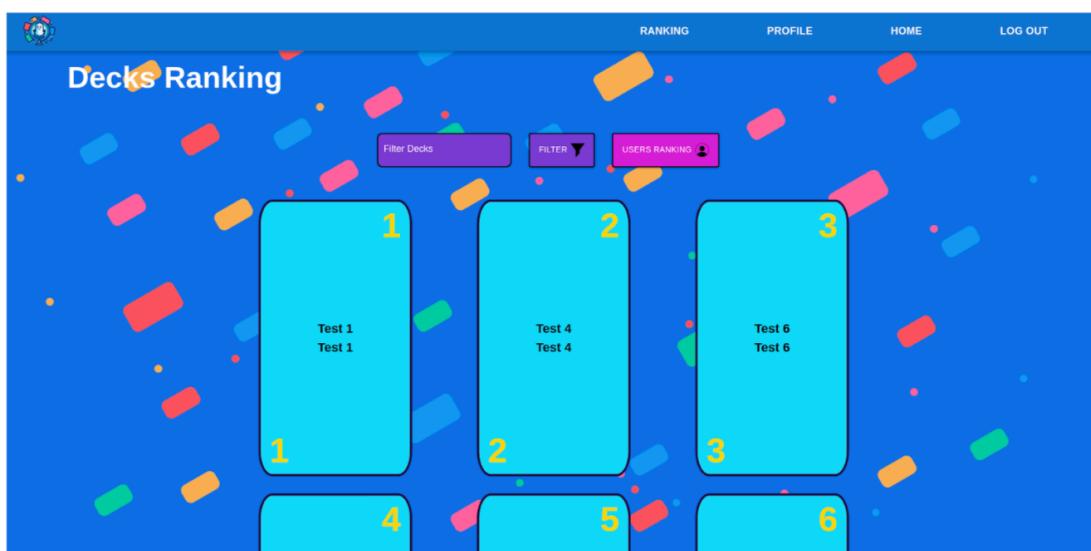
Rysunek 9.16: Widok talii w Public decks.

9.1.8 Ranking użytkowników i talii

Ranking użytkowników zawiera wszystkie konta, które udostępnili co najmniej jedną talię fiszek. Pozycja w rankingu jest zależna od sumy pobrań wszystkich talii udostępnionych przez użytkownika.

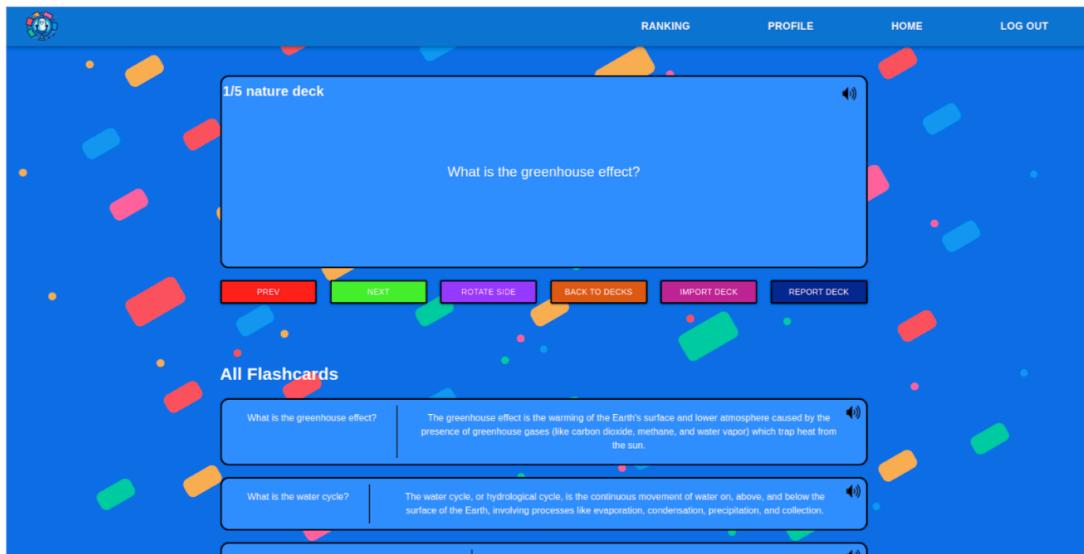


Rysunek 9.17: Ranking użytkowników.



Rysunek 9.18: Ranking publicznych talii.

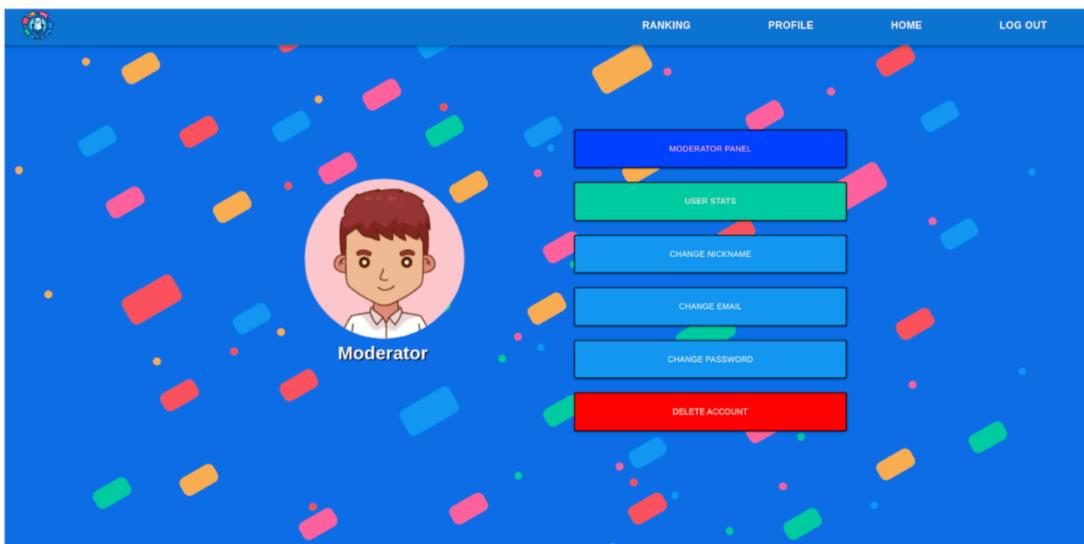
Użytkownik po otwarzeniu konkretnej talii ma możliwość obejrzenia jej zawartości. Jeżeli talia spodoba się użytkownikowi, ma on możliwość zimportowania jej, w ten sposób talia zostanie dodana do "Public decks". Można także zgłosić talię w przypadku gdy zawiera ona wulgaryzmy lub inne nieprzyzwoite wyrażenia. Po zgłoszeniu talii moderator widzi taką talię w zgłoszeniach.



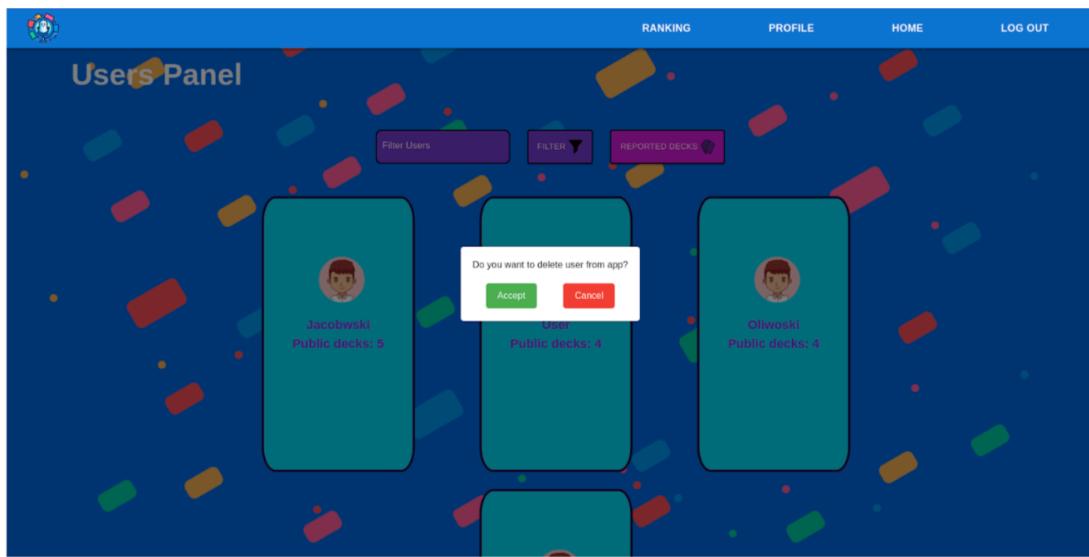
Rysunek 9.19: Widok talii wybranej z rankingu.

9.1.9 Panel moderatora

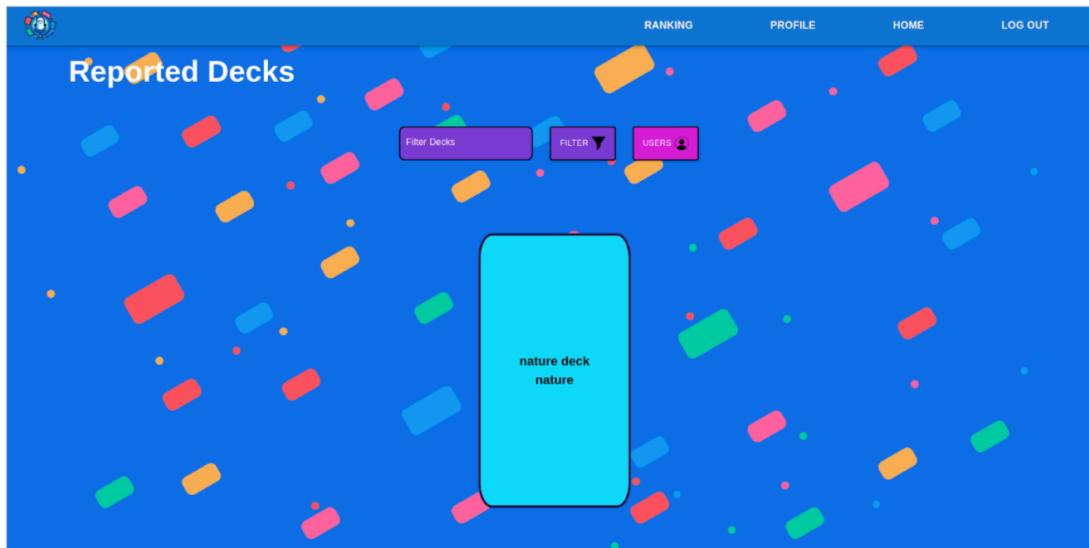
Panel moderatora pozwala na usuwanie użytkowników, usuwanie talii, edycję fiszek lub ich usunięcie. Do opcji moderatora mają dostęp tylko osoby z typem konta moderatora lub administratora, w ich przypadku w profilu pojawia się dodatkowy przycisk przekierowujący do panelu.



Rysunek 9.20: Profil użytkownika konta moderatora.

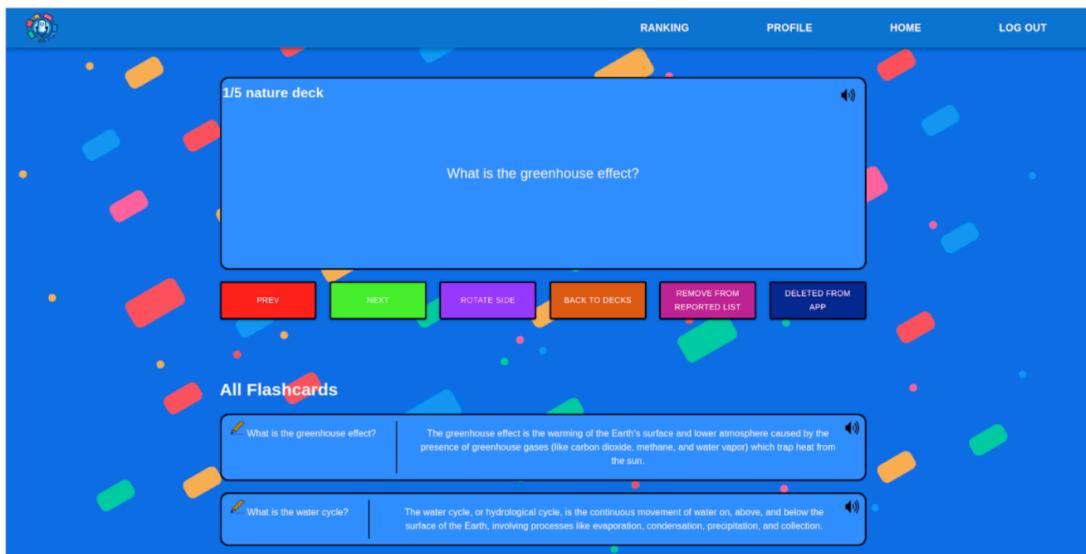


Rysunek 9.21: Usunięcie użytkownika przez moderatora.



Rysunek 9.22: Widok zgłoszonych talii.

Moderator po otwarciu talii może ją przejrzeć w celu sprawdzenia czy report talii był uzasadniony. W przypadku gdy report talii był nie potrzebny, może usunąć ją z listy zgłoszeń. Jeżeli zgłoszenie było prawidłowe może usunąć talię z aplikacji.



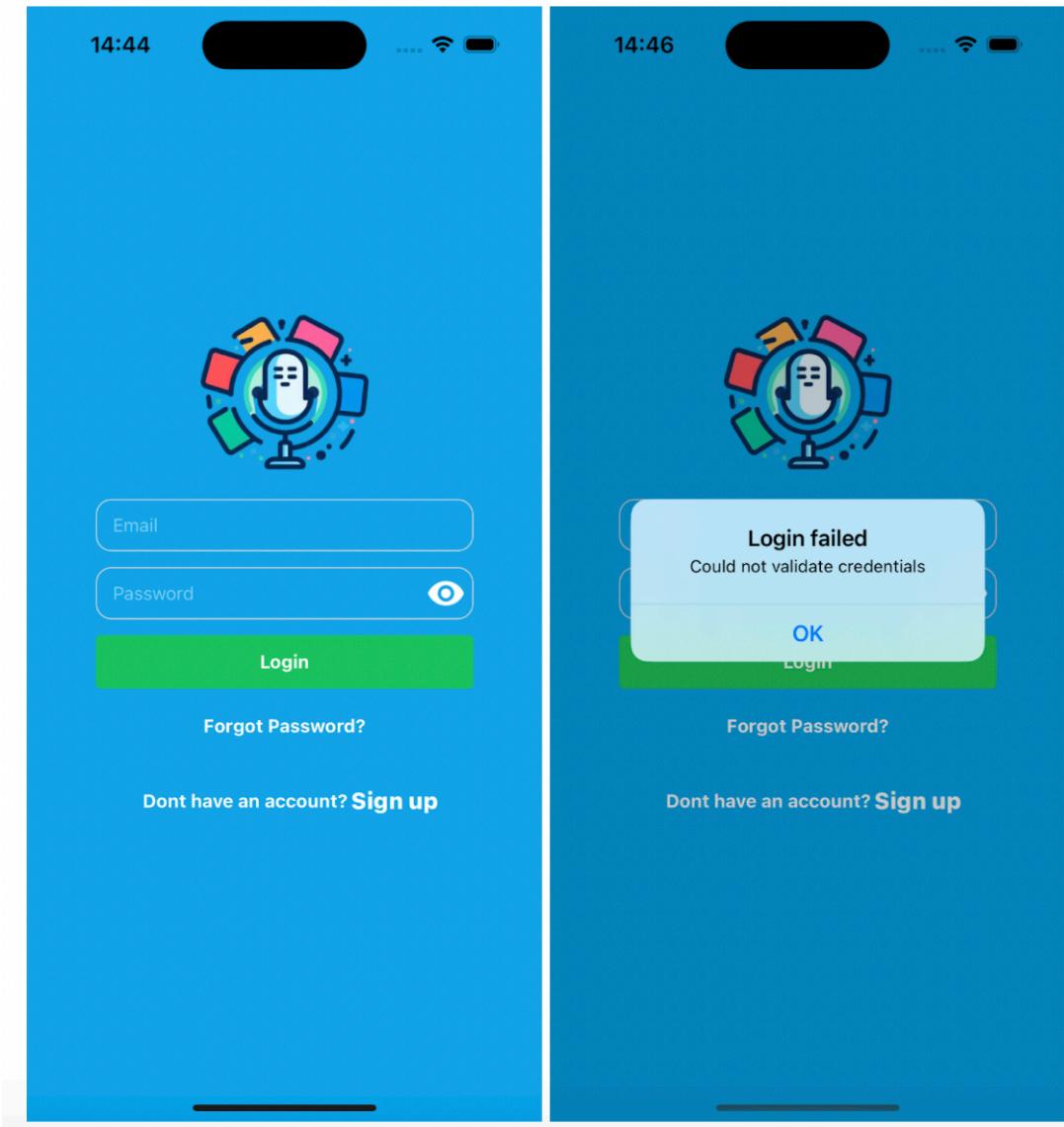
Rysunek 9.23: Widok zgłoszonej talii.

9.2 Aplikacja mobilna

Rozdział przedstawia widoki i działanie aplikacji mobilnej.

9.2.1 Logowanie

Pierwszym widokiem z perspektywy kodu i użytkownika jest ekran logowania. Aby przejść do aplikacji należy podać prawidłowe dane logowania dla zarejestrowanego wcześniej konta. Aplikacja sprawdza poprawność uzupełnionych pól.



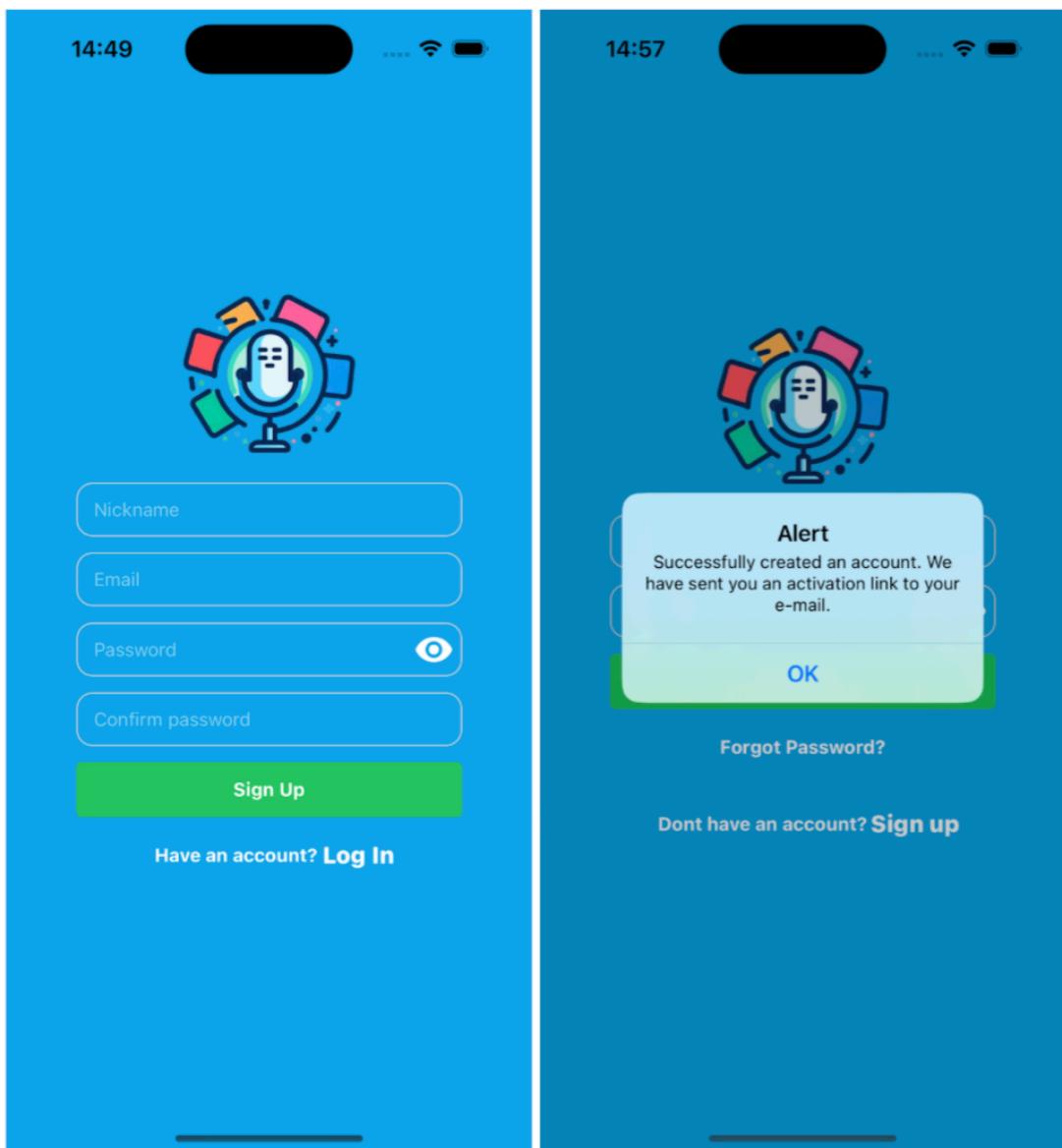
Rysunek 9.24: Ekran logowania.

9.2.2 Rejestracja

Aby utworzyć konto, którym użytkownik później zaloguje się do aplikacji, należy przejść z widoku logowania do widoku rejestracji. Ta możliwa jest jedynie wtedy, kiedy prawidłowo zostaną uzupełnione pola:

- Nazwa użytkownika;
- Adres e-mail;
- Hasło;
- Potwierdź hasło.

Aplikacja obsługuje pełną walidację każdego pola i nie pozwoli na utworzenie użytkownika jeżeli podany adres e-mail lub nazwa są zajęte.



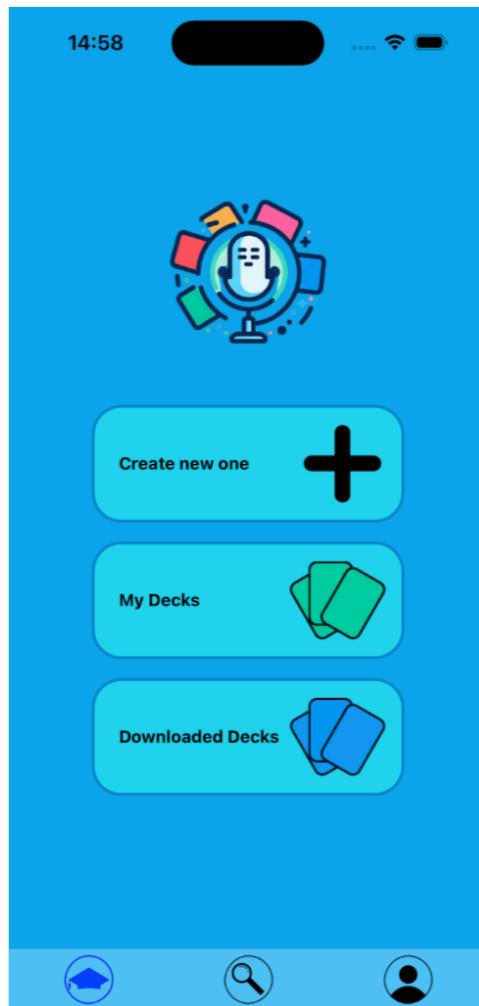
Rysunek 9.25: Ekran rejestracji.

9.2.3 Widok domowy

Po zalogowaniu użytkownik zostaje przekierowany do widoku domowego. Z tego poziomu możliwe jest:

- Utworzenie nowej talii fiszek;
- Przejście do widoku talii użytkownika;
- Przejście do widoku talii pobranych przez użytkownika.

Ponadto, po zalogowaniu użytkownik może w każdej chwili używać dolnego paska nawigacji. Ekran domowy i jemu pochodne znajdują się pod pierwszą ikoną symbolizującą naukę.



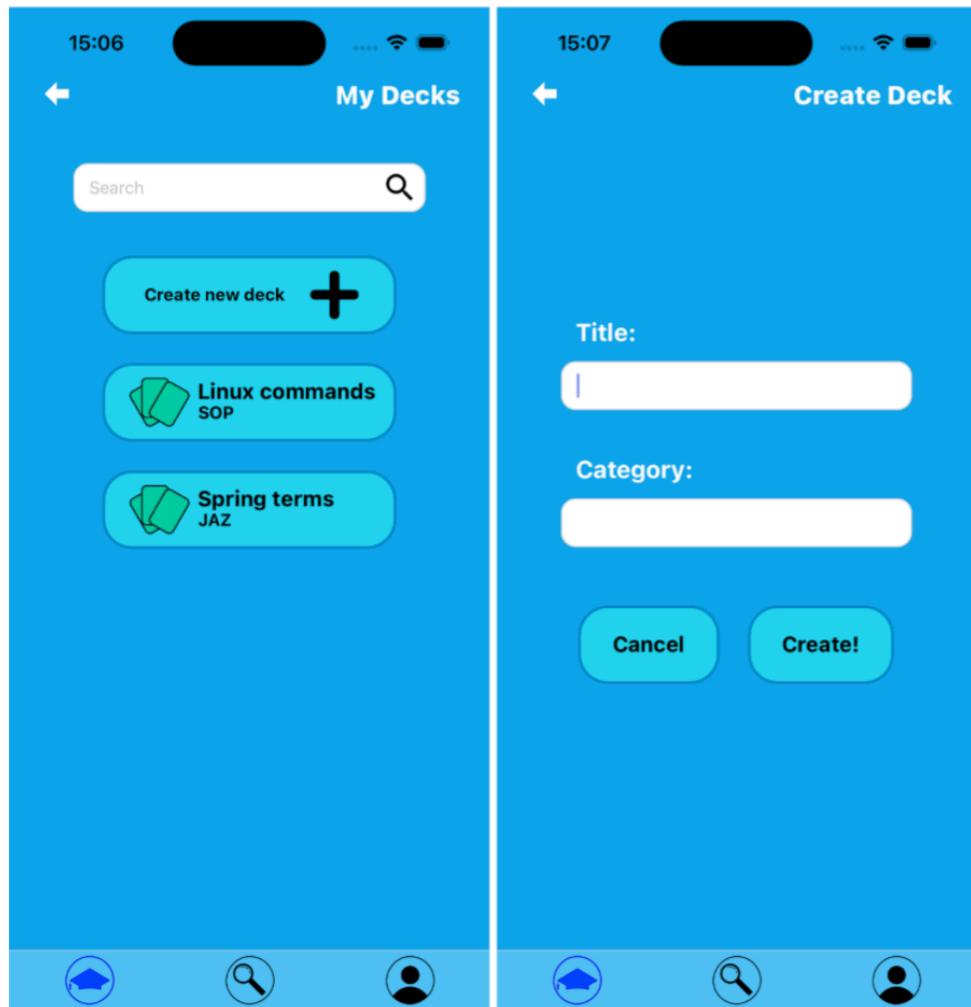
Rysunek 9.26: Ekran domowy.

9.2.4 "My Decks", "My Downloaded Decks" i "Create Deck"

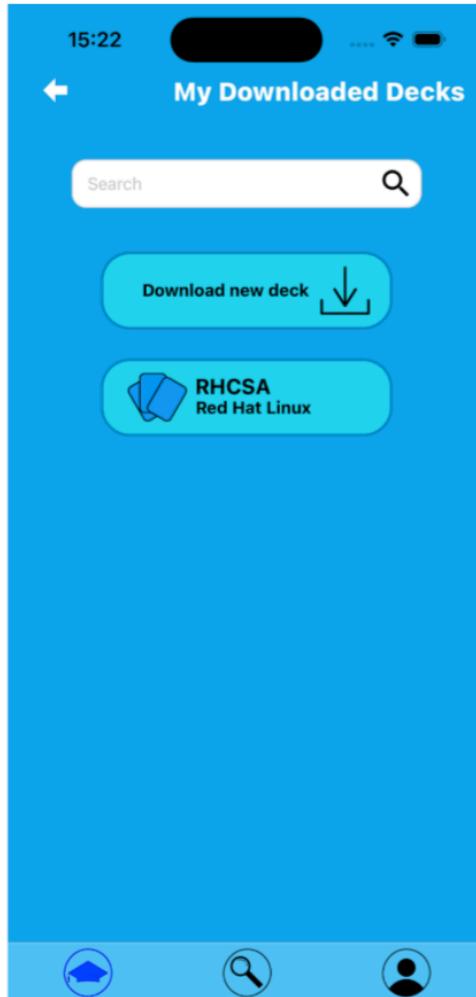
Po wybraniu przycisku "My Decks" w ekranie domowym, użytkownik zostaje przekierowany do widoku w którym przechowywane są jego talie. Z tego poziomu możliwe jest wyszukanie, wybranie lub utworzenie nowej talii.

Podobnie w przypadku wybrania opcji "My Downloaded Decks" - tutaj wyświetlane zostaną pobrane talie, wyszukiwanie oraz przycisk nawigujący do rankingu talii publicznych.

Widok "Create Decks" jest dostępny z ekranu "My Decks" oraz bezpośrednio z ekranu domowego. Aby utworzyć nową talię, wymagane jest podanie jej tytułu i kategorii, oba pola muszą przejść walidacje i nie mogą być puste.



Rysunek 9.27: Widok "My Decks" i "Create Deck".



Rysunek 9.28: Widok "My Downloaded Decks".

9.2.5 "Deck Preview" i "Downloaded Deck Preview"

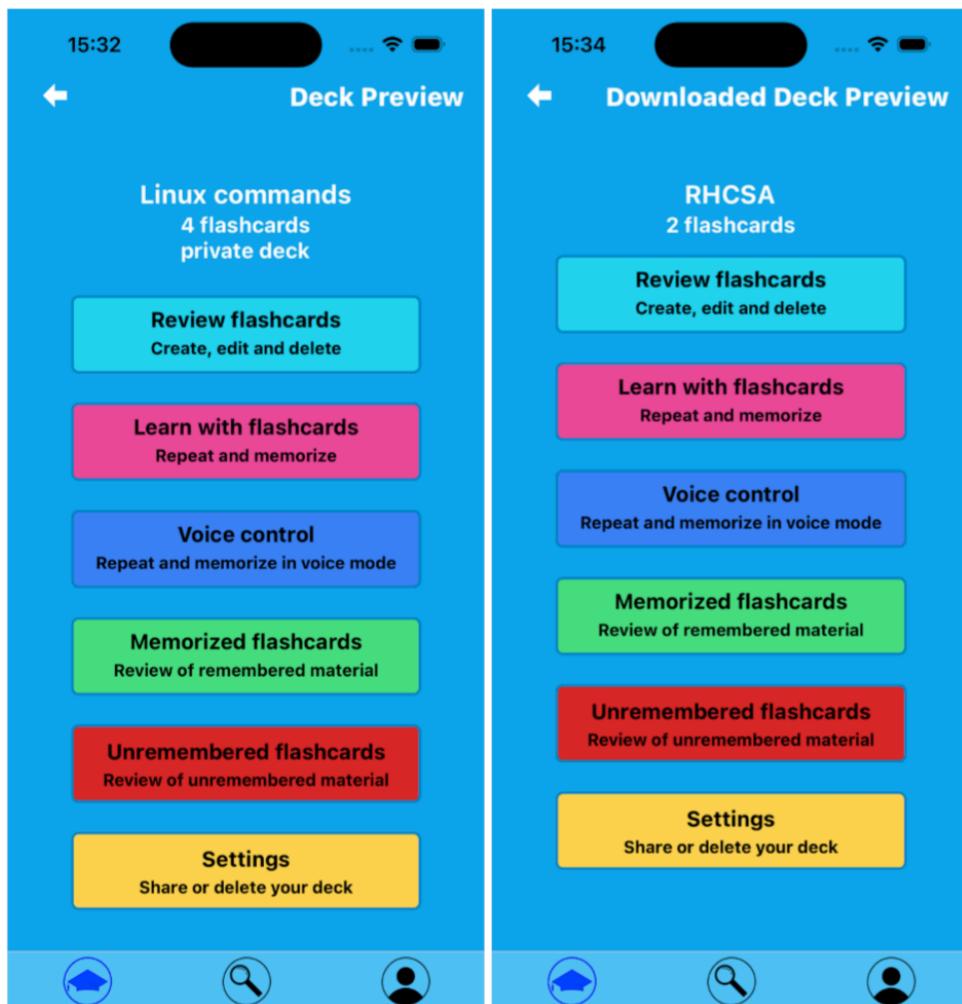
Po wybraniu talii z listy w ekranie "My Decks" lub "My downloaded Decks" użytkownik zostaje przekierowany do ekranu podglądu talii. Widok wyświetla:

- nazwę talii;
- ilość znajdujących się w talii fiszek;
- status - "private" lub "public" (widoczne tylko w ekranie "Deck Preview").

Ponadto możliwa jest nawigacja do kolejnych widoków:

- "Review flashcards";
- "Learn with flashcards";
- "Voice control";
- "Memorized flashcards";

- ”Unmemorized flashcards”;
- ”Settings”.

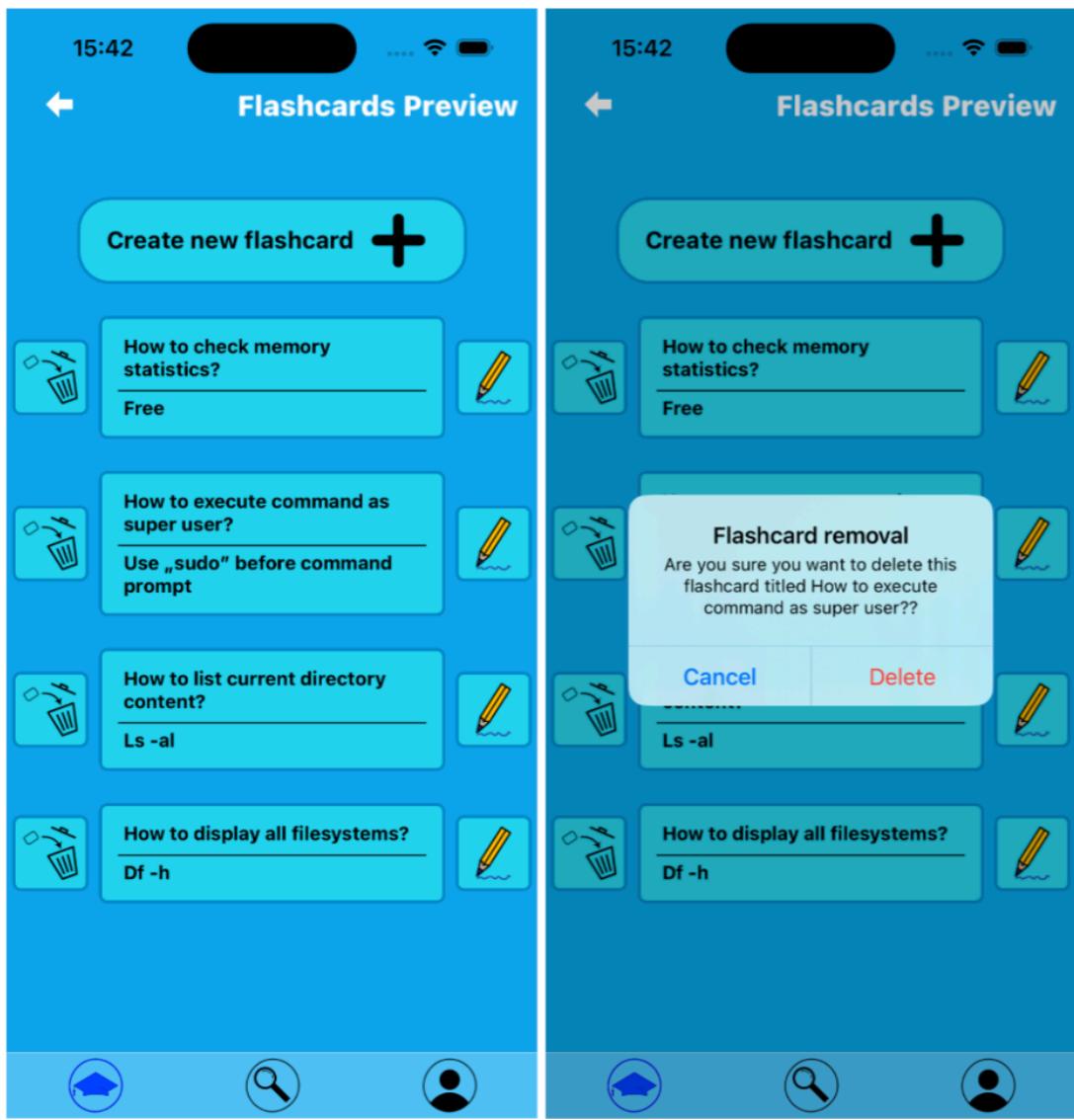


Rysunek 9.29: Widok ”Deck Preview” i ”Downloaded Deck preview”.

9.2.6 ”Review flashcards”

Po wybraniu w widoku podglądu talii przycisku ”Review flashcards”, użytkownik zostanie przeniesiony do podglądu fiszek. W nim wyświetlane są wszystkie fiszki. Funkcjonalności dostępne z tego poziomu to:

- utworzenie nowej fiszki za pomocą przycisku ”Create new flashcard”;
- usunięcie istniejącej fiszki za pomocą przycisku z ikoną kosza;
- edycja istniejącej fiszki za pomocą przycisku z ikoną ołówka.

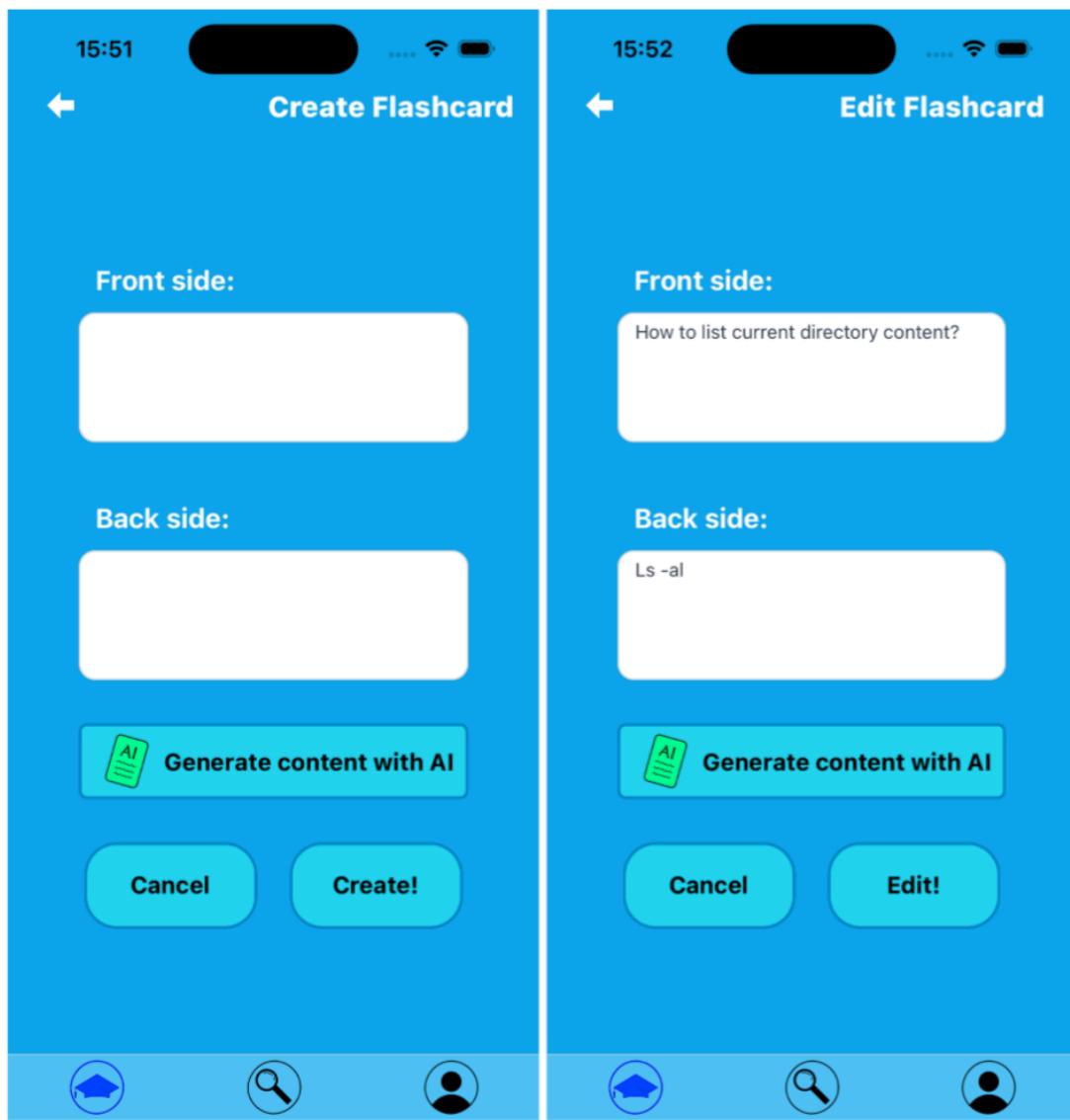


Rysunek 9.30: Widok "Flashcards Preview" i potwierdzenie usunięcia fiszki.

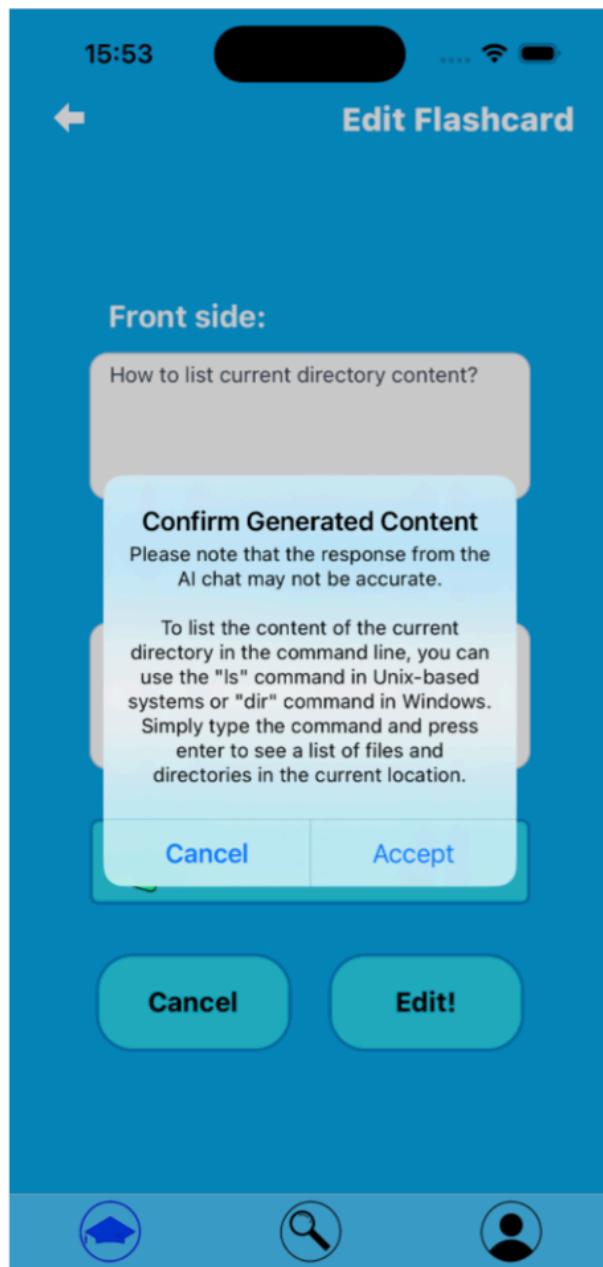
9.2.7 "Create Flashcard" i "Edit Flashcard"

Po wybraniu opcji edycji lub utworzenia fiszki, użytkownik zostanie przekierowany do odpowiedniego widoku w którym możliwe jest:

- uzupełnienie przedniej strony fiszki;
- uzupełnienie tylnej strony fiszki;
- wygenerowanie treści tylnej strony fiszki na podstawie treści strony przedniej;
- zaakceptowanie lub odrzucenie wygenerowanej treści (po naciśnięciu przycisku generowania).



Rysunek 9.31: Widoki "Create Flashcard" i "Edit Flashcard".



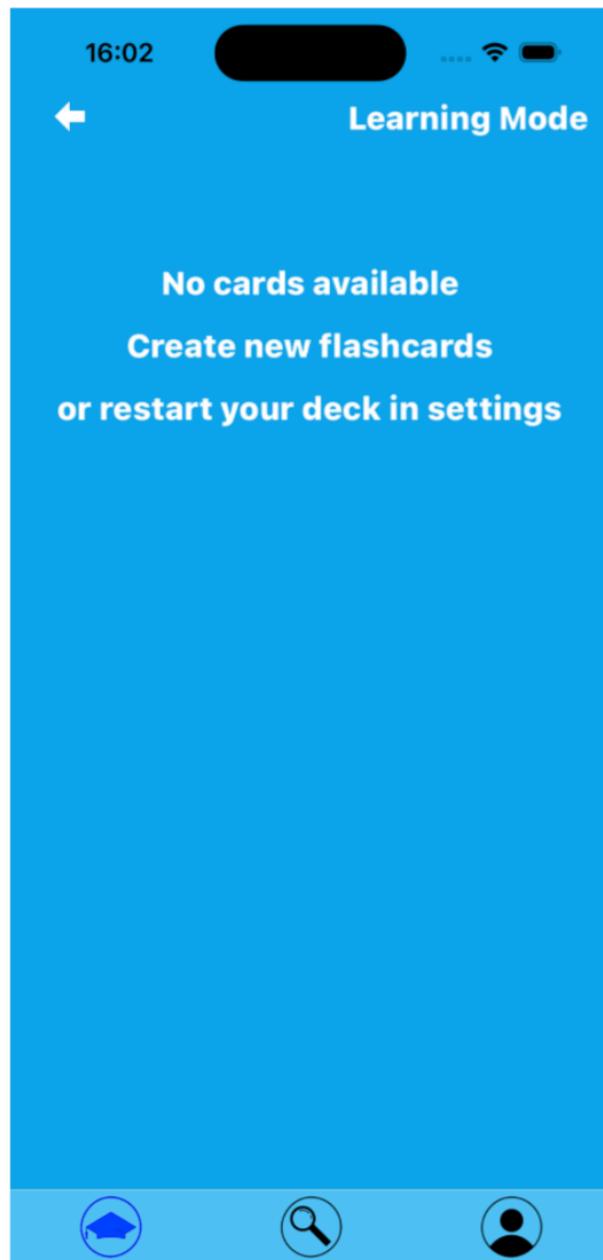
Rysunek 9.32: Podgląd treści wygenerowanej przez AI.

9.2.8 "Learn with flashcards"

Podstawowy tryb nauki, w nim wyświetlane są kolejno fiszki z odkrytymi zagadnieniami. Po naciśnięciu w fiszce, użytkownik może podejrzeć jej definicje. Przejście do kolejnej fiszki odbywa się po odznaczeniu bieżącej fiszki jako tej zapamiętanej za pomocą zielonego przycisku lub jako niezapamiętanej za pomocą przycisku czerwonego.



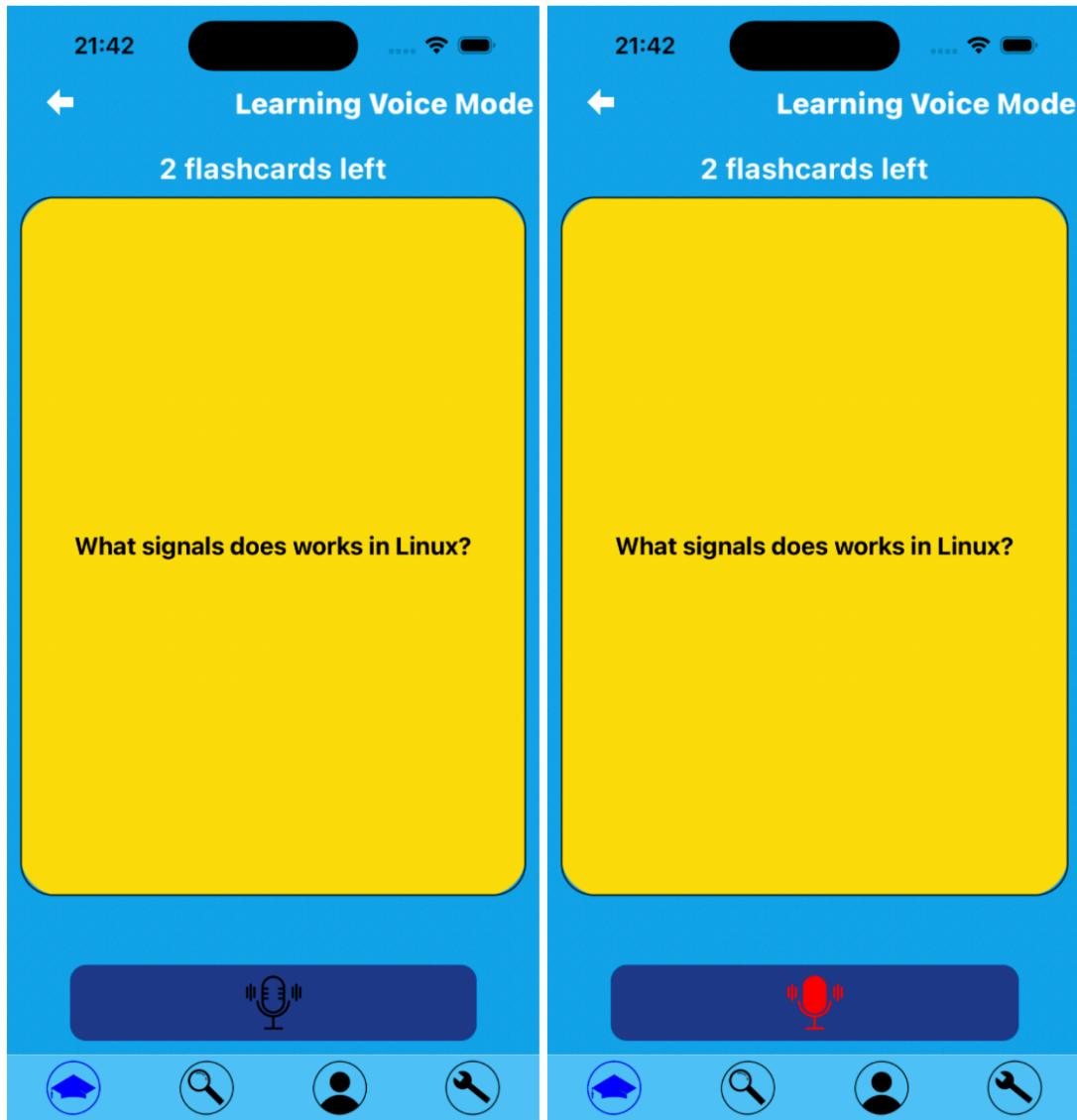
Rysunek 9.33: Podstawowy tryb nauki.



Rysunek 9.34: Widok w przypadku kiedy talia jest pusta lub wszystkie fiszki są oznaczone jako zapamiętane.

9.2.9 "Voice control"

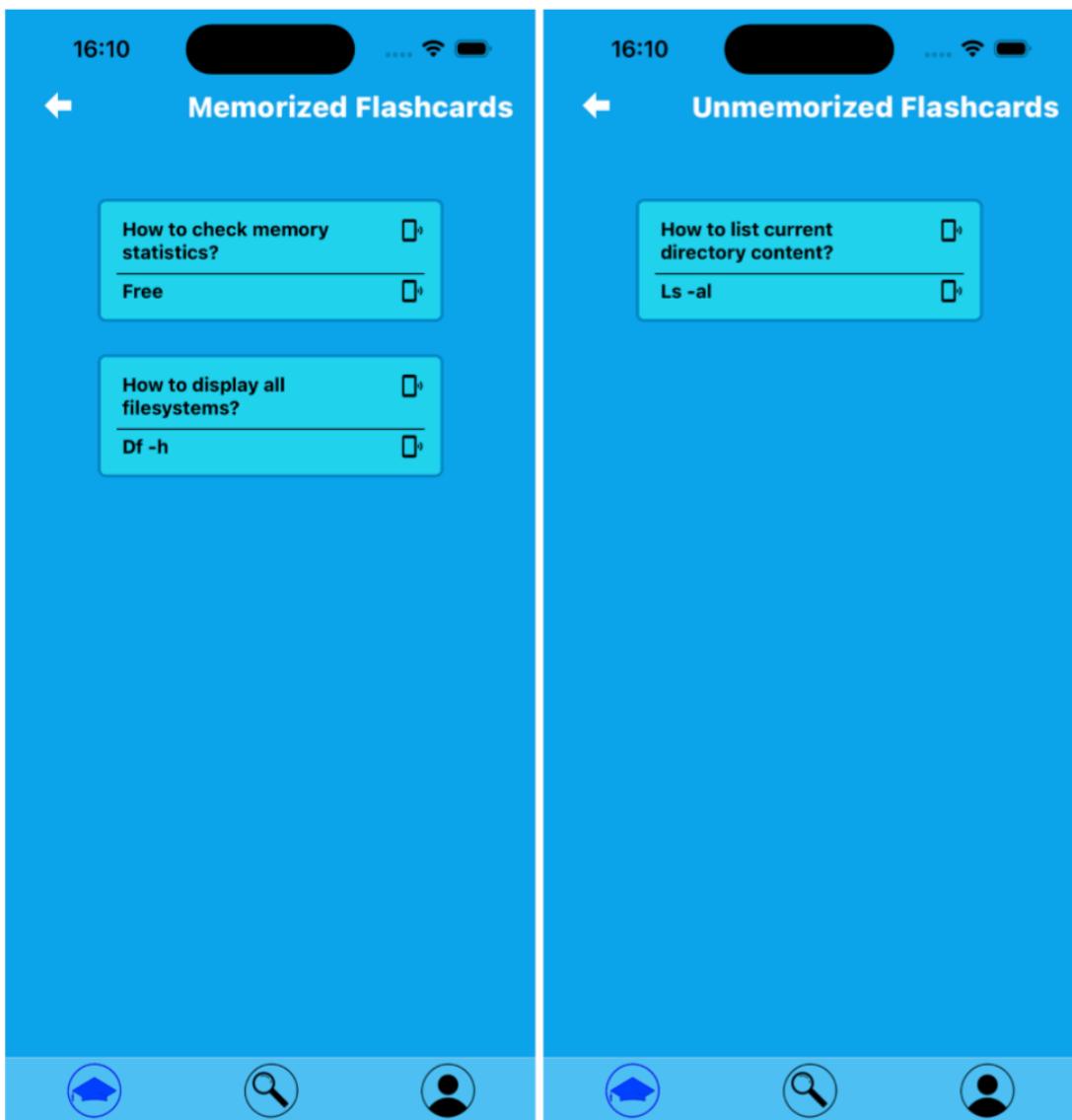
Tryb nauki z obsługą głosową. Widok pozwala na przeglądanie talii za pomocą komend głosowych po naciśnięciu przycisku z ikoną mikrofonu.



Rysunek 9.35: Widok głosowego trybu nauki.

9.2.10 "Memorized flashcards" i "Unmemorized flashcards"

Widoki podglądowe dla fiszek oznaczonych jako zapamiętane i niezapamiętane. Poza możliwością przeglądania fiszek, możliwe jest odsłuchanie ich treści.

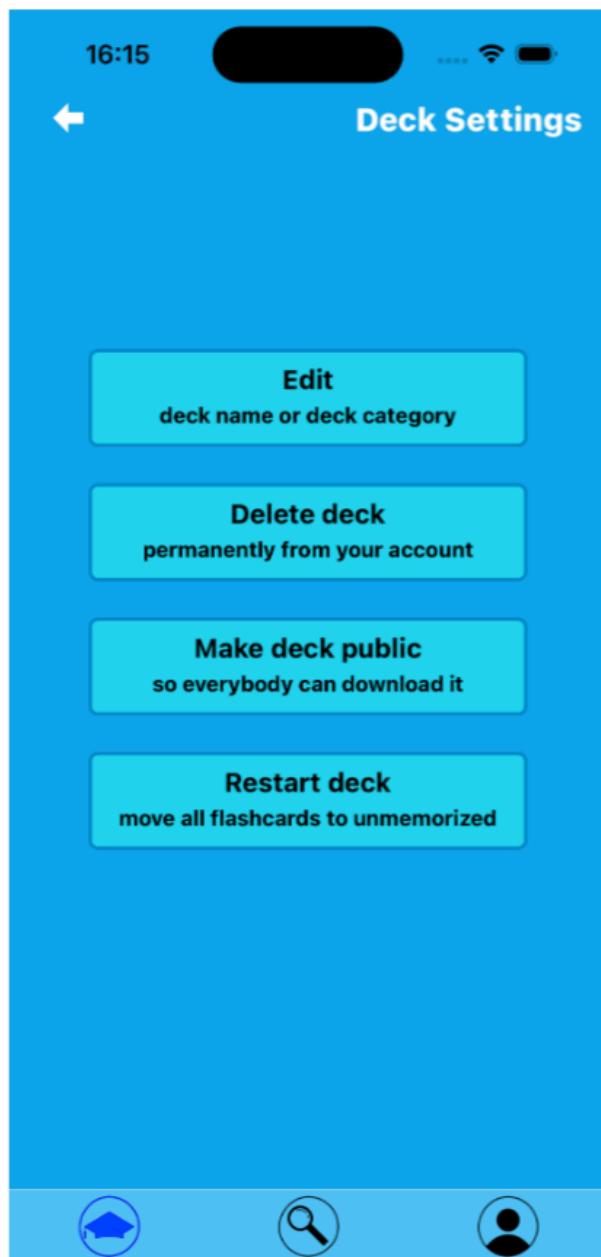


Rysunek 9.36: Widoki "Memorized flashcards" i "Unmemorized flashcards".

9.2.11 "Settings"

Widok opcji danej talii. Z tego poziomu jest możliwe:

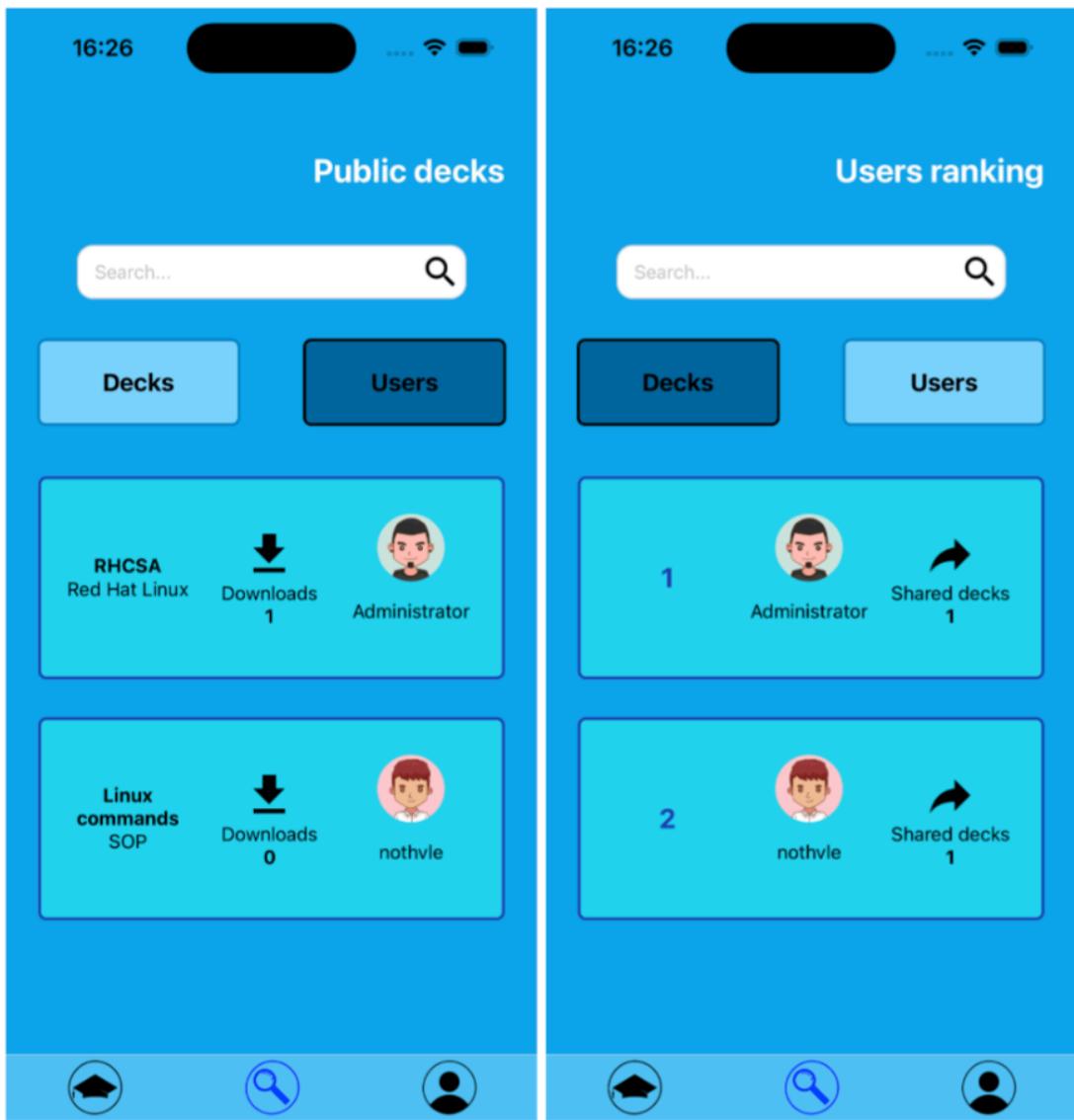
- edycja nazwy i kategorii talii (widok bliźniaczy do "Create Deck");
- usunięcie talii permanentnie;
- udostępnienie talii publicznie lub cofnięcie udostępnienia (opcja niedostępna dla talii bez fiszek oraz talii pobranych z rankingu);
- restart talii, czyli oznaczenie wszystkich fiszek jako niezapamiętych.



Rysunek 9.37: Widok "Deck Settings".

9.2.12 "Public decks" i "Users ranking"

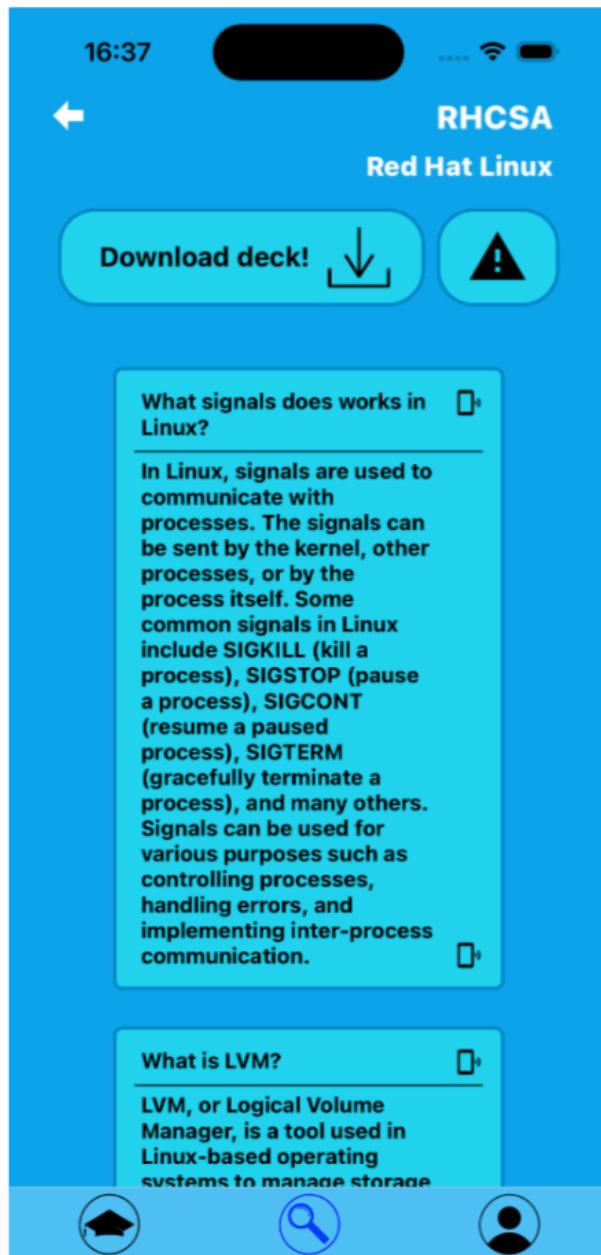
Widok "Public decks" jest dostępny z poziomu dolnego paska nawigacji pod ikoną lupy. Wyświetla listę wszystkich talii udostępnionych publicznie, posegregowanych po liczbie pobrań. Umożliwia przełączenie na widok rankingu użytkowników posegregowanych po liczbie udostępnionych talii. W obu zakładkach możliwe jest wyszukanie zarówno talii jak i użytkowników.



Rysunek 9.38: Widok "Public decks" i "Users ranking".

9.2.13 Podgląd talii publicznej

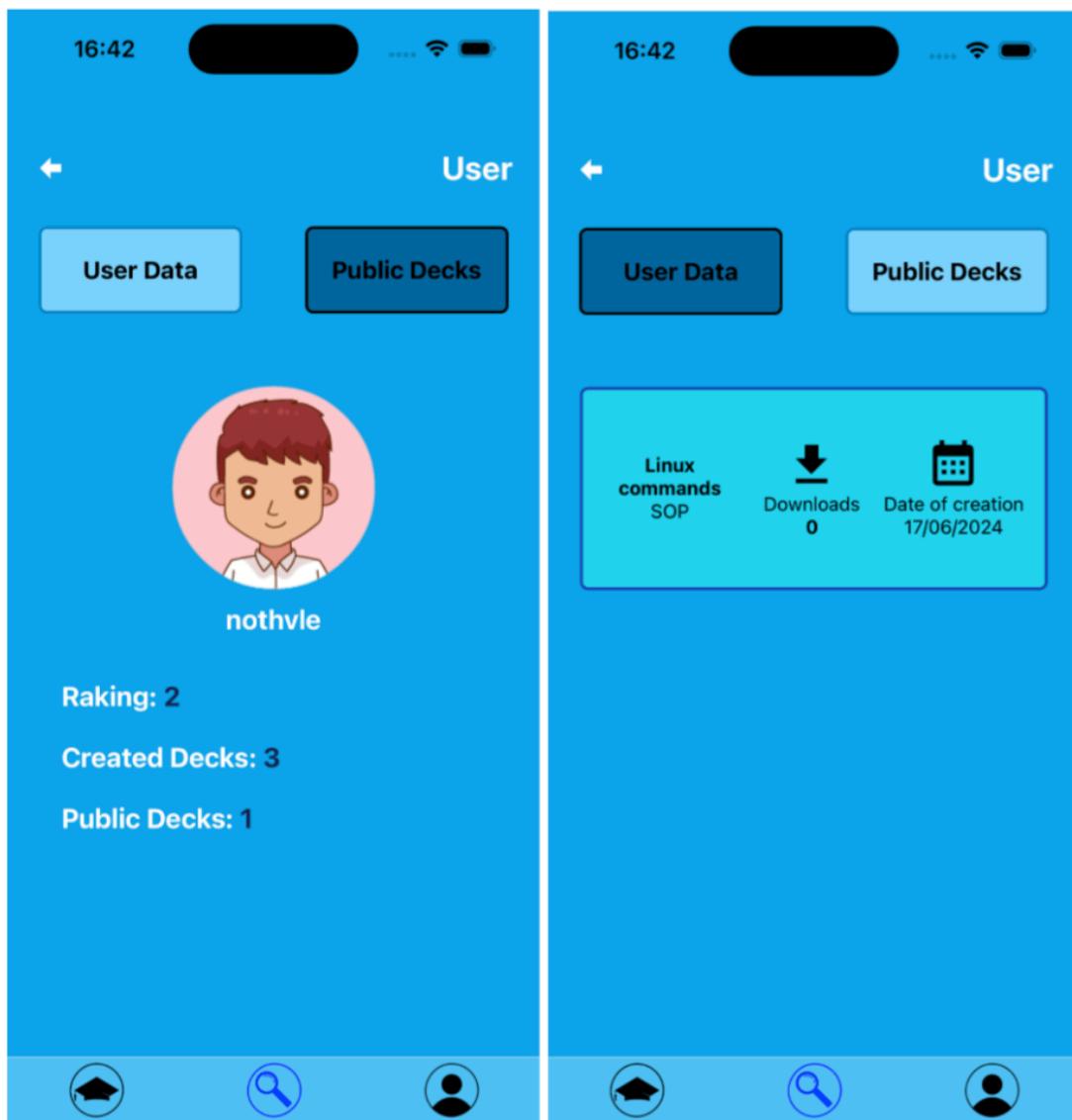
Z poziomu rankingu talii publicznych możliwe jest wybranie interesującej dla użytkownika pozycji. Następuje wtedy przeniesieniu do widoku podglądu talii i jej fiszek. Dostępne funkcjonalności to pobranie talii lub zgłoszenie jej moderatorowi.



Rysunek 9.39: Podgląd talii udostępnionej publicznie.

9.2.14 Podgląd innego użytkownika

Możliwe jest podejrzenie innego konta użytkownika poprzez wybranie go z rankingu. Widok zawiera podstawowe informacje oraz udostępnione talie.

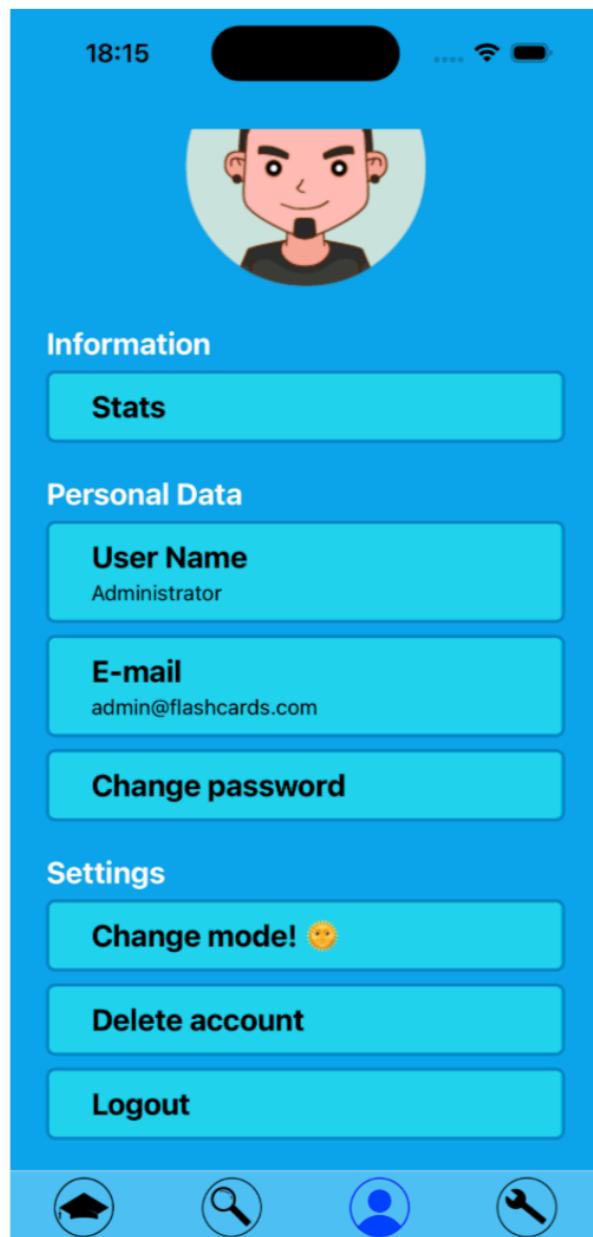


Rysunek 9.40: Podgląd użytkownika i udostępnionych przez niego talii.

9.2.15 Profil

Widok profilu jest dostępny z poziomu dolnego paska nawigacji pod ikoną przedstawiającą sylwetkę. W profilu widoczne są:

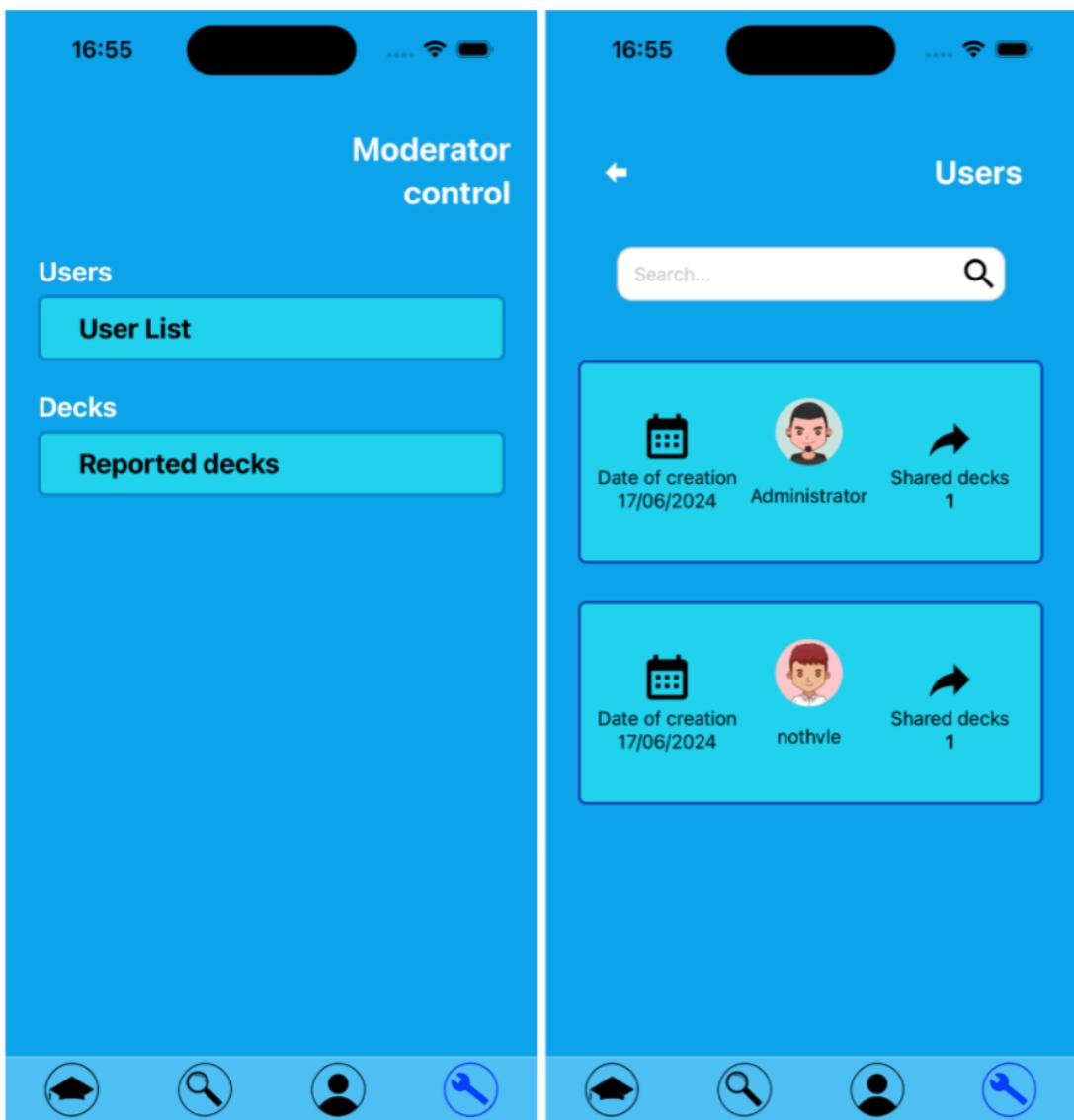
- Informacje o zalogowanym użytkowniku;
- Możliwość zmiany adresu e-mail;
- Możliwość zmiany hasła;
- Możliwość zmiany motywu aplikacji na ciemny;
- Możliwość usunięcia konta;
- Możliwość wylogowania.



Rysunek 9.41: Widok profilu użytkownika.

9.2.16 Widok narzędzi moderatora

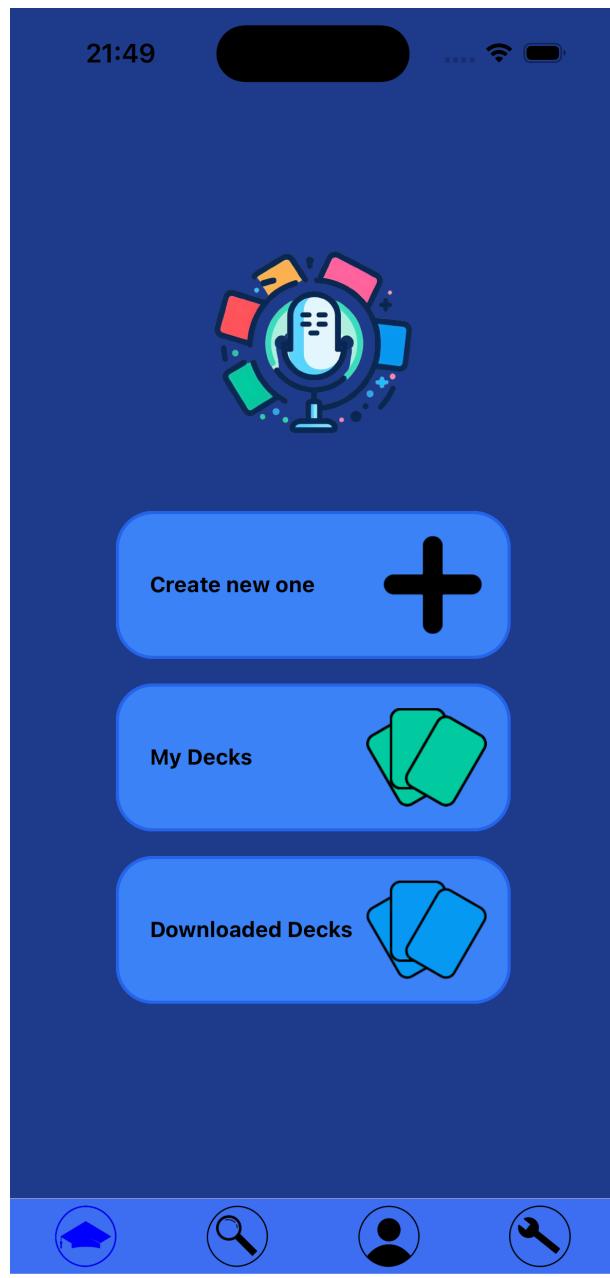
Widok jest dostępny jedynie dla użytkowników wyznaczonych do roli moderatorów i administratorów. Pozwala na zarządzanie pozostałyimi użytkownikami oraz zgłoszonymi taliami.



Rysunek 9.42: Widok panelu moderatora.

9.2.17 Ciemny motyw aplikacji

Ciemny motyw jest dostępny w panelu użytkownika. Po jego przełączeniu zmienia się gama kolorów aplikacji.



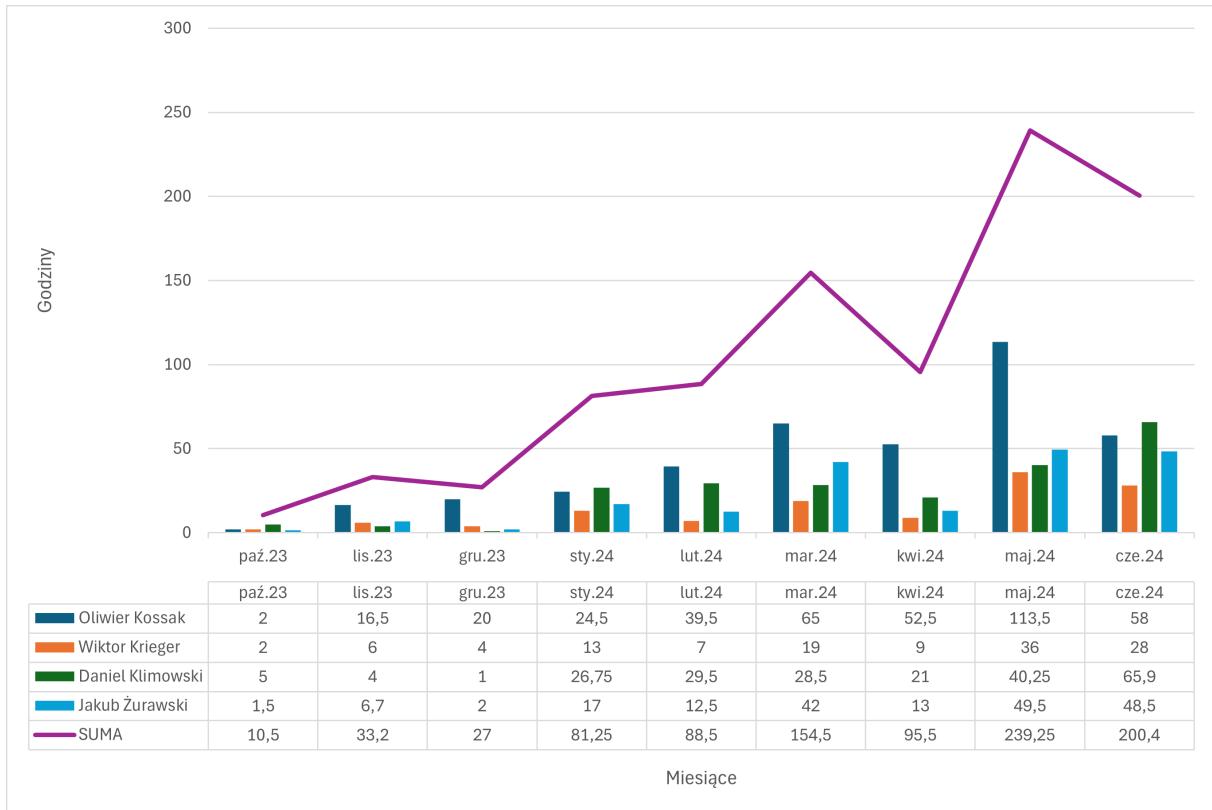
Rysunek 9.43: Ekran domowy po przełączeniu na “Dark mode”.

Rozdział 10

Nakład pracy

Rozdział opisuje wkład i zadania poczynione przez każdego członka zespołu, które zostały wykonane w celu ukończenia projektu. Na początku pierwszego semestru utworzyliśmy listę potencjalnych pomysłów, jakich moglibyśmy się podjąć w celu utworzenia projektu inżynierskiego. Po zdecydowaniu się na pomysł związany z nauką z wykorzystaniem metody fiszek przeszliśmy do wypełniania dokumentacji. Ustaliliśmy wymagania i zakres projektu. Po wypełnieniu dokumentów przeszliśmy do tworzenia graficznych widoków aplikacji, pozwoliło nam to na wypracowanie wspólnej wizji projektu i ponowne przedyskutowanie zaproponowanych rozwiązań.

W drugim semestrze przeszliśmy do implementacji projektu. Postęp pracy i zadania monitorowaliśmy przy pomocy narzędzia do zarządzania projektem Jira. Implementacja projektu odbywała się w sprintach. Po każdym ukończonym przyroście można było zauważać znaczące zmiany w postępie rozwoju projektu. Jako zespół byliśmy także pochlonięcie pracami nad tworzeniem niniejszej książki. Za pomocą wykresu został przedstawiony czas, jaki członkowie zespołu spędzili nad projektem.



Rysunek 10.1: Wykres ilustrujący łączny nakład pracy.

10.1 Oliwier Kossak

Na początku 1 semestru pisania pracy zająłem się wypełnieniem dokumentu założeń wstępnych. Rozdziału dokumenty, które zostały przeze mnie napisane to analiza konkurencji, zakres systemu i wymagania jakościowe. Następnie zająłem się wypełnianiem wymagań w specyfikacji wymagań wstępnych. Gdy wymagania funkcjonalne zostały określone zacząłem tworzyć w canvie atrapę aplikacji mobilnej. Kreowanie atrapy pozwoliło na konsolidację informacji i na ponowną analizę niektórych wymagań. Byłem też osobą odpowiedzialną za konsultowanie naszych wymagań i rozwiązań z promotorem, co czyniło mnie w pewnym sensie leaderem projektu.

W drugim semestrze pisania pracy byłem odpowiedzialny za backend, frontend, testy i pisanie pracy inżynierskiej. Przy pisaniu backendu byłem odpowiedzialny za implementację modeli fiszek, talii, i zgłoszeń. Po utworzeniu modeli zabrałem się za napisanie do nich metod CRUD. Moim zadaniem było także utworzenie rozwiązań związanych ze sztuczną inteligencją w naszej aplikacji. Na początku zająłem się komunikacją z ChatGPT. Napisałem metodę POST, która pozwalała na wysłanie zapytania do ChatGPT. Następnie zająłem się implementacją modelu nlp, który służył do wyszukiwania komendy najbliższej zbliżonej semantycznie do tego, co powiedział użytkownik podczas korzystania z trybu sterowania głosem. Do moich zadań związanych z implementacją strony webowej należało utworzenie następujących widoków:

- Strona domowa;

- Ranking użytkowników i talii;
- Panel moderatora;
- My decks;
- Public decks;
- Tryb uczenia się;
- Create deck;
- Voice control i memorized i not memorized flashcards.

Zaimplementowane przez mnie funkcjonalności na stronie internetowej to:

- Importowanie talii;
- Tryb uczenia się;
- Tryb sterowania głosem;
- Generowanie treści przy użyciu ChatGPT;
- Wylogowanie się użytkownika;
- Tworzenie talii fiszek;
- Edycja talii;
- Usunięcie talii;
- Syntezator mowy.

Byłem też osobą odpowiedzialną za wykonywanie testów. Gdy, któryś z członków zespołu dodawał nowy widok lub funkcjonalność do aplikacji, wykonywałem testy akceptacyjne w celu sprawdzenia czy nowy element aplikacji zachowywał się zgodnie z oczekiwaniami. Oprócz testów akceptacyjnych wykonalem także testy integracyjne w celu sprawdzenia, czy endpointy komunikują się z bazą danych. W książce napisałem poniższe rozdziały:

- 3.3 Grupa docelowa;
- 3.4 Analiza rozwiązań konkurencji ;
- 3.5 Analiza szans oraz zalet względem konkurencyjnych rozwiązań;
- 3.6 Analiza ryzyka;
- 3.7 Społeczny aspekt projektu;
- 7.1 Faza planowania;
- 7.2 Faza implementacji;

- Opisy sprintów w rozdziale 7;
- Szczegóły implementacji w rozdziale 7;
- Rozdział 8 - Testy;
- Rozdział 10 - Nakład pracy;
- Rozdział 11 - Podsumowanie;
- Bibliografia.

10.2 Wiktor Krieger

W czasie semestru zimowego, oprócz pomocy przy wypełnianiu dokumentacji, zająłem się wstępna konfiguracją projektu w serwisie Jira. Ponadto podjąłem się praktycznej analizy konkurencji, testując aplikację z rozwiązaniami zbliżonymi do tych, które planowaliśmy zaimplementować w naszym produkcie. Testy te obejmowały ocenę użyteczności, funkcjonalności oraz interfejsu użytkownika konkurencyjnych aplikacji, co dostarczyło nam cennych informacji pomocnych w ulepszaniu naszego produktu. Równocześnie pomagałem przy wyszukiwaniu prac naukowych dotyczących społecznych aspektów naszego rozwiązania. Na wczesnym etapie projektu stworzyłem również logo, które miało subtelnie sugerować, co oferuje nasz produkt. Logo przedstawia duszka (asystenta AI) pod postacią mikrofonu (funkcje głosowe) otoczonego kolorowymi kartami (fiszkami). Proces tworzenia logo był dla mnie szczególnie ważny, ponieważ zależało mi, aby było ono nie tylko estetyczne, ale również intuicyjnie komunikowało funkcje naszego produktu. Natomiast większość czasu w tamtym okresie poświęciłem na projektowaniu warstwy wizualnej aplikacji w Canvie, co pomogło w stworzeniu wspólnej wizji projektu i zostało wykorzystane jako wzór dla finalnego produktu.

W kolejnym semestrze byłem odpowiedzialny za wizualne aspekty aplikacji. Moje zadania obejmowały projektowanie tła, dobór palety kolorów oraz tworzenie ikon. Dbałem o to, aby wszystkie te elementy były spójne i estetyczne. W części programistycznej zajmowałem się frontendem i częściowo backendem widoków aplikacji webowej, poświęcając najwięcej pracy na ekran profilu. Moje zadania związane z implementacją strony webowej obejmowały utworzenie następujących widoków:

- Rejestracja;
- Logowanie;
- Profil;
- Widok statystyk.

Zaimplementowane przez mnie funkcjonalności na stronie internetowej to:

- Rejestracja i logowanie;
- Wybór Awatara;

- Zmiana danych;
- Wyświetlanie statystyk;
- Usuwanie konta;
- Przejście do panelu moderatora dostępne tylko dla wybranych kont;
- Obsługa błędów w formularzach.

Aby zwizualizować nasz indywidualny wkład wykorzystałem dane zbierane przez cały czas pracy nad projektem i dostosowałem je tak, aby stworzyć czytelny wykres widoczny na początku tego rozdziału. Oprócz tego, w książce napisałem poniższe rozdziały:

- 7.3.1.2 Rejestracja (web);
- 7.3.2.2 Logowanie (web);
- 7.3.4.3 Edycja danych użytkownika (web);
- 7.3.5.3 Usuwanie konta (web);
- 11.2 Porzucone pomysły (aktywacja konta i reset hasła).

10.3 Jakub Żurawski

W pierwszym semestrze, od momentu rozpoczętych prac nad projektem, byłem głównie odpowiedzialny za diagramy oraz część dokumentu założeń wstępnych i część dokumentu specyfikacji wymagań technicznych. W dokumencie założeń wstępnych byłem odpowiedzialny za rozdziały takie jak: opis problemu, cele systemu i kontekst systemu. Z kolei w dokumencie specyfikacji wymagań systemowych byłem odpowiedzialny za wymagania ogólne i dziedzinowe, wymagania pozafunkcjonalne oraz wymagania na środowisko docelowe. Po ukończeniu wyżej opisanych rozdziałów przystąpiłem do tworzenia diagramu przypadków użycia. Diagramy początkowo były tworzone przy pomocy Draw.io - z biegiem czasu okazało się, że wykorzystanie Enterprise Architect daje najlepsze rezultaty.

W drugim semestrze, po podziale zespołu według zakresu pracy, byłem odpowiedzialny za backend, aplikację mobilną oraz pisanie pracy inżynierskiej. W backendzie byłem odpowiedzialny głównie za bezpieczeństwo api, autoryzację użytkownika oraz zarządzanie użytkownikami, np. możliwość edycji danych i usuwanie użytkowników z systemu. Miałem również zajmować się rozwiązyaniem sztucznej inteligencji w naszej aplikacji, jednak problemy napotkane podczas tworzenia aplikacji mobilnej uniemożliwiły mi pełnienie tej roli. W początkowej fazie pisania skupiłem się głównie na autoryzacji. Zacząłem od utworzenia modeli bazy danych dla tokenów, zużytych tokenów i użytkownika. Następnie utworzyłem dependencje do dekodowania tokenu użytkownika i weryfikacji jego ważności. Następnie zabrałem się tworzenie kolejnej dependencji "RoleAccessChecker", która jak nazwa wskazuje, miała sprawdzać rolę użytkownika i blokować dostęp użytkownikom bez uprawnień. Kolejnym krokiem było zabezpieczeniem api przed nieautoryzowanym aplikacjami, adresami czy headerami. W tym celu utworzyłem customowe

middleware'y. Do middleware'ów dodałem również sprawdzanie, czy token użytkownika nie znajduje się w tabeli zużytych tokenów. Gdy bezpieczeństwo było już gotowe, zająłem się tworzeniem widoków do autoryzacji i użytkowników, w celu przetestowania bezpieczeństwa, funkcjonalności i poprawy błędów. Po zaimplementowaniu wstępnych funkcjonalności na backendzie, zająłem się aplikacją mobilną, gdzie odpowiedzialny byłem za weryfikacje czy użytkownik jest zalogowany, a także utworzeniu następujących paneli i widoków jak:

- Panel użytkownika;
- Panel moderatora;
- Panel rankingu i publicznych deck'ów;
- Widok uczenia z obsługą głosu.

Dodatkowo byłem odpowiedzialny za animację np. Loadera, który pokazuję się podczas czytania danych. Z zaimplementowanych funkcjonalności w aplikacji mogę wymienić:

- Weryfikacja czy użytkownik jest zalogowany;
- Metoda 'request' która została użyta w wielkości serwisach;
- Przeglądanie rankingu i udostępnionych deck'ów razem z możliwością filtrowania po słowach kluczowych;
- Ograniczenie ilości zaciąganych danych z api;
- Wylogowywanie użytkownika;
- Edycja danych użytkownika;
- Usuwanie konta;
- Potwierdzanie zmian hasłem;
- Usuwanie zgłoszeń i użytkowników;
- Sterowanie głosem podczas nauki.

Podczas pisania książki byłem odpowiedzialny za rozdziały:

- Rozdział 4 - Organizacja pracy;
- Rozdział 7 - Implementacja;
- Rozdział 11 - Podsumowanie;
- Implementacja w LaTeX.

10.4 Daniel Klimowski

W pierwszym semestrze realizacji projektu inżynierskiego prace skupiły się początkowo na fazie planowania oraz definicji wymagań i założeń w dokumentacji DZW i SWS. W dokumencie założeń wstępnych zredagowałem początkowo punkty od 1. do 8. tj. "Opis problemu", "Cele systemu", opis sekcji "Kontekst systemu", "Zakres systemu (funkcjonalności)", "Wymagania jakościowe i inne", "Wizja konstrukcyjna" i "Ograniczenia". Które później były uzupełniane i modyfikowane przeze mnie oraz zespół projektowy. Tak samo podjąłem się zredagowaniu części opisowej w dokumencie specyfikacji wymagań systemowych. Mniej więcej w tym samym czasie należało zdefiniować swój zakres obowiązków w projekcie, zdecydowałem się wybrać implementacje infrastruktury serwerowej, wspólnie z Jakubem implementację aplikacji mobilnej oraz nadzorować kod niniejszej pracy implementowany w LaTeX. W fazie projektowania wykonałem dodatkowo min. draft projektu w LaTeX, diagram architektury oraz zainicjowałem początkowy projekt aplikacji mobilnej. Pod koniec pierwszego semestru prac spędziłem dużo czasu na nauce języka JavaScript oraz frameworku React Native. Z tymi dwiema technologiami nie miałem okazji pracować wcześniej. Na potrzebny projektu zrealizowałem kurs, na który przeznaczyłem ok. 26 godzin.

Drugi semestr realizowania projektu zaczął się od intensywnych prac związanych z rozwojem aplikacji mobilnej oraz redagowaniem niniejszej książki. W momencie w którym zarówno backend jak i aplikacja webowa i mobilna były już funkcjonalne w znacznym stopniu, podszedłem do implementacji infrastruktury serwerowej. Opis działań rozdzielę poniżej w tych trzech obszarach:

Aplikacja mobilna:

- Widoki logowania, rejestracji oraz zapomniałem hasła (bez łączenia z API);
- Widok ekranu głównego;
- Widok "My decks" z funkcjonalnościami wyboru i wyszukiwania talii;
- Widok "My downloaded decks" z funkcjonalnościami wyboru i wyszukiwania pobranych talii;
- Widok "Create new deck" z funkcjonalnościami tworzenia nowej talii;
- Widok "Display Deck" z funkcjonalnościami podglądu decku oraz szeregiem opcji;
- Widok "Display Flashcards" z funkcjonalnościami podglądu, tworzenia, edycji i usuwania fiszek z talii;
- Widok "Create Flashcard" z funkcjonalnościami tworzenia nowych fiszek i generowania treści przy pomocy ChatGPT;
- Widok "Edit Flashcard" z funkcjonalnością edytowania istniejących fiszek i generowania treści przy pomocy ChatGPT;
- Widok "Learning Mode" oferujący tryb nauki;
- Widok "Memorized Flashcards" i "Unmemorized Flashcards" wyświetlający fiszki oznaczone jako przyswojone i nieprzyswojone z możliwością odsłuchu treści;

- Widok ‘Deck Settings’ z funkcjonalnościami upublicznienia/ukrycia talii, usunięcia talii, zrestartowania talii oraz edycji talii;
- Widok “Display Public Deck” z funkcjonalnościami podglądu, pobrania i zgłoszenia publicznej talii.

Serwer:

- Zainicjowanie maszyny wirtualnej z systemem Ubuntu 20.04 w platformie Azure;
- Konfiguracja sieci w platformie Azure;
- Implementacja aplikacji webowej, api oraz kontenera bazy danych na serwerze;
- Konfiguracja generalna i sieciowa serwera;
- Zakup, wygenerowanie i implementacja tokenu ChatGPT na potrzeby funkcjonalności API i aplikacji.

Książka:

- Wstęp;
- Rozdział 1 - Wstęp;
- Rozdział 2 - Omówienie problemu;
- Rozdział 3 - Projekt w kontekście - sekcja 3.1 i sekcja 3.2;
- Rozdział 5 - Analiza wymagań;
- Rozdział 6 - Architektura projektu i użyte technologie;
- Rozdział 7 - Implementacja;
- Rozdział 9 - Prezentacja projektu;
- Rozdział 10 - Nakład pracy;
- Rozdział 11 - Podsumowanie;
- Generalne poprawki tekstu;
- Implementacja w LaTeX.

Rozdział 11

Podsumowanie

Tworzenie projektu postawiło przed każdym członkiem zespołu nowe wyzwania programistyczne i grupowe. Pozwoliło nam to na poszerzenie naszych umiejętności programistycznych, analitycznych i twórczych. Różne podejście członków zespołu pozwoliło nam na poprawienie naszych umiejętności współpracy.

11.1 Problemy i wyzwania

Realizacja rozległego projektu informatycznego obejmującego równoczesną implementację działających ze sobą nawzajem systemów jest skomplikowanym przedsięwzięciem, które niesie ze sobą szereg problemów i wyzwań. Te w podjętym projekcie wynikały głównie ze specyfiki zastosowanych technologii oraz w dużej mierze z braku wcześniejszego doświadczenia członków zespołu z taką skalą projektu. W niniejszym rozdziale przybliżone zostaną najważniejsze problemy i wyzwania napotkane w trakcie implementacji.

11.1.1 Pycharm Community Edition

Pierwszym poważnym spośród napotkanych problemów były ograniczone możliwości PyCharm Community Edition, środowisko w wersji bezpłatnej nie ma możliwości połączenia się z WSL (Windows Subsystem for Linux), na którym znajdowało się środowisko Ubuntu wraz z projektem do uruchomienia. Rozwiązaniem problemu było skorzystanie z pełnej wersji PyCharm.

11.1.2 Kontener bazy danych

Następnym problemem było skonfigurowanie aplikacji na komputerze jednego z członków projektu. W momencie, w którym była dokonywana próba uruchomienia kontenera dockerowego dla bazy danych, pojawiał się błąd związany z brakiem połączenia z mysql. Po trzydniowej analizie problemu okazało się, że hasło do bazy danych w zmiennych środowiskowych było nieaktualne i różniło się od poprzedniego hasła jednym znakiem.

11.1.3 Loader w aplikacji mobilnej

Jednym z problemów który zahamował rozwój aplikacji mobilnej był błąd spowodowany przez Loader. Przyczyną było użycie biblioteki "moti", która po wywołaniu powodowała błąd o kodzie "useContext is null" uniemożliwiając komplikację aplikacji dla systemu iOS. Możliwym rozwiązaniem naprawienia tego błędu jest użycie innej biblioteki.

11.1.4 WSL i model nlp

Model nlp nie działa na WSL. Po dodaniu do aplikacji modelu nlp okazało się, że aplikacja nie przestaje działać. Backend się zawiesza i zostaje zwrócona informacja segmentation fault, który oznacza brak dostępu do pamięci. Prawdopodobnym powodem wystąpienia problemu jest brak dostępu wsl do pełnych zasobów komputera. Jedynym rozwiązaniem było uruchomienie projektu na komputerze z zainstalowanym linuxem.

11.1.5 Obsługa przy pomocy Mozilla Firefox

Strona internetowa nie działa poprawnie w przeglądarce Mozilla Firefox. Z nieznanych przyczyn pozyjonowanie talli i użytkowników nie wczytuje się poprawnie i talie wychodzą poza ekran. Firefox ma ograniczone wsparcie dla Web Speech API, co uniemożliwia korzystanie z funkcjonalności pozwalającej sterować talię fiszek używając komend głosowych.

11.1.6 Api i HTTPS

Przy realizacji rozproszonej infrastruktury wewnętrz serwera, trudna okazała się implementacja pełnego HTTPS. Strona korzystała w protokołu HTTPS, jednak łączyła się z api przez protokół HTTP. Przeglądarki internetowe mogą traktować taką zawartość jako "niebezpieczną". Wymagane jest wtedy wyłączenie zabezpieczeń przeglądarki aby korzystanie ze strony było możliwe. Rozwiązanie tego problemu wymagało zaimplementowania zaawansowanej konfiguracji Nginx która pozwoliła przekierowywać cały ruch z domeny po https na api.

11.1.7 Zatrzymanie nasłuchiwanie mikrofonu

Na stronie webowej po zatrzymaniu trybu sterowania głosem mikrofon jest jeszcze aktywny przez kilka sekund i strona reaguje na wypowiedziane przez użytkownika komendy głosowe. Próby zmiany zachowania działania mikrofonu tak żeby po zatrzymaniu kontroli głosowej przestał od razu nasłuchiwać nie przyniosły oczekiwanych rezultatów.

11.1.8 Strumieniowanie głosu w aplikacji Expo Go

W późnym etapie projektu napotkano problem związany ze strumieniowaniem głosu w aplikacji mobilnej. Przy pomocy platformy Expo Go nie jest to możliwe. Aby rozwiązać ten problem zdecydowano się na

implementację obsługi głosowej za pomocą przetwarzania powtarzanych co kilka sekund nagrani. Implementacja rozwiązania ze strumieniem głosu bezpośrednio musiałaby wiązać się z gruntowną przebudową całego środowiska aplikacji mobilnej.

11.2 Zmiany w trakcie realizacji

Proces planowania i projektowania podjętego w projekcie systemu informatycznego wyprzedził o kilka miesięcy implementacje kluczowych funkcjonalności i założeń. Z powodu napotkanych wyzwań oraz nieprzewidzianych problemów doszło do konfrontacji rzeczywistości ze sprecyzowanymi wcześniej założeniami. Z tego powodu w projekcie musiało dojść do kilku opisanych w niniejszej sekcji zmian.

11.2.1 Limit logowań

Priorytet wymagania dotyczący limitu logowania został zmniejszony z must na should, ze względu na to, że aplikacja nie przechowuje żadnych danych wrażliwych użytkowników i zamiast tego wymagania został podniesiony priorytet generowania treści fiszek na podstawie słów kluczowych.

11.2.2 Generowanie definicji przez ChatGTP

Opis wymagania dotyczącego generowania treści przez ChatGPT został zmieniony. W pierwotnej wersji ta funkcjonalność miała działać tak, że użytkownik wpisywał w pole kilka słów i kluczowych i na tej podstawie ChatGPT miał zwrócić treść, ale takie podejście doprowadziłoby do generowania losowych treści. Zmieniona funkcjonalność działa tak, że użytkownik wpisuje w fiszek cokolwiek by chciał następnie aby wygenerować odpowiedź od ChatGPT klika przycisk generuj i treść wpisana w fiszkę jest wysyłana do chatu i ten na podstawie otrzymanej wiadomości zwraca odpowiedzi w postaci wygenerowanego tekstu odnoszącego się do zawartości fiszki.

11.2.3 System operacyjny w infrastrukturze serwerowej

W trakcie implementacji infrastruktury serwerowej i przenoszeniem projektu do chmury zmieniono wymagania dotyczące docelowego systemu operacyjnego. W fazie planowania wybrano system CentOS 8. W późnej fazie projektu przy wdrażaniu wirtualnej maszyny na platformie chmurowej okazało się, że Azure proponuje jedynie starsze wersje CentOS 7 których wsparcie kończy się w czerwcu 2024. Z tego powodu zdecydowano się wybrać system Ubuntu 20.04 z możliwością podniesienia wersji do Ubuntu 22.04 ze znacznie dłuższym okresem wsparcia.

11.3 Porzucone pomysły

Zalożenie dużego projektu informatycznego wygenerowało wiele pomysłów i rozwiązań. Niektóre nie znalazły finalnego zastosowania. Te zostały opisane w niniejszej sekcji.

11.3.1 Dyktowanie treści fiszki

Zespół rozważał możliwość dodania funkcji dyktowanie treści fiszki przy użyciu mowy. Użytkownik po kliknięciu ikony mikrofonu miałby możliwość dyktowania treści fiszki. Zespół zrezygnował z tego pomysłu ze względu na problemy z poprawnym przekonwertowaniem mowy na tekst. Utworzony tekst nie zawsze odpowiadał temu, co mówi użytkownik.

11.3.2 Sterowanie głosowe przez ChatGPT

Przy pierwszym podejściu do sterowania talii przy użyciu komend głosowych został wykorzystany ChatGPT. Po przetworzeniu na tekstu słów wypowiadanych przez użytkownika, treść była wysyłana do ChatGPT. ChatGPT oprócz wypowiadanych przez użytkownika słów otrzymywał również zestaw komend, na podstawie otrzymanych informacji miał za zadanie zwrócić komendę, która najbardziej pasuje semantycznie do wypowiadanych przez użytkownika słów. Na podstawie otrzymanej informacji zwrotnej aplikacja wykonywała określona akcję. Przy kilku pierwszych wysłanych requestach to podejście się sprawdzało, niestety jeżeli użytkownik w zbyt krótkim czasie wykonał za kilka komend głosowych, to request wysyłane do ChatGPT były blokowane i zwracany był błąd statusu "error 500". To ograniczenie zmusiło nas do zrezygnowania z używania czatu do sterowania głosowego.

11.3.3 Aktywacja konta i reset hasła

W końcowej fazie projektu zrezygnowaliśmy również z aktywacji konta i resetu hasła. Funkcjonalności te zostały pozostawione na koniec i choć ich implementacja nie powinna stanowić wyzwania, częściowo były nawet zaimplementowane w projekcie, uznaliśmy o ich odrzuceniu z uwagi na nagły czas i chęć skupienia na dopracowaniu istotniejszych funkcjonalności

11.4 Przyszłość projektu

Aktualnie nie przewiduje się zorganizowanej kommercializacji projektu. Pozostanie on jednak dalej utrzymywany w oparciu o inne, bardziej korzystne rozwiązania chmurowe lub lokalną infrastrukturę serwerową. Plan kommercializacji i rozwoju systemu Fishki jest uzależniony od zdobycia bazy użytkowników oraz ich feedbacku. W przypadku zmiany decyzji, aplikacja zostanie rozbudowana o nowe funkcjonalności niezbędne do funkcjonowania w środowisku produkcyjnym. Zostanie dodana możliwość zresetowania hasła w przypadku gdy użytkownik zapomni hasła do logowania. Dodanie achievementów związanych z tworzeniem, udostępnianiem i ilością pobranych talii, jest to kolejna mechanika, która ma na celu zwiększenie aktywności użytkowników. Następnym rozpatrywanym ulepszeniem aplikacji jest dodanie narzędzi sztucznej inteligencji, które pozwolą na generowanie talii fiszek z dokumentów przesyłanych przez użytkownika. Użytkownik będzie miał możliwość wrzucenia dokumentu w formacie pdf, csv lub xml i na podstawie zawartości dokumentu zostanie utworzona talia fiszek. Biorąc pod uwagę rozwiązania kommercializacji w konkurencyjnych systemach, najlepszym rozważanym sposobem monetyzacji, który nie zniechęci użytkowników powinien działać w oparciu o dobrowolne darowizny.

11.5 Zakończenie

Niniejszy projekt udało się wypracować zgodnie z założonymi wymaganiami i wizją, która pozostała w dużym stopniu niezmienna. Podjęcie się wytworzenia systemu o tak wysokiej złożoności było dla nas wszystkich wyzwaniem z jakim nie mieliśmy do czynienia wcześniej. Pomimo wielu wyzwań oraz ograniczeń, udało się zrealizować Fishki, aplikację oferującą unikatowe funkcjonalności dla tego typu rozwiązania.

Bibliografia

- [1] Green C. *Classics in the History of Psychology*. Dostęp: 11 maja 2024, godz. 7:00. Classics in the History of Psychology. 1885. URL: <https://psychclassics.yorku.ca/Ebbinghaus/memory5.htm> (term. wiz. 11.05.2024).
- [2] Wikipedia. *Grywalizacja*. Dostęp: 11 maja 2024, godz. 8:40. 2024. URL: <https://pl.wikipedia.org/wiki/Grywalizacja> (term. wiz. 11.05.2024).
- [3] Redakcja Ably. *Gamification: How to Increase Active Users (MAU and DAU)*. Dostęp: 11 maja 2024, godz. 8:30. 2024. URL: <https://ably.com/blog/how-to-increase-active-users> (term. wiz. 11.05.2024).
- [4] R Bartosz. *Kilka ryzyk związanych z korzystaniem z ChatGPT*. Dostęp: 10 czerwca 2024, godz. 16:40. 2023. URL: <https://dfe.org.pl/ryzyka-chatgpt/> (term. wiz. 10.06.2024).
- [5] Roger P. *Praktyczne podejście do inżynierii oprogramowania*. Warszawa, 2004.
- [6] Osborn A. *Applied Imagination: Principles and Procedures of Creative Problem-Solving*. New York: Scribner, 1953.
- [7] Pei J Han J Kamber M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, an imprint of Elsevier, 2012.
- [8] Mielnik T. *Testy integracyjne*. Dostęp: 31 maja 2024, godz. 14:00. 2024. URL: <https://testerzy.pl/baza-wiedzy/artykuly/testy-integracyjne> (term. wiz. 31.05.2024).
- [9] Oliveira B. *pytest Quick Start Guide*. Birmingham, 2018.
- [10] Redakcja Testerzy.pl. *Testowanie akceptacyjne*. Dostęp: 31 maja 2024, godz. 15:00. 2024. URL: <https://testerzy.pl/baza-wiedzy/testowanie-akceptacyjne> (term. wiz. 31.05.2024).