Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: IST

ZESPOŁOWE PRZEDSIĘWZIĘCIE INFORMATYCZNE

Opracowanie aplikacji mobilnej wspierającej osoby starsze i ich opiekunów

Tomasz Chojnacki
Kamil Ciągło
Jakub Czerwiński
Jakub Zehner

Opiekun pracy
dr inż. Marcin Jodłowiec

Słowa kluczowe: aplikacja mobilna, React Native, ASP.NET, osoby starsze, dostępność

WROCŁAW 2023

Spis treści

DOKUMENTACJA PROJEKTOWA

1. Cel i zakres przedsięwzięcia

Celem pracy jest opracowanie systemu informatycznego składającego się z aplikacji mobilnej oraz usług uruchamianych na serwerze, które przyczynią się do pomocy osobom starszym w codziennych czynnościach, a także zwiększenia bezpieczeństwa ich funkcjonowania. Postawiono ten cel, gdyż społeczeństwo się starzeje i seniorzy coraz częściej korzystają ze smartfonów. W niedługim czasie większość seniorów będzie korzystała z nich na co dzień. Osoby starsze mają też swoje potrzeby, które nie są adresowane przez dużą liczbę aplikacji. Wymienione wyżej fakty sugerują, iż w kolejnych latach zapotrzebowanie na aplikacje dla osób starszych wzrośnie, a niewielka liczba konkurencyjnych aplikacji oznacza szansę na zajęcie części rynku już teraz, zanim zacznie pojawiać się więcej podobnych rozwiązań.

System składać się będzie z szeregu modułów wspierających codzienne aktywności seniorów oraz osób opiekującymi się nimi. Kluczową funkcjonalnością będzie możliwość zarządzania lekami i ich przyjęciami. Seniorzy będą mogli zapisywać, kiedy powinni brać dane leki oraz zaznaczać ich przyjęcia. System będzie również oferował seniorom możliwość zagrania w gry stymulujące umysł w celu zmniejszenia ryzyka wystąpienia chorób związanych z pamięcią. Kolejnym aspektem będzie możliwość tworzenia i odsłuchiwania notatek, aby ułatwić ich prowadzenie i wyeliminować potrzebę szukania kartek papieru, gdy senior chce coś szybko zapisać.

Ważnym aspektem projektu będą kwestie bezpieczeństwa. Łączenie seniora z opiekunem będzie musiało oferować odpowiedni poziom bezpieczeństwa ze względu na wrażliwość danych, do których opiekun będzie miał dostęp. Opiekunowie będą też powiadamiani w przypadkach kryzysowych (np. gdy senior nie weźmie leku na czas lub wyśle sygnał SOS).

Dużą wagę w projekcie będą miały zagadnienia dostępności (ang. *accessibility*) oraz użyteczności (ang. *usability*) ze względu na potrzeby docelowych odbiorców. Na każdym etapie projektu kluczowe powinno być zapewnienie wysokiej jakości tych dwóch aspektów.

Projekt ma charakter implementacyjny, w związku z tym wytworzone artefakty będą stanowić kody źródłowe aplikacji mobilnej oraz usług uruchamianych na serwerze. Dodatkowo wykonana zostanie dokumentacja projektowa aplikacji. Zawierać ona będzie wszelkie decyzje projektowe, zaczynając od analizy istniejących, konkurencyjnych rozwiązań, przez założenia wstępne i pozyskiwanie wymagań, aż po projekt produktu programowego. Opisane również zostaną najciekawsze części implementacji, przeprowadzone testy na grupie docelowych odbiorców w celu weryfikacji wymagań oraz dokumentacja użytkownika.

2. Słownik pojęć

Na potrzeby przedsięwzięcia został utworzony uporządkowany alfabetycznie słownik pojęć, zawierający definicje najistotniejszych zagadnień związanych z projektem. Część terminów zawiera angielskie tłumaczenia, które są wykorzystywane w zawartych w pracy listingach i niektórych diagramach.

- **Administrator (admin)** – *rola* użytkownika systemu, która pozwala na wykonanie dowolnej operacji w systemie.
- **Alert** – powiadomienie wysyłane na urządzenie mobilne użytkownika w sytuacji kryzysowej, m.in. w przypadku nieprzyjęcia leku przez seniora.
- **CRON** – popularny system harmonogramowania zadań w systemach uniksowych.
- **Członek (member)** – *rola* użytkownika systemu, która daje dostęp do wszystkich standardowych funkcji systemu.
- **Gadżet (gadget)** – skrót do jednego z ekranów aplikacji lub często wykonywanej czynności, który może być umieszczony jako *kafelek* w *panelu*.

- **Gra (game)** – dostępna w aplikacji mobilnej aktywność rozrywkowa, która ma na celu utrzymanie sprawności umysłowej seniora.
- **Kafelek (dashboard item)** – element interfejsu *panelu*, uruchamiający akcję *gadżetu*.
- **Konto (account)** – zbiór informacji identyfikujących konkretnego użytkownika systemu, w tym w szczególności jego dane logowania oraz dane kontaktowe.
- **Lek (medication)** – substancja lecznicza zażywana regularnie przez *seniora*.
- **Memory** – jedna z *gier* dostępnych w aplikacji mobilnej, polegająca na odkrywaniu pasujących do siebie kart z obrazkami.
- **Menu** – domyślny widok aplikacji mobilnej dla *opiekuna*, zawierający listę wszystkich dostępnych funkcji systemu.
- **Notatka (note)** – zapisana w systemie wiadomość tekstowa, która może być odczytana przez *seniora* i jego *opiekunów* (notatka publiczna) lub wyłącznie *seniora* (notatka prywatna).
- **Opiekun (caretaker)** – *profil*, którym posługuje się *opiekun seniora* (najczęściej będący członkiem rodziny), który zezwala na nadzór nad *seniorem*.
- **Panel (dashboard)** – domyślny widok aplikacji mobilnej dla *seniora*, zawierający zbiór konfigurowalnych skrótów do najczęściej używanych funkcji, zwanych *kafelkami*.
- **Parowanie (pairing)** – proces łączenia *profilu seniora* ze znajdującym się na innym *koncie profilem opiekuna* poprzez skanowanie kodu QR.
- **Profil (profile)** – *senior* lub *opiekun*, konfiguracja *konta* wybierana po zalogowaniu, która determinuje widoczne dla użytkownika funkcje systemu.
- **Przyjęcie (intake)** – zapisany w systemie fakt spożycia *leku* przez *seniora*.
- **Przypomnienie (reminder)** – zapisany w systemie alarm, który przypomina *seniorowi* o konieczności przyjęcia danego *leku*.
- **Rola (role)** – *administrator* lub *członek*, określa uprawnienia danego *konta*.
- **Senior** – *profil*, którym posługuje się osoba starsza, zezwalający na dostęp do większości modułów aplikacji i umożliwiający *parowanie* z *opiekunami*.
- **Senso** – nazwa kodowa systemu widoczna na ekranie startowym aplikacji mobilnej.
- **Sudoku** – jedna z *gier* dostępnych w aplikacji mobilnej, polegająca na uzupełnianiu brakujących cyfr w siatce, tak aby w każdym wierszu, kolumnie i kwadracie znalazły się wszystkie cyfry od 1 do 9.
- **Sygnał SOS (SOS signal)** – rodzaj *alertu* wysyłanego do wszystkich *opiekunów*, inicjowany poprzez naciśnięcie przez *seniora* przycisku SOS w *panelu*.
- **Tablica wyników (leaderboard)** – ranking graczy tworzony na podstawie osiągów *seniorów* w dostępnych *grach*.
- **Urządzenie (device)** – telefon lub tablet z systemem Android, lub iOS, na którym zainstalowana jest aplikacja mobilna.
- **Wordle** – jedna z *gier* dostępnych w aplikacji mobilnej, polegająca na odgadywaniu słów na podstawie podpowiadanych liter.
- **Wynik (leaderboard entry)** – wpis w *tablicy wyników*, zawierający informacje o osiągnięciach danego *seniora* w danej *grze*.

3. Stan wiedzy w obszarze przedsięwzięcia

Przed przystąpieniem do projektu przeanalizowane zostały istniejące już aplikacje mobilne mające na celu pomoc osobom starszym. Bazując na artykułach przedstawiających najlepsze aplikacje dla seniorów [1, 2] wyodrębnione zostało kilka aplikacji oferujących ciekawe rozwiązania:

- SeniorApp,
- Przypomnienie o lekach,
- BIG Launcher.

3.1. SeniorApp

SeniorApp jest aplikacją, która pomaga znaleźć osoby, które zajmą się opieką nad osobą starszą, lub zaoferują im pomoc w podstawowych czynnościach (sprzątanie, zakupy). Aplikacja posiada minimalistyczny interfejs, zrozumiały dla seniora, oraz prosty sposób sortowania usług i przeglądania ofert. Aplikacja jest darmowa, jednak osoby oferujące usługi podają ich cenę, a całość obsługiwana jest za pomocą bezgotówkowych transakcji, użytkownicy mają „portfel aplikacji", do którego mogą podpiąć konto bankowe. Plusem aplikacji jest to, że pozwala w dość prosty sposób znaleźć osoby, które zaoferują seniorom jakąś pomoc, jednak jest to raczej wyszukiwarka usług, niż coś, co faktycznie może pomóc seniorom w codziennym funkcjonowaniu.

3.2. Przypomnienia o lekach

Przypomnienia o lekach to bezpłatna aplikacja, która przypomina o zażywaniu leków i pomaga w ogólnym monitorowaniu zdrowia. Pozwala ona na tworzenie listy leków wraz z dawkowaniem, monitorowanie stanu zdrowia i przechowywanie listy przyjętych leków. Dodatkowo wysyła ona przypomnienia o wzięciu leku. Plusem aplikacji jest to, że ułatwia ona seniorom codzienne życie i pozwala w prosty sposób monitorować leki i stan zdrowia.

3.3. BIG Launcher

BIG Launcher to aplikacja przeznaczona dla seniorów oraz osób słabowidzących. Zmienia ona wygląd interfejsu tak, aby wszystko było duże i miało zwiększony kontrast. Wszystko może być w dostosowane do użytkownika, więc senior może zmienić wygląd interfejsu, jeśli będzie tego chciał. Plusem aplikacji jest duża wszechstronność i implementowanie faktycznie przydatnych seniorom funkcjonalności. Niestety, aplikacja jest dostępna jedynie na urządzenia z systemem Android, a wersja darmowa zawiera pewne ograniczenia w dostosowywaniu i bezpieczeństwie oraz wyświetlana informację zachęcającą do zakupu pełnej wersji oprogramowania.

3.4. Podsumowanie

Istnieje wiele aplikacji mających na celu pomoc seniorom. Większość z nich dobrze spełnia swoje zadania, jednak ogranicza się do pojedynczych aspektów życia seniorów. Może to być problemem w przypadku, gdy senior potrzebuje pomocy w każdym z nich, zwłaszcza że obsługa wielu aplikacji, każda z innym interfejsem, może być trudna dla osób starszych. Projektowane rozwiązanie powinno rozwiązać ten problem, oferując większą różnorodność pomocy dla seniorów, jednocześnie inspirując się dobrymi praktykami obecnymi w istniejących rozwiązaniach.

4. Założenia wstępne

4.1. Role w aplikacji

W opisywanym systemie powinny znaleźć się dwie główne role: **użytkownik** oraz **administrator (admin)**. Administrator powinien mieć dostęp do całości aplikacji, jedynie osoby pracujące nad aplikacją powinni mieć dostęp do tej roli. Użytkownik powinien być rolą ogólnodostępną i domyślną, oferującą ograniczony dostęp do danych w aplikacji.

W celu dalszego podziału obowiązków zaproponowano wprowadzenie *profili*. Powinny one udostępnić *użytkownikowi* możliwość posiadania różnych uprawnień (w zależności od wybranego profilu) bez potrzeby przelogowywania się na inne konto. W podstawowej wersji aplikacji powinny znaleźć się dwa profile: **seniora** oraz **opiekuna**. Profil seniora umożliwi użytkownikom normalne korzystanie z aplikacji i będzie reprezentował osobę starszą. Profil opiekuna będzie utworzony dopiero po połączeniu z jakimś seniorem i będzie pozwalał na pomaganie i opiekowanie się osobą starszą.

4.2. Wysokopoziomowa lista wymagań

Aplikacja skierowana jest do osób starszych, zatem powinna przede wszystkim odpowiadać na ich potrzeby. Na etapie zbierania wymagań sporządzona została prosta lista funkcjonalności, które powinny być oferowane przez system. Lista ta została później przedstawiona grupie seniorów, którzy ocenili pozycje z listy oraz dodali kilka własnych pomysłów. Na podstawie zebranych informacji została sporządzona wysokopoziomowa lista najważniejszych wymagań:

- Aplikacja powinna być przystępna dla osób starszych.
- Aplikacja powinna ułatwiać życie codzienne seniorów poprzez system zarządzania lekami, czy możliwość tworzenia notatek.
- Aplikacja powinna zapewniać seniorom bezpieczeństwo zarówno w postaci sposobu parowania opiekuna z seniorem, jak i w postaci alertów i powiadomień wysyłanych opiekunom w sytuacjach kryzysowych.
- Aplikacja powinna pomagać seniorom w zachowaniu zdrowego umysłu poprzez udostępnienie prostych gier logicznych.
- Aplikacja powinna być prosta do dostosowania dla konkretnego seniora, nawet przez jego opiekunów.

4.3. Przyjęte ograniczenia

Przyjęte ograniczenia wynikają głównie z ograniczonego czasu na realizację projektu. Część z nich to proponowane seniorom funkcjonalności, które spotkały się z raczej negatywnym odbiorem. Niektóre zostały też odrzucone po przeprowadzonej analizie technologii, gdyż ich realizacja byłaby bardzo trudna i wymagałaby dostosowania aplikacji do konkretnego sprzętu. Finalnie spisane zostały następujące ograniczenia:

- Aplikacja nie będzie umożliwiała śledzenia pozycji seniora (gdyby ten np. oddalił się nieświadomie od miejsca zamieszkania).
- Tablice wyników graczy w grach pokazują jedynie globalny ranking.
- Aplikacja nie będzie posiadała systemu wykrywania upadków, gdyż implementacja byłaby problematyczna i wymagała dostosowania aplikacji pod konkretny sprzęt (np. konkretny model smartwatcha).
- Aplikacja w wersji na system iOS nie będzie mieć powiadomień, gdyż wymaga to założenia płatnego konta deweloperskiego.
- Aplikacja nie będzie oferować funkcji lupy (odpowiednik powiększenia w aparacie), gdyż nie byłoby to wygodne dla seniorów.

- Aplikacja nie będzie udostępniała możliwości integracji z eReceptą.
- Aplikacja nie będzie posiadała funkcjonalności znanych z mediów społecznościowych (znajomi, czaty, grupowe rozmowy).

4.4. Dobór technologii

Aplikacja mobilna – React Native

Do rozwoju aplikacji mobilnej wybrany został React Native wraz z językiem TypeScript. React Native jest zestawem narzędzi, który pozwala tworzyć wieloplatformowe aplikacje mobilne. Jego zastosowanie oznacza, że nie jest potrzebne utrzymywanie dwóch osobnych wersji aplikacji, każda na inną platformę. Kod zostanie napisany w języku TypeScript, gdyż obecność typów pozwala na dużo prostsze przekazywanie danych między serwerem a aplikacją mobilną. Do wspomagania nad aplikacją postanowiono użyć też Expo, czyli platformy, która ułatwia pracę z React Native, między innymi umożliwiając dostęp do aparatu. Wśród rozważanych technologii znalazł się również początkowo Flutter. Mimo wielu zalet nie został on wybrany, głównie z powodu braku doświadczenia z tą technologią u członków zespołu, oraz artykułów zalecających użycie Reacta w nowych projektach [3].

Serwer – ASP.NET Core MVC

Po stronie serwera wybraną technologią został ASP.NET Core wraz z językiem C#. Jest to technologia, z którą zespół miał częstą styczność i w której czuje się pewnie. Dodatkowo elementy takie jak Entity Framework Core (EF Core) oraz Language Integrated Query (LINQ) mogą ułatwić pracę z bazą danych. EF Core pozwala korzystać z bazy danych jak ze zwykłej kolekcji i w bardzo prosty sposób tworzyć czy modyfikować bazę edytując kod C# (podejście code-first) [4]. LINQ pozwala w prosty sposób operować na kolekcjach i w połączeniu z EF Core sprawia, że korzystanie z bazy danych po stronie serwerowej staje się niemal trywialne. Do tego wszystkiego .NET jest cały czas wspierany przez Microsoft, więc można liczyć, iż w przyszłości zostaną wprowadzane nowe funkcjonalności czy pomoce, które ułatwią utrzymywanie kodu.

Baza danych – PostgreSQL

Przy wyborze bazy danych pierwszą ważną decyzją jest wybór między bazą relacyjną (SQL) a nierelacyjną (NoSQL). Wybrana została baza relacyjna głównie ze względu na strukturę danych, ale nie tylko [5]. W opisywanym projekcie bardzo ważne są powiązania między odpowiednimi tabelami. Bazy nierelacyjne byłyby lepszym wyborem w przypadku luźno powiązanych danych, jednak tutaj większość danych powinno dać się w prosty sposób połączyć z konkretnym seniorem.

Wśród baz relacyjnych wybrany został PostgreSQL. Jest to przede wszystkim wszechstronna baza danych, która oferuje bardzo dużo funkcji, które mogą ułatwić skalowanie aplikacji. Kolejnym plusem PostgreSQL jest fakt, iż jest on darmowym, otwartoźródłowym rozwiązaniem, przez co pozwala na proste dostosowanie go do swoich potrzeb w razie problemów. Jest to też SZBD, z którym zespół był najlepiej zapoznany i z którym już pracował, co jest bardzo ważne przy tak ograniczonym czasowo projekcie.

System kontroli wersji i zarządzania projektem – Git/GitHub

Istnieje wiele systemów kontroli wersji, jednak wybrany został Git wraz z GitHubem, czyli internetowym serwisem hostingowym przeznaczonym dla projektów wykorzystujących Gita. Jest to system, z którym cały zespół od dłuższego czasu już pracował. Sam Git jest relatywnie prostym systemem, jednak pozwala na przeprowadzanie wszystkich akcji potrzebnych do dostarczenia kodu. GitHub natomiast oferuje bardzo dużo funkcji, które pomagają z innymi aspektami projektu jak zarządzanie nim, czy system ciągłej integracji i ciągłego dostarczania (CI/CD). GitHub Projects [6] pozwala tworzyć listy zadań z podziałem na sprinty i kategorie, tablice Kanban i wykresy potrzebne

do efektywnego zarządzania pracą w projekcie. GitHub Actions [7], czyli platforma CI/CD pozwala między innymi na przeprowadzanie testów i sprawdzanie składni kodu przed umieszczeniem go na głównej gałęzi. Dodatkowo z pomocą tego systemu możliwe jest automatyczne wdrażanie kodu w chmurze, co ułatwi aktualizowanie wersji produkcyjnej aplikacji. To sprawi, że cały projekt będzie mógł znaleźć się w jednym miejscu bez potrzeby przełączania się między innymi aplikacjami.

5. Specyfikacja i analiza wymagań

5.1. Kategoryzacja wymagań

Uzyskana w poprzedniej sekcji wysokopoziomowa lista wymagań została poddana analizie, w wyniku której zostały wyspecyfikowane szczegółowe wymagania funkcjonalne i niefunkcjonalne.

Wszystkie wymagania zostały skategoryzowane za pomocą metody MoSCoW. Jest to popularna metoda, która wyróżnia cztery kategorie wymagań [8]:

- **Must Have (M)** – wymagania, które muszą być spełnione przez aplikację.
- **Should Have (S)** – wymagania, które powinny być spełnione przez aplikację, ale nie są niezbędne do jej działania.
- **Could Have (C)** – wymagania, których pominięcie ma mały wpływ, ale mogą być włączone do projektu bez większego problemu.
- **Won't Have (W)** – wymagania, które nie będą wdrażane w MVP, ale mogą zostać potencjalnie rozpatrzone w przyszłości.

Co więcej, jako źródła wymagań podane zostały podstawowe założenia aplikacji (wymagania zapewniające spełnienie naszej wizji aplikacji), lub użytkownik (senior, opiekun, lub w przypadku braku wyszczególnienia, oboje), czyli dodatkowe wymagania zebrane od seniorów lub ich opiekunów po przedstawieniu im wstępnej wizji aplikacji.

Jeśli chodzi o element aplikacji, którego wymaganie dotyczy, ponownie wyszczególniamy użytkownika (o ile nie pojawił się jako źródło wymagania), oraz bazę danych, aplikację mobilną, usługi serwerowe czy aplikację (rozumianą jako całość aplikacji, gdy spełnienie wymagania leży zarówno po stronie aplikacji mobilnej, jak i usług serwerowych).

Wymagania zostały zawarte w Tab. 1 i 2.

5.2. Wymagania funkcjonalne i niefunkcjonalne

Wymaganie funkcjonalne	MoSCoW	Źródło wymagania	Którego elementu aplikacji dotyczy
Posiadanie profili zarówno dla seniora, jak i jego opiekunów.	M	Podstawowe założenia aplikacji	Użytkownik, baza danych
Parowanie seniora z opiekunem powinno odbywać się przez skanowanie kodu QR wyświetlanego na urządzeniu seniora.	M	Podstawowe założenia aplikacji – bezpieczeństwo	Użytkownik, aplikacja
Możliwość zapisywania notatek.	S	Użytkownik – senior	Baza danych
Możliwość oznaczenia notatki jako prywatnej – brak dostępu do notatki dla opiekunów.	S	Użytkownik – senior	Usługi serwerowe

Wymaganie funkcjonalne	MoSCoW	Źródło wymagania	Którego elementu aplikacji dotyczy
Możliwość odczytywania notatek.	S	Użytkownik	Aplikacja
Możliwość odsłuchania notatek (text-to-speech).	C	Użytkownik – senior	Aplikacja mobilna
Możliwość zapisania notatek z użyciem głosu (speech-to-text, opcja wbudowana w większość klawiatur).	W	Użytkownik – senior	Aplikacja mobilna
Możliwość zapisywania dawkowania leków.	M	Podstawowe założenia aplikacji	Użytkownik, baza danych
Możliwość zapisywania przyjęć danych leków.	M	Podstawowe założenia aplikacji	Użytkownik – senior baza danych
Możliwość odczytywania listy leków.	M	Użytkownik	Aplikacja
Możliwość sprawdzenia historii przyjęć leków.	M	Użytkownik	Aplikacja
Możliwość aktualizowania dawkowania leków – zmiana przyjmowanej ilości lub częstotliwości.	S	Użytkownik	Aplikacja
Możliwość przeglądania historii przyjęć konkretnego leku.	M	Użytkownik	Aplikacja
Aplikacja powinna wysyłać powiadomienia z przypomnieniem o wzięciu leku do seniora.	S	Użytkownik – senior	Usługi serwerowe
Aplikacja powinna wysyłać powiadomienia z informacją o tym, że senior nie wziął leku do jego opiekunów.	S	Użytkownik – opiekun	Usługi serwerowe
Aplikacja powinna umożliwić seniorom wysłanie sygnału SOS do swoich opiekunów.	M	Podstawowe założenia aplikacji – bezpieczeństwo	Aplikacja
Wszystkie powiadomienia i wysłane sygnały SOS powinny zostać zapisane w bazie danych.	M	Podstawowe założenia aplikacji	Baza danych
Użytkownicy powinni mieć możliwość przeglądania historii alertów.	S	Użytkownik	Aplikacja
W aplikacji powinny znaleźć się proste gry logiczne.	S	Użytkownik – senior	Aplikacja mobilna
Najlepsze wyniki w grach powinny zostać zapisywane w celu tworzenia rankingu graczy.	S	Użytkownik	Baza danych
Opiekunowie powinni mieć dostęp do wszystkich publicznych notatek swojego seniora.	S	Użytkownik – opiekun	Usługi serwerowe

Wymaganie funkcjonalne	MoSCoW	Źródło wymagania	Którego elementu aplikacji dotyczy
Opiekunowie powinni mieć dostęp do listy leków swojego seniora.	M	Podstawowe założenia aplikacji	Użytkownik – opiekun, usługi serwerowe
Opiekunowie powinni móc zarządzać (dodawać, edytować, usuwać) lekami swojego seniora.	M	Podstawowe założenia aplikacji	Użytkownik, usługi serwerowe
Opiekunowie powinni mieć dostęp do historii przyjęć leków swojego seniora.	M	Podstawowe założenia aplikacji	Użytkownik – opiekun, usługi serwerowe
Opiekunowie powinni mieć możliwość edytowania panelu swojego seniora.	M	Podstawowe założenia aplikacji	Użytkownik, aplikacja

Tabela 1: Lista wymagań funkcjonalnych.

Wymaganie niefunkcjonalne	MoSCoW	Źródło wymagania	Którego elementu aplikacji dotyczy
Aplikacja powinna dostosowywać się do wybranej przez użytkownika wielkości czcionki.	M	Podstawowe założenia aplikacji – przystępność	Użytkownik, aplikacja mobilna
Grafiki w aplikacji powinny być proste i intuicyjne dla osób starszych.	M	Podstawowe założenia aplikacji – przystępność	Użytkownik, aplikacja mobilna
Warstwa graficzna aplikacji powinna posiadać duży kontrast i ograniczoną paletę kolorów.	M	Podstawowe założenia aplikacji – przystępność	Użytkownik, aplikacja mobilna
Możliwość dostosowania motywu aplikacji (jasny/ciemny motyw), domyślnie wybierany motyw jak w systemie.	S	Podstawowe założenia aplikacji – przystępność	Użytkownik, aplikacja mobilna
Aplikacja dostępna w dwóch wersjach językowych – polskiej i angielskiej.	M	Chęć zwiększenia zasięgów aplikacji	Użytkownik, aplikacja mobilna
System powinien być dostępny przez cały czas (24/7).	M	Podstawowe założenia aplikacji	Usługi serwerowe
Aplikacja będzie dostępna na telefony z systemem iOS oraz Android.	M	Podstawowe założenia aplikacji	Aplikacja mobilna
Domyślnym widokiem dla seniora jest widok panelu (widok prosty, duże kafelki).	M	Podstawowe założenia aplikacji – przystępność	Użytkownik, aplikacja mobilna
Domyślnym widokiem dla opiekuna jest widok menu (widok listy).	S	Użytkownik – opiekun	Aplikacja mobilna

Wymaganie niefunkcjonalne	MoSCoW	Źródło wymagania	Którego elementu aplikacji dotyczy
Przycisk SOS powinien być cały czas widoczny na panelu i wyróżniać się niezależnie od motywu.	M	Podstawowe założenia aplikacji – bezpieczeństwo	Aplikacja
Po naciśnięciu przycisku SOS powinno się wykonać prostą czynność, aby nie uruchomić go przypadkiem.	M	Podstawowe założenia aplikacji – bezpieczeństwo	Aplikacja

Tabela 2: Lista wymagań niefunkcjonalnych.

5.3. Reguły biznesowe

Bazując na przestawionych powyżej wymaganiach oraz wiedzy dziedzinowej, zostały wyszczególnione następujące reguły biznesowe, podzielone według modelowanych obiektów:

Konto

- **REG/001** – Może istnieć wiele kont.
- **REG/002** – Konto musi posiadać adres e-mail.
- **REG/003** – Adres e-mail konta musi być unikalny.
- **REG/004** – Konto musi posiadać hasło.
- **REG/005** – Konto musi mieć nazwę wyświetlaną.
- **REG/006** – Konto może posiadać numer telefonu.
- **REG/007** – Konto musi być powiązane z dokładnie jedną rolą.
- **REG/008** – Konto może być właścicielem maksymalnie jednego profilu seniora.
- **REG/009** – Konto może być właścicielem wielu profili opiekunów.
- **REG/010** – Konto może być zarządzane przez wiele obcych profili opiekunów.
- **REG/011** – Konto może mieć powiązane wiele notatek.
- **REG/012** – Konto może mieć powiązane wiele kafelków.
- **REG/013** – Konto może mieć maksymalnie sześć powiązanych kafelków.
- **REG/014** – Konto może mieć powiązane wiele przypomnień.
- **REG/015** – Konto może mieć powiązane wiele wyników.
- **REG/016** – Konto może mieć maksymalnie jeden wynik dla konkretnej gry.
- **REG/017** – Konto może mieć powiązane wiele aktywowanych alertów.
- **REG/018** – Konto może mieć maksymalnie jedno powiązane urządzenie.

Rola

- **REG/019** – Istnieją dwie role: „administrator” i „członek”.

Profil

- **REG/020** – Może istnieć wiele profili.
- **REG/021** – Profil może posiadać alias ustawiony przez użytkownika.
- **REG/022** – W przypadku niepodania aliasu przyjmuje on wartość nazwy konta seniora.
- **REG/023** – Profil musi mieć dokładnie jedno konto właściciela.
- **REG/024** – Profil musi mieć dokładnie jedno zarządzane przez siebie konto.
- **REG/025** – Kombinacja zarządzanego konta i konta właściciela profilu musi być unikalna.
- **REG/026** – Zarządzane przez profil konto może być różne od konta właściciela profilu.
- **REG/027** – Jeżeli konto właściciela i zarządzane są takie same, to profil ma typ „senior".
- **REG/028** – Jeżeli konto właściciela i zarządzane są różne, to profil ma typ „opiekun".

Notatka

- **REG/029** – Może istnieć wiele notatek.
- **REG/030** – Notatka musi posiadać treść.
- **REG/031** – Notatka musi posiadać datę utworzenia.
- **REG/032** – Notatka musi być „publiczna" lub „prywatna".
- **REG/033** – Notatka może mieć tytuł.
- **REG/034** – W przypadku braku tytułu jest on określany jako początkowy fragment treści.
- **REG/035** – Notatka musi być powiązana z dokładnie jednym kontem.

Kafelek

- **REG/036** – Może istnieć wiele kafelków.
- **REG/037** – Kafelek musi posiadać pozycję na panelu.
- **REG/038** – Pozycja musi być z przedziału od 1 do 6.
- **REG/039** – Kafelek musi być powiązany z dokładnie jednym kontem.
- **REG/040** – Kafelek musi być powiązany z dokładnie jednym gadżetem.
- **REG/041** – Kombinacja konta i rodzaju gadżetu musi być unikalna.

Gadżet

- **REG/042** – Istnieje predefiniowana, stała lista gadżetów.

Lek

- **REG/043** – Może istnieć wiele leków.
- **REG/044** – Lek musi mieć nazwę.
- **REG/045** – Lek może posiadać ilość w opakowaniu.
- **REG/046** – Jeżeli ilość w opakowaniu jest podana, to musi być liczbą dodatnią.
- **REG/047** – Lek może posiadać jednostkę (np. mg, ml).
- **REG/048** – W przypadku niepodania jednostki jest ona określana jako „tabletki".
- **REG/049** – Lek może być powiązany z wieloma przypomnieniami.

Przypomnienie

- **REG/050** – Może istnieć wiele przypomnień.
- **REG/051** – Przypomnienie musi być „aktywne” lub „nieaktywne”.
- **REG/052** – Przypomnienie musi określać ilość leku do przyjęcia.
- **REG/053** – Ilość leku do przyjęcia musi być liczbą dodatnią.
- **REG/054** – Przypomnienie może określać ilość posiadanego leku.
- **REG/055** – Jeżeli ilość posiadana jest podana, to musi być liczbą dodatnią.
- **REG/056** – Przypomnienie może posiadać częstotliwość przyjmowania (w formacie CRON).
- **REG/057** – Przypomnienie może posiadać opis.
- **REG/058** – Przypomnienie musi być powiązane z dokładnie jednym kontem.
- **REG/059** – Przypomnienie musi być powiązane z dokładnie jednym lekiem.
- **REG/060** – Przypomnienie może być powiązane z wieloma przyjęciami.

Przyjęcie

- **REG/061** – Może istnieć wiele przyjęć.
- **REG/062** – Przyjęcie musi posiadać datę i godzinę spożycia.
- **REG/063** – Przyjęcie musi określać ilość przyjętego leku.
- **REG/064** – Ilość przyjętego leku musi być liczbą dodatnią.
- **REG/065** – Przyjęcie musi być powiązane z dokładnie jednym przypomnieniem.

Wynik

- **REG/066** – Może istnieć wiele wyników.
- **REG/067** – Wynik musi mieć wartość punktową.
- **REG/068** – Wartość punktowa musi być dodatnią liczbą całkowitą.
- **REG/069** – Wynik musi być powiązany z dokładnie jednym kontem.
- **REG/070** – Wynik musi być powiązany z dokładnie jedną grą.
- **REG/071** – Kombinacja konta i gry musi być unikalna.

Gra

- **REG/072** – Istnieją trzy gry: „Wordle”, „Sudoku” i „Memory”.

Alert

- **REG/073** – Może istnieć wiele alertów.
- **REG/074** – Alert musi posiadać datę i godzinę aktywacji.
- **REG/075** – Alert musi być powiązany z dokładnie jednym kontem.
- **REG/076** – Alert musi być powiązany z dokładnie jednym typem alertu.

Typ alertu

- **REG/077** – Istnieją trzy typy alertów: „SOS”, „lek do przyjęcia” i „zapomniano o leku”.

Urządzenie

- **REG/078** – Może istnieć wiele urządzeń.
- **REG/079** – Urządzenie musi posiadać token identyfikujący.
- **REG/080** – Urządzenie musi posiadać datę podłączenia do konta.
- **REG/081** – Urządzenie musi być powiązane z dokładnie jednym kontem.
- **REG/082** – Urządzenie musi być powiązane z dokładnie jednym typem urządzenia.

Typ urządzenia

- **REG/083** – Istnieją dwa typy urządzeń: „Android” i „iOS”.

5.4. Przypadki użycia

Na bazie przedstawionych wymagań szczegółowych, zostały utworzone przypadki użycia przedstawione poniżej. Każdy przypadek użycia został zaklasyfikowany do jednego z sześciu modułów systemu:

- **Alerty i powiadomienia**,
- **Gry**,
- **Konta i profile**,
- **Leki**,
- **Notatki**,
- **Panel**.

Alerty i powiadomienia

Diagram przypadków użycia – alerty i powiadomienia został przedstawiony na rysunku 1. Przedstawia on moduł systemu mający na celu informowanie opiekunów o zdarzeniach związanych z seniorem, a także przypominanie seniorowi o przyjmowaniu leków.

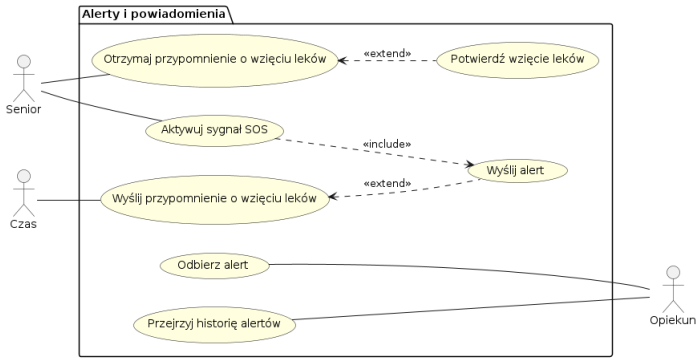
Aktorzy modułu alertów i powiadomień:

- **Senior** – użytkownik systemu będący osobą starszą.
- **Opiekun** – użytkownik systemu będący opiekunem osoby starszej.
- **Czas** – aktor zewnętrzny, który jest odpowiedzialny za dostarczanie informacji o czasie i wyzwolenie części alertów.

Przypadki użycia modułu alertów i powiadomień:

- **Otrzymaj przypomnienie o wzięciu leków** – senior w systemie może otrzymać powiadomienie push, które poinformuje go o konieczności przyjęcia leku.
- **Potwierdź wzięcie leków** – senior po otrzymaniu powiadomienia o konieczności przyjęcia leku, może zaznaczyć w aplikacji, że lek został przyjęty, co spowoduje potwierdzenie przyjęcia leku w systemie, a co za tym idzie niepowiadomienie opiekunów o nieprzyjęciu leku.
- **Aktywuj sygnał SOS** – senior w systemie może aktywować sygnał SOS, który spowoduje wysłanie powiadomienia do systemu o potrzebie pomocy.
- **Wyślij przypomnienie o wzięciu leków** – system może wysłać powiadomienie push do seniora, które poinformuje go o konieczności przyjęcia leku.

- **Wyślij alert** – system może wysłać powiadomienie push do opiekunów, które poinformuje ich o zdarzeniu związanym z seniorem. Przypadek użycia jest zawsze wywoływany przez aktywację sygnału SOS. A także może zostać wywołany przez niepotwierdzenie przyjęcia leku przez seniora.
- **Odbierz alert** – opiekun w systemie może otrzymać powiadomienie push, które poinformuje go o zdarzeniu związanym z seniorem.
- **Przejrzyj historię alertów** – opiekun w systemie może przejrzeć historię alertów, które zostały aktywowane.



Rysunek 1: Diagram przypadków użycia – alerty i powiadomienia.

Gry

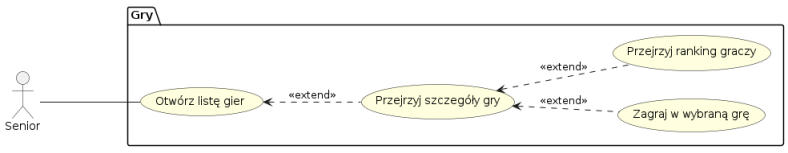
Diagram przypadków użycia – gry został przedstawiony na rysunku 2. Przedstawia on moduł systemu mający na celu dostarczenie seniorom gier logicznych, które pomogą w ćwiczeniu pamięci i koncentracji.

Aktorzy modułu gier:

- **Senior** – użytkownik systemu będący osobą starszą.

Przypadki użycia modułu gier:

- **Otwórz listę gier** – senior w systemie może otworzyć listę dostępnych gier. Z której może wybrać grę, której szczegóły chce zobaczyć.
- **Przejrzyj szczegóły gry** – senior w systemie może przejrzeć szczegóły wybranej gry. Do przypadku użycia można opcjonalnie przejść z przypadku – otwórz listę gier.
- **Przejrzyj ranking graczy** – senior w systemie może przejrzeć ranking graczy, który zawiera najlepsze wyniki w grach w dostępnych grach. Do przypadku użycia można opcjonalnie przejść z przypadku – przejrzyj szczegóły gry.
- **Zagraj w wybraną grę** – senior w systemie może zagrać w wybraną grę. Do przypadku użycia można opcjonalnie przejść z przypadku – przejrzyj szczegóły gry.



Rysunek 2: Diagram przypadków użycia – gry.

Konta i profile

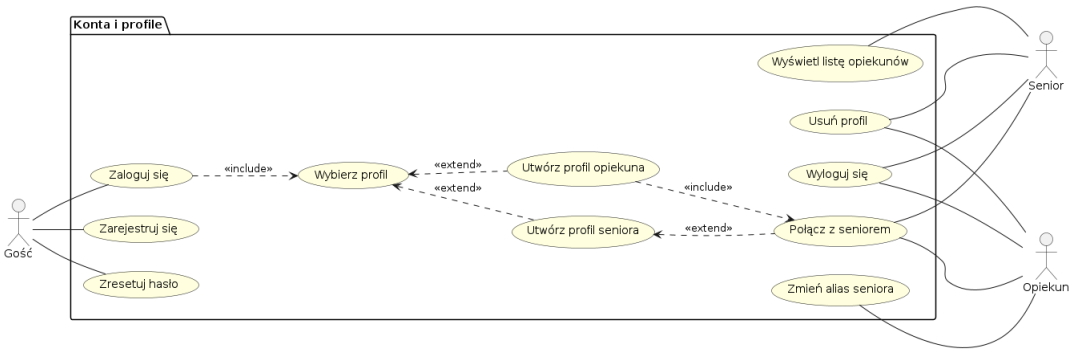
Diagram przypadków użycia – konta i profile został przedstawiony na rysunku 3. Przedstawia on moduł systemu mający na celu zarządzanie kontami i profilami użytkowników. Umożliwia on zarówno tworzenie nowych kont i profilii, jak i zarządzanie istniejącymi.

Aktorzy modułu kont i profili:

- **Senior** – użytkownik systemu będący osobą starszą.
- **Opiekun** – użytkownik systemu będący opiekunem osoby starszej.
- **Gość** – użytkownik systemu, który nie jest zalogowany.

Przypadki użycia modułu kont i profili:

- **Zaloguj się** – gość w systemie może zalogować się do swojego konta.
- **Zarejestruj się** – gość w systemie może zarejestrować się, aby uzyskać dostęp do aplikacji.
- **Zresetuj hasło** – gość w systemie może zresetować hasło do swojego konta.
- **Wybierz profil** – gość w systemie po zalogowaniu może wybrać profil, którym chce się posługiwać. Sprawia to, że zmienia swoją rolę w systemie na seniora lub opiekuna.
- **Utwórz profil opiekuna** – z poziomu wyboru profilu, można utworzyć nowy profil opiekuna, który będzie powiązany z kontem aktualnie zalogowanego użytkownika.
- **Utwórz profil seniora** – z poziomu wyboru profilu, można utworzyć nowy profil seniora, który będzie powiązany z kontem aktualnie zalogowanego użytkownika.
- **Wyświetl listę opiekunów** – senior w systemie może wyświetlić listę opiekunów, którzy są powiązani z jego profilem.
- **Usuń profil** – senior i opiekun w systemie mogą usunąć swój profil. W przypadku seniora istnieje możliwość usunięcia powiązanego z nim profilu opiekuna.
- **Wyloguj się** – senior i opiekun w systemie mogą się wylogować ze swojego konta.
- **Połącz z seniorem** – opiekun w systemie podczas tworzenia nowego profilu musi się połączyć z seniorem, aby móc zarządzać jego profilem. W przypadku seniora jest to opcjonalne, gdyż może on zarządzać swoim profilem samodzielnie.
- **Zmień alias seniora** – opiekun w systemie może zmienić alias seniora, z którym jest powiązany.



Rysunek 3: Diagram przypadków użycia – konta i profile.

Leki

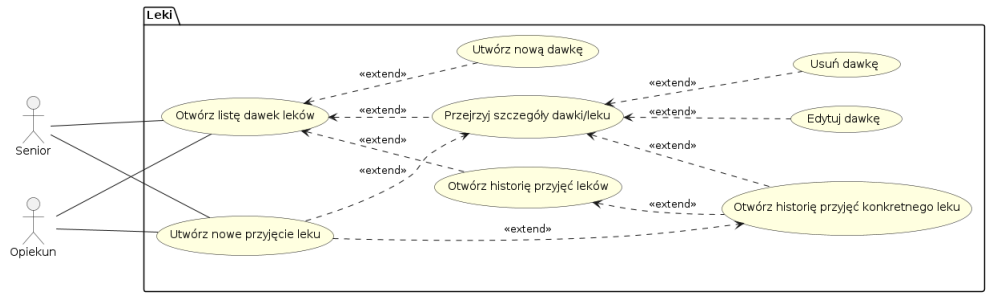
Diagram przypadków użycia – leki został przedstawiony na rysunku 4. Przedstawia on moduł systemu mający na celu zarządzanie lekami, które są przyjmowane przez seniora.

Aktorzy modułu leków:

- **Senior** – użytkownik systemu będący osobą starszą.
- **Opiekun** – użytkownik systemu będący opiekunem osoby starszej.

Przypadki użycia modułu leków:

- **Otwórz listę dawek leków** – senior i opiekun w systemie mogą otworzyć listę dawek leków. Z której mogą wybrać dawkę, której szczegóły chcą zobaczyć, otworzyć historię przyjęć leków lub przejść do tworzenia nowej dawki.
- **Utwórz nowe przyjęcie leku** – senior i opiekun w systemie mogą utworzyć nowe przyjęcie leku. Do przypadku użycia można opcjonalnie przejść z przypadku – przejrzyj szczegóły dawki/leku lub otwórz historię przyjęć konkretnego leku.
- **Utwórz nową dawkę** – senior i opiekun w systemie mogą utworzyć nową dawkę leku. Dawka leku zawiera takie szczegóły jak nazwa leku, ilość leku w opakowaniu, jednostka leku, ilość leku do przyjęcia, godziny przyjęć leku oraz opcjonalny opis.
- **Przejrzyj szczegóły dawki/leku** – senior i opiekun w systemie mogą przejrzeć szczegóły wybranej dawki/leku. Wyświetlane są wszystkie te informacje, które zostały podane podczas tworzenia dawki. Z przypadku użycia można opcjonalnie przejść do usuwania dawki lub edycji dawki, a także otworzyć historię przyjęć konkretnego leku.
- **Otwórz historię przyjęć leków** – senior i opiekun w systemie mogą otworzyć historię przyjęć wszystkich leków. W której wyświetlane są wszystkie przyjęcia leku, które zostały utworzone w systemie. Z przypadku użycia można opcjonalnie przejść do historii przyjęć konkretnego leku.
- **Usuń dawkę** – senior i opiekun w systemie mogą usunąć wybraną dawkę leku. Spowoduje to jej dezaktywację oraz zachowanie całej historii przyjęć leku w tej dawce.
- **Edytuj dawkę** – senior i opiekun w systemie mogą edytować wybraną dawkę leku. Spowoduje to zmianę wszystkich szczegółów dawki leku, które zostały podane podczas jej tworzenia, jednak niemożliwa jest zmiana leku, który jest powiązany z dawką.
- **Otwórz historię przyjęć konkretnego leku** – senior i opiekun w systemie mogą otworzyć historię przyjęć konkretnego leku.



Rysunek 4: Diagram przypadków użycia – leki.

Notatki

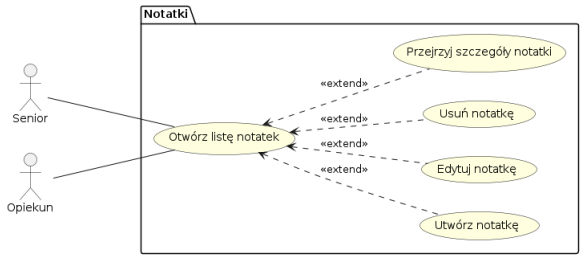
Diagram przypadków użycia – notatki został przedstawiony na rysunku 5. Przedstawia on moduł systemu mający na celu zarządzanie notatkami, które są tworzone przez seniora i mogą być odczytywane przez opiekunów.

Aktorzy modułu notatek:

- **Senior** – użytkownik systemu będący osobą starszą.
- **Opiekun** – użytkownik systemu będący opiekunem osoby starszej.

Przypadki użycia modułu notatek:

- **Otwórz listę notatek** – senior i opiekun w systemie mogą otworzyć listę notatek. Senior może zobaczyć wszystkie swoje notatki, a opiekun może zobaczyć notatki seniora, które nie zostały oznaczone jako prywatne. Z listy notatek można przejść do szczegółów notatki, usuwania notatki, edytowania notatki lub tworzenia nowej notatki.
- **Przejrzyj szczegóły notatki** – senior i opiekun w systemie mogą przejrzeć szczegóły wybranej notatki, czyli zobaczyć lub wysłuchać jej zawartości.
- **Usuń notatkę** – senior i opiekun w systemie mogą usunąć wybraną notatkę. Spowoduje to jej bezpowrotne usunięcie z systemu.
- **Edytuj notatkę** – senior i opiekun w systemie mogą edytować wybraną notatkę. Spowoduje to zmianę jej zawartości.
- **Utwórz notatkę** – senior i opiekun w systemie mogą utworzyć nową notatkę. Notatka może mieć tytuł oraz treść.



Rysunek 5: Diagram przypadków użycia – notatki.

Panel

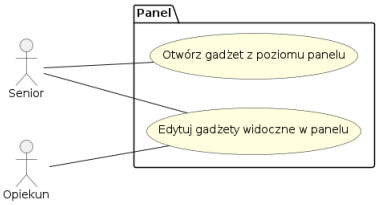
Diagram przypadków użycia – panel został przedstawiony na rysunku 6. Przedstawia on moduł systemu mający na celu zarządzanie kafelkami, które są wyświetlane na panelu głównym

Aktorzy modułu panelu:

- **Senior** – użytkownik systemu będący osobą starszą.
- **Opiekun** – użytkownik systemu będący opiekunem osoby starszej.

Przypadki użycia modułu panelu:

- **Otwórz gadżet z poziomu panelu** – senior w systemie może otworzyć wybrany gadżet z poziomu panelu. Co spowoduje otwarcie przejście do danego modułu aplikacji.
- **Edytuj gadżety widoczne w panelu** – senior i opiekun w systemie mogą edytować wybrane gadżety, które są widoczne w panelu. Zmieniać ich pozycję, usuwać je lub dodawać nowe.



Rysunek 6: Diagram przypadków użycia – panel.

6. Projekt produktu programowego

6.1. Wysokopoziomowy projekt architektury systemu

Jednymi z najpopularniejszych architektur oprogramowania są architektury monolityczne oraz mikroserwisowe [9]. Aplikacja monolityczna to taka, w której różne komponenty są łączone w pojedynczy program uruchamiany na jednej platformie. Natomiast architektura mikroserwisowa zakłada, że system składa się z wielu małych serwisów, realizujących rozłączne części logiki biznesowej.

Obie przedstawione architektury mają swoje wady i zalety. Mikroserwisy są często wybierane ze względu na łatwość utrzymywania i skalowalność. Jednakże, w przypadku prostszych aplikacji, wydajność systemów monolitycznych jest porównywalna. Ponadto, przy tym rozwiązaniu występuje mniej problemów ze wdrożeniem i konfiguracją, a sam rozwój aplikacji jest prostszy [9, 10].

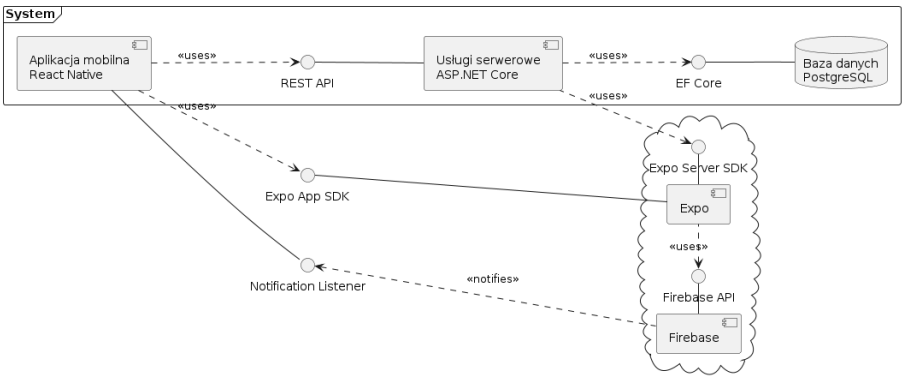
Na podstawie wybranych uprzednio technologii, jak i powyższych rozważań, do implementacji usług serwerowych proponowanego rozwiązania została wybrana architektura monolityczna. Zostały one zrealizowane jako zbiór usług REST API zbudowanych na platformie ASP.NET Core MVC. Na potrzeby rozwoju aplikacji, może ona zostać uruchomiona lokalnie w kontenerze Docker, natomiast w środowisku produkcyjnym, hostowanym na wirtualnym prywatnym serwerze DigitalOcean, uruchamiana jest bez wirtualizacji, za pośrednictwem serwera Kestrel wystawionego na zewnątrz za pośrednictwem serwera proxy nginx.

Trwałość danych zapewniona jest przez pojedynczą instancję bazy danych PostgreSQL, utrzymywaną na tym samym wirtualnym prywatnym serwerze. Łączność z bazą danych zapewnia zestaw pakietów Entity Framework Core, uwzględniany w ramach platformy ASP.NET.

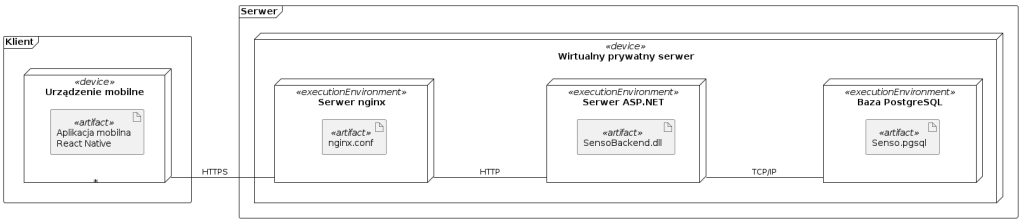
Formę interfejsu między aplikacją mobilną a usługami serwerowymi stanowi wymienione wcześniej REST API, które zostało wyspecyfikowane oraz udokumentowane za pomocą standardu OpenAPI.

Powiadomienia push obsługiwane są przez usługę Expo, która korzysta wewnętrznie z Firebase Cloud Messaging. Zgodnie z przyjętymi ograniczeniami, powiadomienia są wysyłane tylko na urządzenia Android. Warstwa serwerowa wysyła żądania do API Expo poprzez specjalną bibliotekę, gdy następuje potrzeba powiadomienia użytkownika. Na telefonie użytkownika usługa Google Play Services oczekuje powiadomień z serwera Firebase i wyświetla je w postaci notyfikacji systemowych po otrzymaniu.

Aspekt fizyczny oraz logiczny opisanej powyżej wysokopoziomowej architektury został przedstawiony w formie diagramów na Rys. 7 i 8.



Rysunek 7: Uproszczona architektura logiczna systemu.

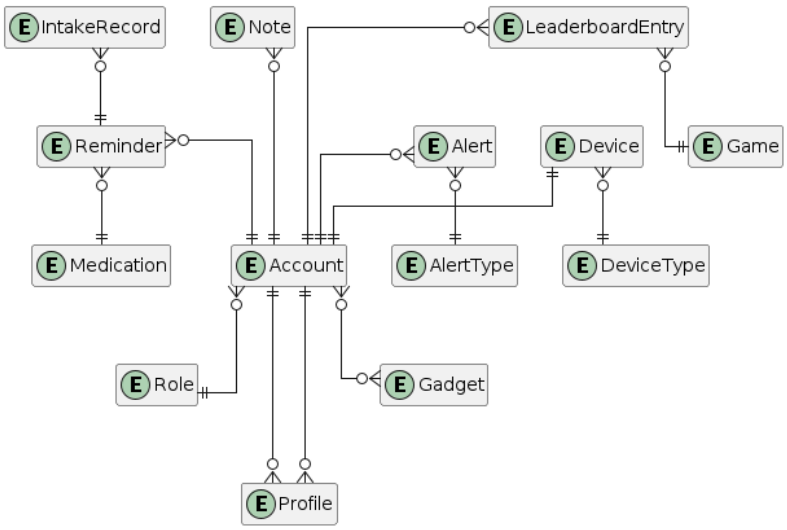


Rysunek 8: Architektura fizyczna systemu.

6.2. Projekt bazy danych

Diagram związków encji systemu powstał bezpośrednio ze zbioru reguł biznesowych określonych w sekcji 5.3. Wykorzystane zostały angielskie nazwy encji, aby przedstawić określenia, które zostaną użyte w implementacji. Tłumaczenia dla poszczególnych encji znajdują się w sekcji 2. „Słownik pojęć". W przypadku, gdy nazwa encji jest złożona z kilku słów, jest ona zapisana w notacji *PascalCase*, co odpowiada szeroko stosowanej konwencji nazewnictwa z języka C#.

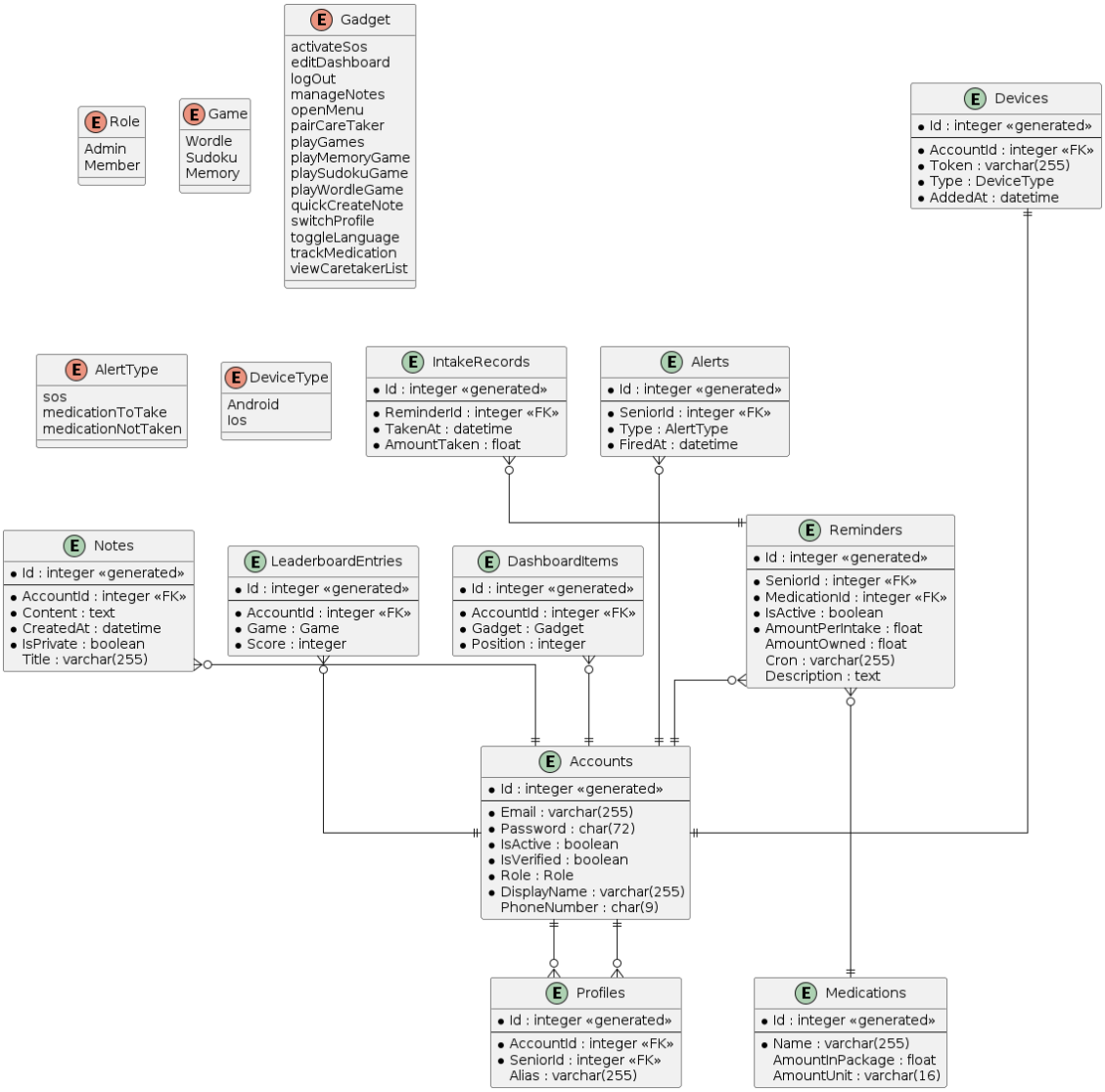
Diagram ERD został przedstawiony na Rys. 9. Symbol dwóch linii ustawionych poprzecznie do połączenia oznacza krotność „dokładnie jeden", natomiast symbol trzech linii z okręgiem oznacza krotność „wiele" (w tym zero).



Rysunek 9: Diagram związków encji.

Utworzony diagram związków encji posłużył do wyspecyfikowania dokładnej struktury bazy danych, tj. tabel oraz ich kolumn. Przedstawione na Rys. 9 encje utworzone na bazie reguł REG/019, REG/042, REG/072, REG/077 oraz REG/083, czyli `Role`, `Gadget`, `Game`, `AlertType` oraz `DeviceType` zostały zareprezentowane jako typy wyliczeniowe, które nie stanowią osobnych tabel w bazie danych.

Do każdej tabeli dodano sztuczny klucz główny `Id`, ułatwiający implementację. Określone zostały klucze obce, typy danych, ograniczenia, indeksy oraz opcjonalność kolumn. Ostateczny projekt bazy danych został przedstawiony na diagramie na Rys. 10.



Rysunek 10: Diagram bazy danych.

Baza danych została wdrożona podejściem *code-first* dostępnym w Entity Framework Core. Polega ono na utworzeniu schematu bazy danych na podstawie zdefiniowanych w kodzie źródłowym klas C#, reprezentujących encje. Na podstawie takich deklaracji, korzystających ze specjalnych konwencji oraz atrybutów, EF jest w stanie nałożyć odpowiednie ograniczenia na kolumny, utworzyć relacje i indeksy, a nawet wygenerować migracje w przypadku zmian w modelu [4].

Przykładowy kod źródłowy modelu został przedstawiony na Listingu 1.

```
1  public class Alert
2  {
3      public required int Id { get; set; }
4      public required AlertType Type { get; set; }
5      public required DateTimeOffset FiredAt { get; set; }
6
7      public required int SeniorId { get; set; }
8      public Account? Senior { get; set; }
9  }
```

Listing 1: Reprezentacja modelu alertu w kodzie.

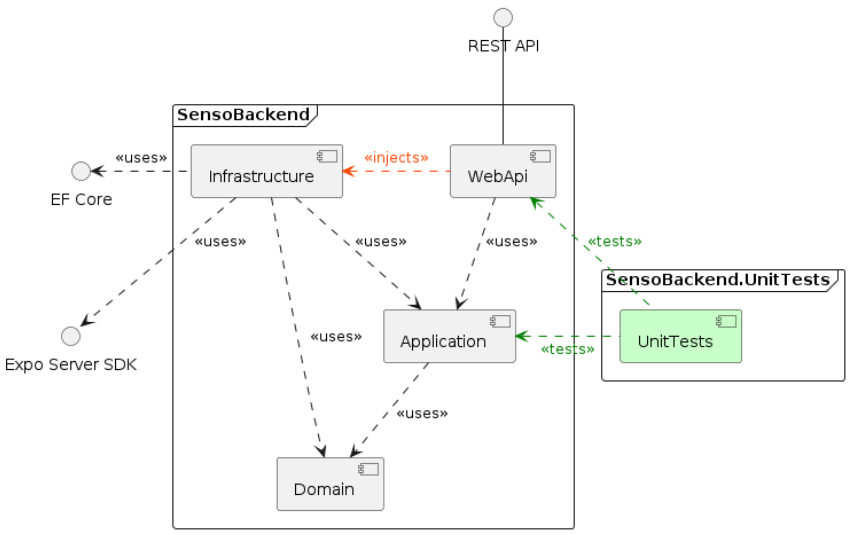
6.3. Projekt architektury usług serwerowych

Istnieje wiele wzorców architektonicznych wykorzystywanych w aplikacjach webowych. Jednym z nich jest architektura warstwowa, która w najpopularniejszym wariancie zakłada podział aplikacji na trzy warstwy: prezentacji, logiki biznesowej oraz dostępu do danych. Komponenty z danej warstwy powinny komunikować się tylko z komponentami warstwy tej samej lub niższej [11].

Zyskującym obecnie na popularności rozwinięciem architektury warstwowej jest *clean architecture*, zwany czasami *onion architecture* [12]. Szczególnie popularna w ekosystemie .NET jest implementacja tego wzorca zaproponowana przez Jasona Taylora w prezentacji „Clean Architecture with ASP.NET Core 3.0" [13]. Oprócz architektury warstwowej, podejście to bazuje na metodyce DDD (Domain Driven Design). Proponowane przez Taylora rozwiązanie, które zostało wykorzystane w prezentowanym w tej pracy systemie, składa się z czterech warstw:

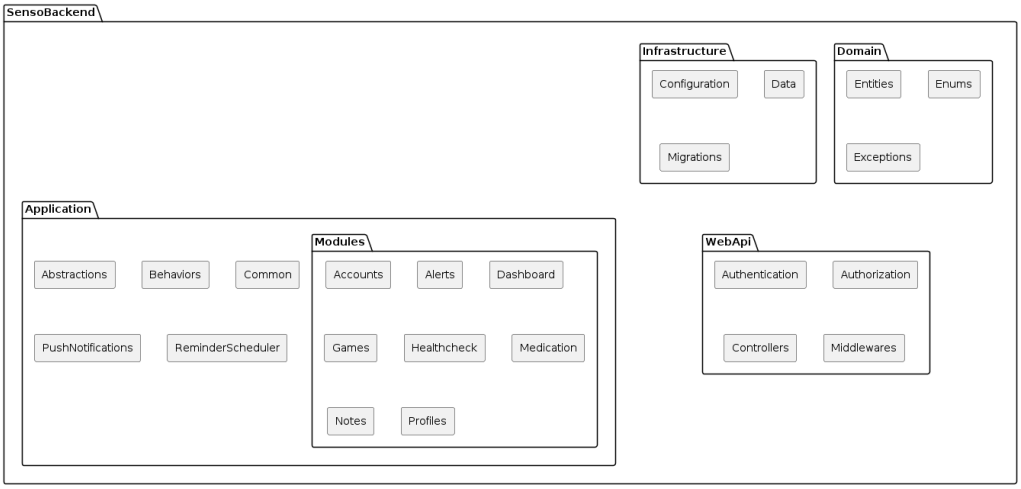
- **prezentacji** – zawiera kontrolery serializujące obiekty do formatu JSON i statusów HTTP;
- **aplikacji** – zawiera logikę biznesową aplikacji, za pomocą wzorca CQRS;
- **dziedzinowa** – zawiera encje, wyliczenia i wyjątki specyficzne dla dziedziny;
- **infrastruktury** – zawiera implementację dostępu do danych, m.in. połączenie z bazą danych.

Taki wzorzec pozwala na dużą niezależność warstw, w tym na łatwą zmianę bazy danych, czy sposobu prezentacji. Opiera się on mocno na zasadzie odwrócenia zależności oraz wzorcu CQRS [12], co pozwala na łatwe testowanie aplikacji oraz uporządkowanie kodu źródłowego zgodnie z metodyką *vertical slicing*. Jest to technika implementacji systemu według konkretnych modułów, a nie warstw [14]. W omawianym systemie pozwoliło to na ukończenie modułu kont i profili, który był kluczowy, bez konieczności dokonywania przygotowań w innych modułach.



Rysunek 11: Realizacja wzorca *clean architecture* w systemie.

Klasy wykorzystane do implementacji usług serwerowych serwera zostały uporządkowane w strukturę katalogów przedstawioną na Rys. 10.



Rysunek 12: Diagram pakietów dla usług serwerowych.

Każdy moduł realizowany w warstwie aplikacji składa się ze zbioru komend i zapytań, będących klasami, które są przekazywane do odpowiednich *handlerów*. Zastosowany jest w tym celu wzorzec projektowy mediator, który odpowiada za odpowiednie przetwarzanie zapytań, w tym na rejestrowanie wspólnych zachowań, takich jak logowanie czy walidacja. Poniżej, na Listingach 2 i 3 znajduje się przykład realizacji jednego z zachowań w systemie.

```
namespace SensoBackend.Application.Modules.Healthcheck.Contracts;

public enum HealthcheckStatus { Ok, Unhealthy }

public sealed record HealthcheckDto
{
    public required HealthcheckStatus Server { get; init; }
    public required HealthcheckStatus Database { get; init; }
}
```

Listing 2: Przykład implementacji zachowania – kontrakty.

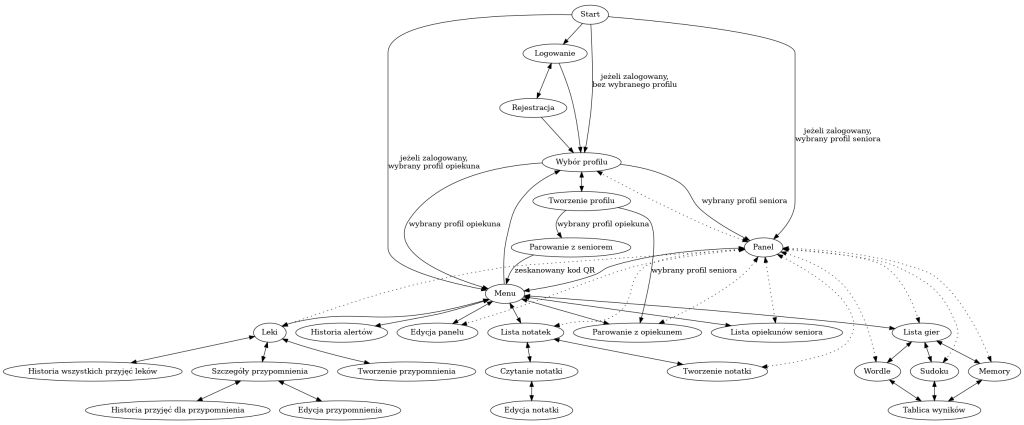
```
namespace SensoBackend.Application.Modules.Healthcheck;

public sealed record GetHealthRequest : IRequest<HealthcheckDto>;

public sealed class GetHealthHandler(AppDbContext context)
    : IRequestHandler<GetHealthRequest, HealthcheckDto>
{
    public async Task<HealthcheckDto> Handle(
        GetHealthRequest request,
        CancellationToken ct) => new HealthcheckDto
            {
                Server = HealthcheckStatus.Ok,
                Database = await context.Database.CanConnectAsync(ct)
                    ? HealthcheckStatus.Ok
                    : HealthcheckStatus.Unhealthy
            };
}
```

Listing 3: Przykład implementacji zachowania – zapytanie i jego obsługa.

Oprócz wzorców projektowych mediator i polecenie, często wykorzystane w usługach serwerowych proponowanego rozwiązania są: adapter, budowniczy, singleton, łańcuch zobowiązań, iterator i strategia. Wraz z zastosowaniem metodyk takich jak SOLID i DRY pozwoliło to na łatwą reużywalność i testowalność kodu źródłowego.

6.4. Projekt architektury aplikacji mobilnej

Projektowanie architektury aplikacji mobilnej rozpoczęto od określenia listy ekranów aplikacji oraz metod nawigacji między nimi. Graf ten został przedstawiony na Rys. 13. Linie ciągłe oznaczają stałe połączenia między ekranami, natomiast linie przerywane oznaczają konfigurowalne linki, które są dostępne lub nie w zależności od ustawień aplikacji. Kierunki nawigacji zostały oznaczone grotami strzałek, natomiast automatyczne przekierowania i ich warunki są opisane za pomocą etykiet na krawędziach.



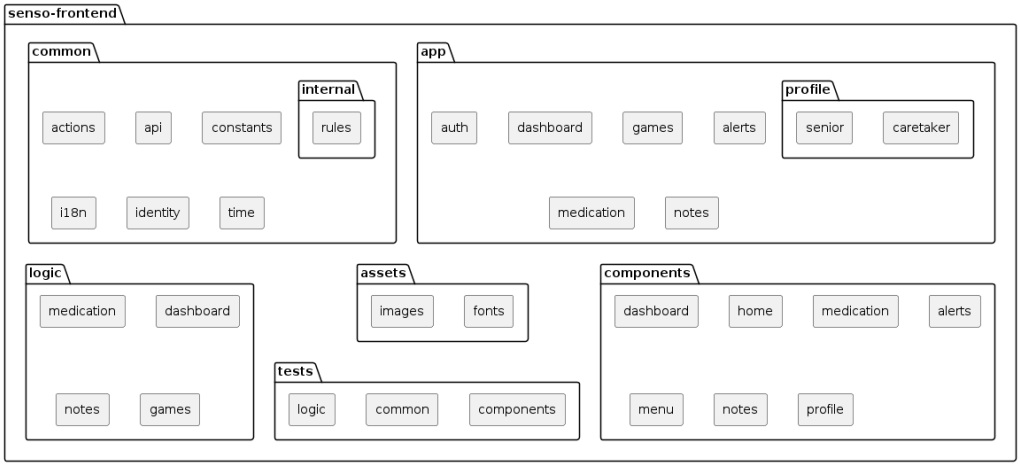
Rysunek 13: Mapa ekranów aplikacji mobilnej.

W kwestii struktury kodu źródłowego aplikacji mobilnej występowała mniejsza dowolność niż w przypadku usług serwerowych, ze względu na ograniczenia narzucane przez bibliotekę React Native i platformę Expo. W związku z tym zdecydowano się na horyzontalny, a nie pionowy podział kodu na pakiety, co oznacza grupowanie według warstwy, a nie implementowanego epiku [15].

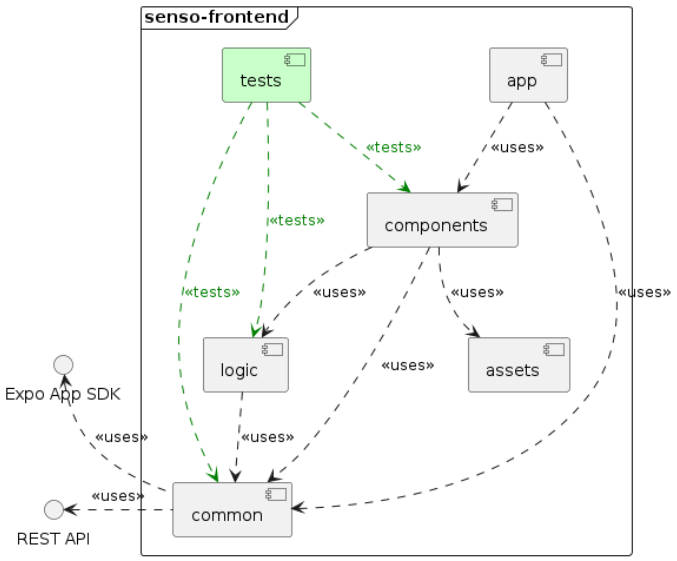
Wyszczególniono popularnie stosowane katalogi, takie jak:

- `app` – zawiera komponenty renderujące poszczególne ekrany aplikacji;
- `assets` – zawiera statyczne zasoby, takie jak obrazy, czcionki, ikony itd.;
- `common` – zawiera narzędzia wspólne dla całej aplikacji, np. odpowiedzialne za nawigację, połączenie z API, czy też stylowanie komponentów;
- `components` – zawiera mniejsze komponenty, takie jak przyciski i pola tekstowe;
- `logic` – zawiera logikę biznesową aplikacji, w formie zbioru funkcji czystych;
- `tests` – zawiera testy jednostkowe i integracyjne.

Tam, gdzie to było możliwe, dzielą się następnie na podkatalogi odpowiadające modułom systemu, np. `app/games`, `app/notes` itd. Diagram pakietów dla aplikacji mobilnej został przedstawiony na Rys. 14, natomiast zależności między nimi zostały ukazane na Rys. 14.



Rysunek 14: Diagram pakietów dla aplikacji mobilnej.



Rysunek 15: Zależności pomiędzy pakietami aplikacji mobilnej.

7. Implementacja

W poniższej sekcji zostały przedstawione szczegóły implementacyjne ciekawszych elementów systemu. Pominięte zostały kwestie konfiguracyjne, czy też realizacje trywialnych funkcjonalności, takich jak dodawanie, usuwanie i modyfikowanie encji. Zamiast tego opis skupia się na zastosowaniu wzorców projektowych, skomplikowanych elementach logiki biznesowej, integracji z zewnętrznymi usługami oraz implementacji algorytmów. Przykłady zostały wzbogacone o fragmenty kodu źródłowego oraz diagramy tam, gdzie było to możliwe.

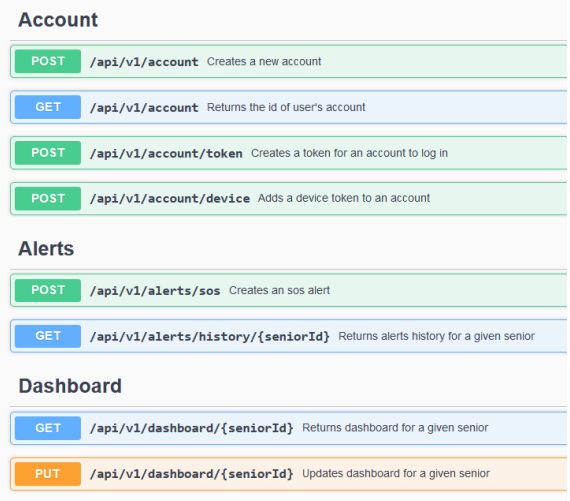
7.1. Połączenie aplikacji mobilnej z usługami serwerowymi

Pobieranie danych z usług jest pozornie prostym zadaniem, jednakże trzeba zadbać o wiele aspektów, takich jak: obsługa błędów, zapamiętywanie danych w pamięci podręcznej, automatyczne odświeżanie, uwierzytelnianie i tym podobne. W ostatnich latach powstało wiele bibliotek, które mają na celu obsługę tego typu zadań, udostępniając programistom wysokopoziomowe interfejsy. Jednym z najpopularniejszych rozwiązań jest TanStack Query (dawniej React Query). Biblioteka ta jest jednak bardzo rozbudowana i wykracza poza potrzeby proponowanego rozwiązania. Zdecydowano się zatem na SWR, które jest lżejszą alternatywą dla TanStack Query, a jednocześnie posiada wszystkie potrzebne w prezentowanej aplikacji funkcjonalności [16].

Kolejnym problemem jest zapewnienie odpowiedniej typizacji danych. Proponowany system korzysta zarówno w aplikacji mobilnej (TypeScript), jak i w usługach serwerowych (C#), z języków typowanych statycznie. Jednakże komunikacja między nimi odbywa się za pomocą formatu JSON, w związku z czym konieczne jest dokonywanie serializacji i deserializacji danych.

W celu rozwiązania tego problemu zdecydowano się na wykorzystanie standardu OpenAPI, który jest otwartoźródłowym standardem do opisu interfejsów REST API. Pozwala on na zdefiniowanie specyfikacji poszczególnych punktów końcowych, w tym ich parametrów, struktury przyjmowanych i zwracanych danych, a także kodów odpowiedzi.

Po stronie serwera wykorzystano automatyczne generowanie specyfikacji na podstawie kodu źródłowego, za pomocą biblioteki Swashbuckle. Istnieje również wizualna wersja specyfikacji, udostępniana za pomocą interfejsu Swagger UI pod ścieżką `/swagger`. Fragment tej dokumentacji został przedstawiony na Rys. 16.



Rysunek 16: Fragment wizualnej reprezentacji specyfikacji OpenAPI.

Dokumentacja powstaje na bazie typów metod kontrolerów, jak i nałożonych na nie atrybutów. Przykład definicji jednej z metod został przedstawiony na Listingu 4. Opisy tekstowe wyciągane są z komentarzy XML, natomiast kody odpowiedzi z atrybutów `ProducesResponseType`.

```
1  /// <summary>
2  /// Deletes the note with a given id.
3  /// </summary>
4  /// <param name="noteId"> id of the note </param>
5  /// <response code="204"> returns the updated note </response>
6  /// <response code="401"> if the user is not logged in </response>
7  /// <response code="403"> if the user is not allowed to access the note </response>
8  /// <response code="404"> if the note with the given id was not found </response>
9  [HasPermission(Permission.MutateNotes)]
10 [HttpDelete("{noteId}")]
11 [ProducesResponseType(StatusCodes.Status204NoContent)]
12 [ProducesResponseType(StatusCodes.Status401Unauthorized)]
13 [ProducesResponseType(StatusCodes.Status403Forbidden)]
14 [ProducesResponseType(StatusCodes.Status404NotFound)]
15 public async Task<IActionResult> Delete(int noteId)
16 {
17     await mediator.Send(
18         new DeleteNoteRequest { AccountId = this.GetAccountId(), NoteId = noteId }
19     );
20     return NoContent();
21 }
```

Listing 4: Przykład definicji metody kontrolera w Swagger.

Przeniesienie typów do aplikacji zapewnia biblioteka openapi-fetch wraz z narzędziem openapi-typescript. Pozwalają one na wygenerowanie plików TypeScript z definicjami typów na podstawie utworzonej uprzednio specyfikacji OpenAPI.

Całość połączona jest przez stworzone na potrzeby projektu hooki, scalające SWR z wygenerowanymi interfejsami. Są to m.in. `useQuery` i `useMutation`. Przykłady użycia pierwszego z nich znajdują się na Listingu 5.

```
1 // Simple request
2 const { data: user } = useApi({ url: "/user" });
3
4 // Dynamic request
5 const [id, setId] = useState(123);
6 const { data: user } = useApi({ url: "/users/{id}", params: { path: { id } } })
```

Listing 5: Przykłady pobierania danych z serwera w aplikacji.

7.2. Statyczna analiza kodu

W celu zapewnienia wysokiej jakości tworzonego oprogramowania zdecydowano się na wykorzystanie narzędzi do statycznej analizy kodu. Do analizy kodu aplikacji mobilnej wykorzystano narzędzie ESLint, korzystające z reguł wykrywających potencjalne błędy, jak rozwiązania nieidiomatyczne, czy też niezgodne z konwencjami [17].

Narzędzie to zezwala na importowanie tzw. *pluginów*, czyli zewnętrznych zestawów gotowych reguł. W proponowanym rozwiązaniu wykorzystano pakiety specyficzne dla bibliotek React, React Native, Expo oraz ogólnych reguł dotyczących języka TypeScript.

Na szczególną uwagę zasługują natomiast niestandardowe reguły, które zostały zaimplementowane specjalnie dla potrzeb prezentowanego projektu. Są to:

- `senso-app-routes` – sprawdza, czy wszystkie łącza w aplikacji korzystają ze zdefiniowanych stałych, a nie żadnych innych ciągów znaków;
- `senso-export-policy` – wymusza odpowiednią składnię eksportów;
- `senso-import-sources` – wymusza odpowiednią składnię importów;
- `senso-parent-import` – blokuje importowanie z katalogów nadrzędnych, co prowadzi do zależności cyklicznych, które powodują częste błędy kompilacji;
- `senso-stack-screen` – wymusza korzystanie z dobrego komponentu paska nawigacji;
- `senso-style-wrapper` – sprawdza, czy wszystkie definicje stylów korzystają ze zdefiniowanych funkcji pomocniczych zamiast rozwiązań wbudowanych w bibliotekę;
- `senso-test-location` – weryfikuje, że testy znajdują się w dobrych katalogach.

Każda reguła składa się z funkcji wywoływanej przy przejściu przez drzewo składni abstrakcyjnej na każdym jego węźle. Funkcja ta musi sprawdzić, czy dany węzeł spełnia kryteria reguły, a jeśli nie, to zgłosić błąd. Przykładowa implementacja została przedstawiona na Listingu 6. Reguła wykrywa wyrażenia wywołania funkcji `StyleSheet.create` i proponuje ich zamianę na wykorzystywaną wewnętrznie w projekcie funkcję `sty.create`.

```
module.exports = {
  meta: { type: "suggestion" },
  create(context) {
    return {
      CallExpression({ callee }) {
        if (
          callee.type === "MemberExpression" &&
          callee.object.name === "StyleSheet" &&
          callee.property.name === "create"
        ) {
          context.report({
            node: callee,
            message:
              "Use ‘sty.create‘ or ‘sty.themedHook‘ from ‘@/common/styles‘ instead.",
          });
        }
      },
    };
  },
};
```

Listing 6: Implementacja reguły ESLint senso-style-wrapper.

7.3. Internacjonalizacja aplikacji mobilnej

Na potrzeby aplikacji powstało autorskie rozwiązanie do internacjonalizacji, które pozwala na tłumaczenie tekstu na bazie pliku JSON. Dostępne są dwa języki – polski i angielski. Rozwiązanie pozwala na tłumaczenie pojedynczych słów, jak i całych zdań, a także wspiera odmianę liczebników z podmiotami, np. „1 lek”, „2 leki”, „5 leków”. Do odmiany wykorzystany jest specjalny parametr `count` – dozwolone jest przekazywanie parametrów do tłumaczeń, które są dodatkowo wstawiane w miejsca oznaczone klamrami w treści tłumaczenia.

Przykładowe klucze tłumaczeń zostały przedstawione na Listingu 7.

```
{
    "auth.password": {
        "en": "Password",
        "pl": "Hasło"
    },
    "medication.pills": {
        "en": "pills",
        "en_1": "pill",
        "pl": "tabletek",
        "pl_1": "tabletka",
        "pl_2,3,4": "tabletki"
    },
    "profiles.scanQR.alertDescription": {
        "en": "Do you want to add \"{name}\" as your senior?",
        "pl": "Czy chcesz dodać \"{name}\" jako swojego seniora?"
    }
}
```

Listing 7: Wybrane klucze tłumaczeń w aplikacji mobilnej.

Z powyższych kluczy można skorzystać w aplikacji w sposób przedstawiony na Listingu 8.

```
function Component() {
    const { t } = useI18n();

    return (
        <View>
            <Text>{t("auth.password")}</Text>
            <Text>{t("medication.pills", { count: 2 })}</Text>
            <Text>{t("profiles.scanQR.alertDescription", { name: "John" })}</Text>
        </View>
    );
}
```

Listing 8: Przykład użycia tłumaczeń w aplikacji mobilnej.

Interfejs aplikacji jest automatycznie odświeżany przy zmianie języka. Cały system jest mockowany tak, że w trakcie uruchamiania testów użyta jest zawsze angielska wersja aplikacji. Nie dotyczy to jednak oczywiście testów jednostkowych samego mechanizmu tłumaczeń. Statyczna analiza kodu blokuje próby użycia wpisanych bezpośrednio w kodzie komponentów tekstów, zamiast korzystania z funkcji tłumaczących.

7.4. Walidacja danych, obsługa wyjątków

Istotną częścią proponowanego systemu jest walidacja danych oraz obsługa wyjątków. Autorom zależało na tym, aby obsługa tych aspektów była prosta, jednolita i trudna do pominięcia.

Do podstawowej walidacji wykorzystany został pakiet FluentValidation, który pozwala na definiowanie reguł w postaci klas. Sposób użycia jest dość intuicyjny i został przedstawiony na Listingu 9.

```
public sealed class CreateNoteValidator : AbstractValidator<CreateNoteRequest>
{
    public CreateNoteValidator()
    {
        RuleFor(r => r.Dto.Content).NotEmpty().WithMessage("Content is required.");
        RuleFor(r => r.Dto.Title).MaximumLength(255).WithMessage("Title is too long.");
    }
}
```

Listing 9: Zastosowanie FluentValidation do walidacji notatek.

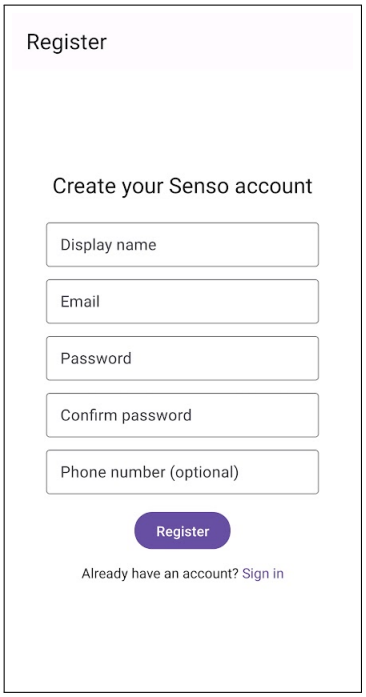
Wszystkie klasy walidujące są automatycznie (za pomocą mechanizmu refleksji) rejestrowane w kontenerze IoC, a następnie za pomocą mediatora wywoływane przy każdej komendzie.

Do obsługi błędów zastosowano wyjątki, które są rzucane w przypadku nieprawidłowych danych i tłumaczone na odpowiednie kody odpowiedzi HTTP. W przypadku wystąpienia wyjątku, który nie jest automatycznie obsługiwany, jest on wyświetlany w konsoli oraz skutkuje zwróceniem kodu `500 Internal Server Error`. Na potrzeby tej logiki powstała klasa `ApiException` oraz pośrednik `ExceptionMiddleware`. Ta pierwsza pozwala na dołączenie kodu odpowiedzi HTTP do dowolnego wyjątku, a druga przechwytuje wszystkie wyjątki rzucone w trakcie przetwarzania zapytania i przekształca je na odpowiedzi.

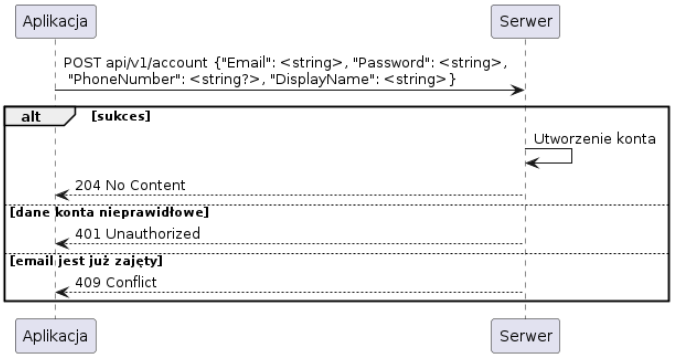
7.5. Uwierzytelnianie i autoryzacja

Do uwierzytelniania został wykorzystany standard JWT. Jest to otwarty standard, który pozwala na poświadczanie tożsamości użytkownika za pomocą podpisanych tokenów. Stanowi on alternatywę dla tradycyjnych rozwiązań opartych o ciasteczka i sesje. JWT jest rozwiązaniem często krytykowanym, z uwagi na jego wady [18, 19]. Jednakże na potrzeby proponowanego systemu jest to rozwiązanie wystarczające, a jego zaletą jest prostota implementacji.

Rejestracja w systemie polega na utworzeniu nowego konta na bazie podanych przez użytkownika w formularzu danych. Formularz rejestracji został przedstawiony na Rys. 17. Podane dane są walidowane, a podane hasło jest haszowane przed zapisaniem do bazy danych. Proces rejestracji przedstawia diagram sekwencji na Rys. 18.

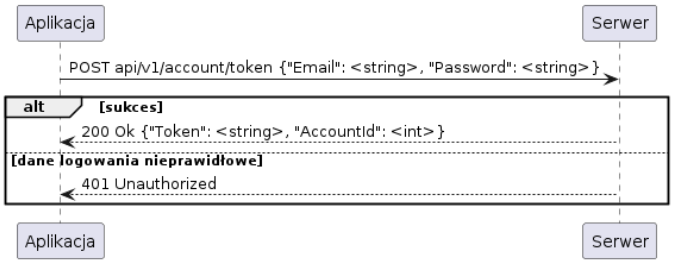


Rysunek 17: Formularz rejestracji w aplikacji mobilnej.



Rysunek 18: Diagram sekwencji rejestracji konta.

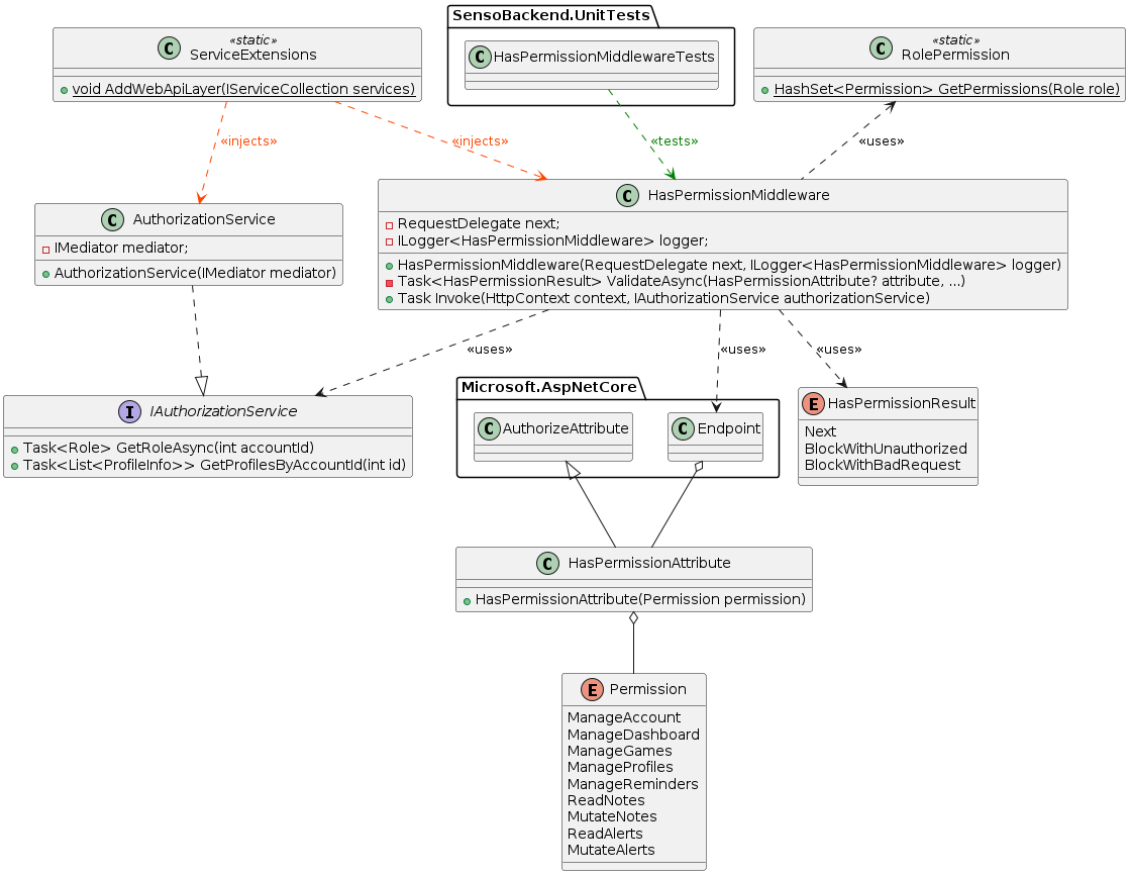
Logowanie wymaga podania przez użytkownika adresu e-mail oraz hasła w formularzu podobnym do poprzedniego. Dane są weryfikowane po stronie serwera, a następnie, jeżeli są poprawne, zwracany jest token JWT, zapisywany w pamięci podręcznej aplikacji wraz z identyfikatorem konta użytkownika. Proces logowania przedstawia diagram sekwencji na Rys. 19.



Rysunek 19: Diagram sekwencji logowania i uzyskania JWT.

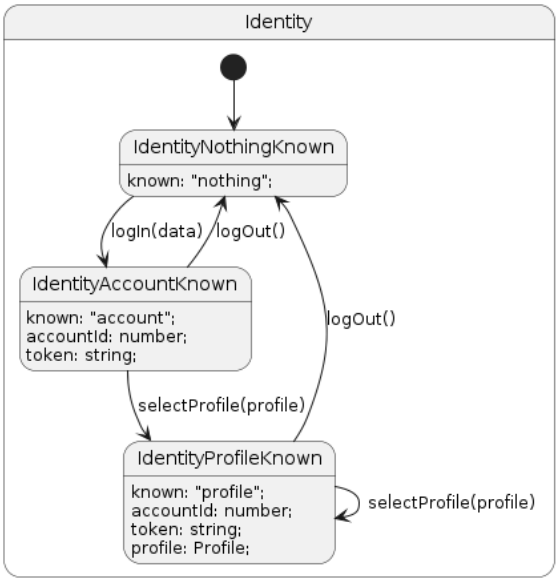
Za autoryzację po stronie serwera odpowiada stworzone na potrzeby projektu rozwiązanie. Wbudowane w ASP.NET Core atrybuty autoryzacji nie zostały wykorzystane, ponieważ nie spełniały dość specyficznych przypadków użycia projektu. Przykładowo, dopuszczania do zasobu seniora, ale również jego opiekunów, bez konieczności powtarzania logiki.

Powstał atrybut `HasPermissionAttribute`, który pozwala na sprawdzenie, czy użytkownik posiada określone w parametrze uprawnienie. W przypadku, gdy nie ma on dostępu do zasobu lub nie jest w ogóle uwierzytelniony, automatycznie zwracany jest kod odpowiedzi `401 Unauthorized`. Każda próba dostępu do zasobu jest logowana, a w przypadku niepowodzenia opisany jest również jego powód. Za odpowiednią obsługę atrybutu odpowiada pośrednik `HasPermissionMiddleware`, który wykonuje serię testów, sprawdzając m.in. czy użytkownik jest uwierzytelniony, czy jest administratorem, typ jego profilu i inne warunki, aby określić, dopuszczenie lub zablokowanie dostępu do zasobu. Diagram klas dla tego rozwiązania został przedstawiony na Rys. 20.



Rysunek 20: Diagram klas dla autoryzacji.

Po stronie aplikacji mobilnej stan uwierzytelnienia reprezentowany jest przez maszynę stanów. Jej obecny stan jest przechowywany w pamięci podręcznej aplikacji, co zapewnia trwałość stanu logowania nawet po zamknięciu aplikacji. Wspomniany diagram stanów został przedstawiony na Rys. 21.



Rysunek 21: Diagram stanów dla uwierzytelnienia w aplikacji mobilnej.

Dostęp do aktualnego stanu można uzyskać za pomocą hooka `useIdenity`, natomiast aktualnie przechowywany token jest automatycznie dołączany do nagłówka `Authorization` przy każdym zapytaniu wykonanym przez funkcję `useQuery`.

7.6. Parowanie seniora i opiekuna

Parowanie seniora i opiekuna odbywa się za pomocą skanowania kodu QR, zgodnie ze sporządzonymi wcześniej wymaganiami. Celem takiego rozwiązania jest zapewnienie bezpieczeństwa i wygody. Parowanie inicjuje senior, aby zminimalizować szansę ataku na seniora, np. poprzez wyświetlanie kodów QR, które mogłyby zostać przez niego omyłkowo zeskanowane. Każdy kod ma krótki termin ważności, po którym przestaje być aktywny. Kod jest również jednorazowy i wygasa po pierwszym skanowaniu.

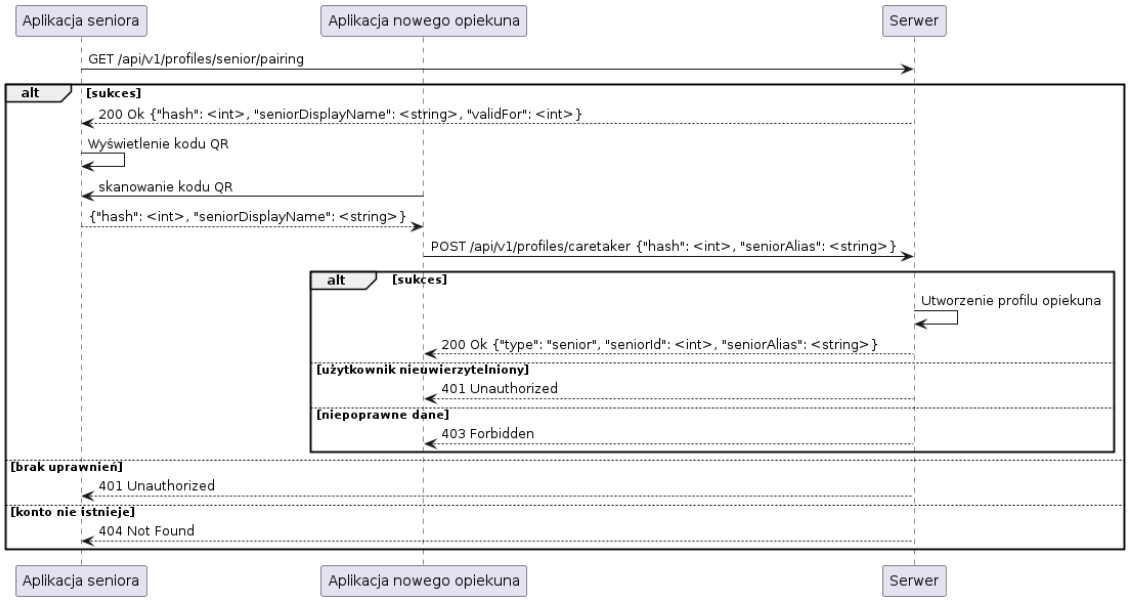
Ekran, który wykorzystuje senior, został przedstawiony na Rys. 22. Wyświetlony kod może zostać zeskanowany przez opiekuna, który w ten sposób zostaje powiązany z kontem seniora. Opiekun może podać niestandardowy alias dla seniora, jeżeli tego nie zrobi, zostanie wykorzystana nazwa wyświetlana konta seniora.

Do wyświetlania i skanowania kodów QR wykorzystano biblioteki React QR Code oraz Expo BarCodeScanner. Użycie skanera wymaga zgody użytkownika na dostęp do aparatu urządzenia.

Każdy kod stanowi 32-bitowy hasz, który w warstwie serwerowej jest mapowany do struktury zawierającej identyfikator konta seniora, jego nazwę wyświetlaną oraz datę wygaśnięcia. Kod jest zwracany z serwera wraz z liczbą sekund do wygaśnięcia i nazwą seniora.

Rysunek 22: Ekran wyświetlania kodu QR w aplikacji mobilnej.

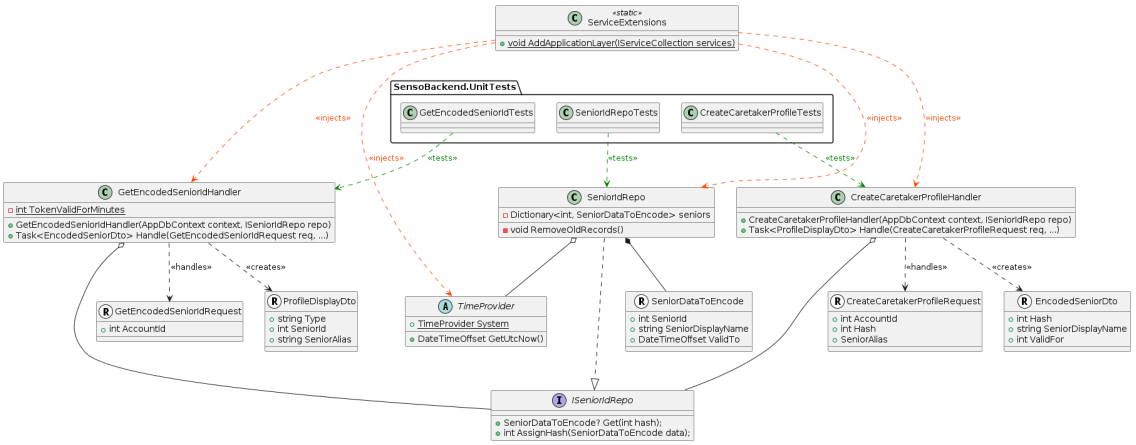
Proces parowania został przedstawiony na diagramie sekwencji na Rys. 23.



Rysunek 23: Diagram sekwencji parowania seniora i opiekuna.

Po stronie serwera, za przechowywanie kodów odpowiedzialne jest specjalne repozytorium, przechowujące kody w pamięci. Oznacza to, że po ponownym uruchomieniu serwera, wszystkie kody zostaną utracone, co nie stanowi problemu, ponieważ kody i tak są jednorazowe i mają krótki termin ważności.

Repozytorium pozwala na nadawanie nowych kodów, jak i sprawdzanie ważności i zawartości nadanych poprzednio haszy. Odpowiedni wycinek systemu został przedstawiony na diagramie klas na Rys. 24. Na diagramie przestawiono jedynie warstwę aplikacji.



Rysunek 24: Diagram klas dla parowania seniora i opiekuna.

7.7. System powiadomień

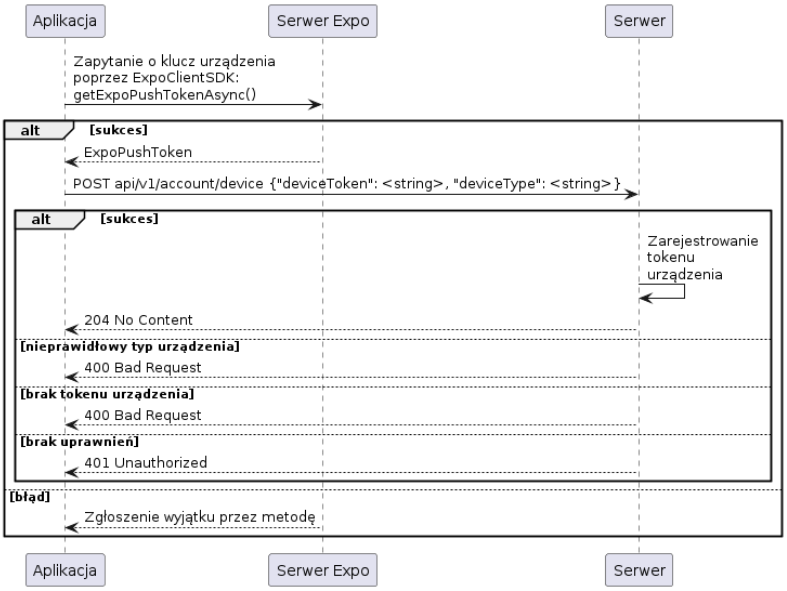
Jak zostało wspomniane w sekcji 6.1., system powiadomień opiera się o zestaw narzędzi Expo, oferujący abstrakcję nad różnymi dostawcami usług powiadomień push. Najpopularniejszymi dostawcami są odpowiednio Firebase Cloud Messaging dla systemu Android oraz Apple Push Notifications dla iOS.

Expo pozwala na dobór odpowiedniego dostawcy w zależności od systemu operacyjnego. Jednakże zgodnie z ustalonymi ograniczeniami, system wspiera powiadomienia jedynie na systemie Android, ponieważ APN wymaga utrzymania płatnego konta deweloperskiego.

Takie rozwiązanie wymaga dokonania konfiguracji w wielu miejscach, m.in. dodania aplikacji do paneli deweloperskich Firebase oraz Expo, generacji kluczy i umieszczenia ich w odpowiednich katalogach projektu aplikacji mobilnej.

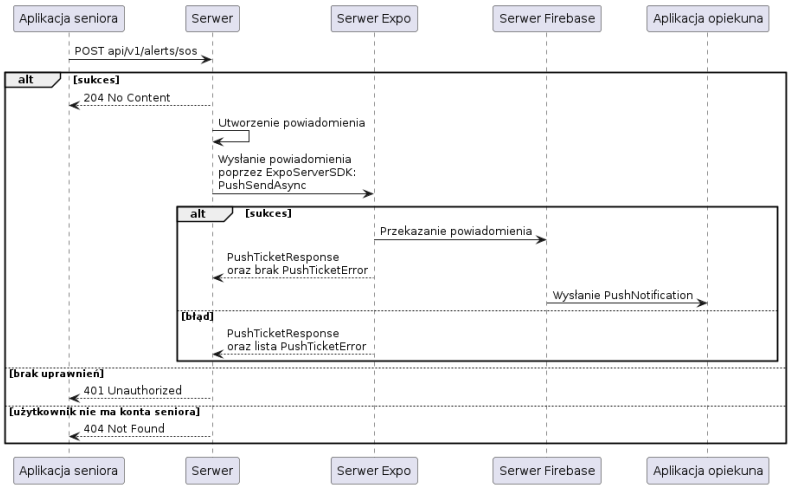
Po stronie serwera powstał punkt końcowy, pozwalający na przypisanie urządzenia do konta. Zgodnie z REG/018 i REG/081, konto może mieć jedynie jedno urządzenie, na które wysyła powiadomienia, a na danym urządzeniu może być aktywne tylko jedno konto. Warto zauważyć, że ograniczenie to nie dotyczy profili, tzn. jeżeli użytkownik jest opiekunem dwóch seniorów, to może otrzymywać powiadomienia od obu z nich.

Diagram sekwencji dla rejestracji urządzenia został przedstawiony na Rys. 25.



Rysunek 25: Diagram sekwencji rejestracji urządzenia.

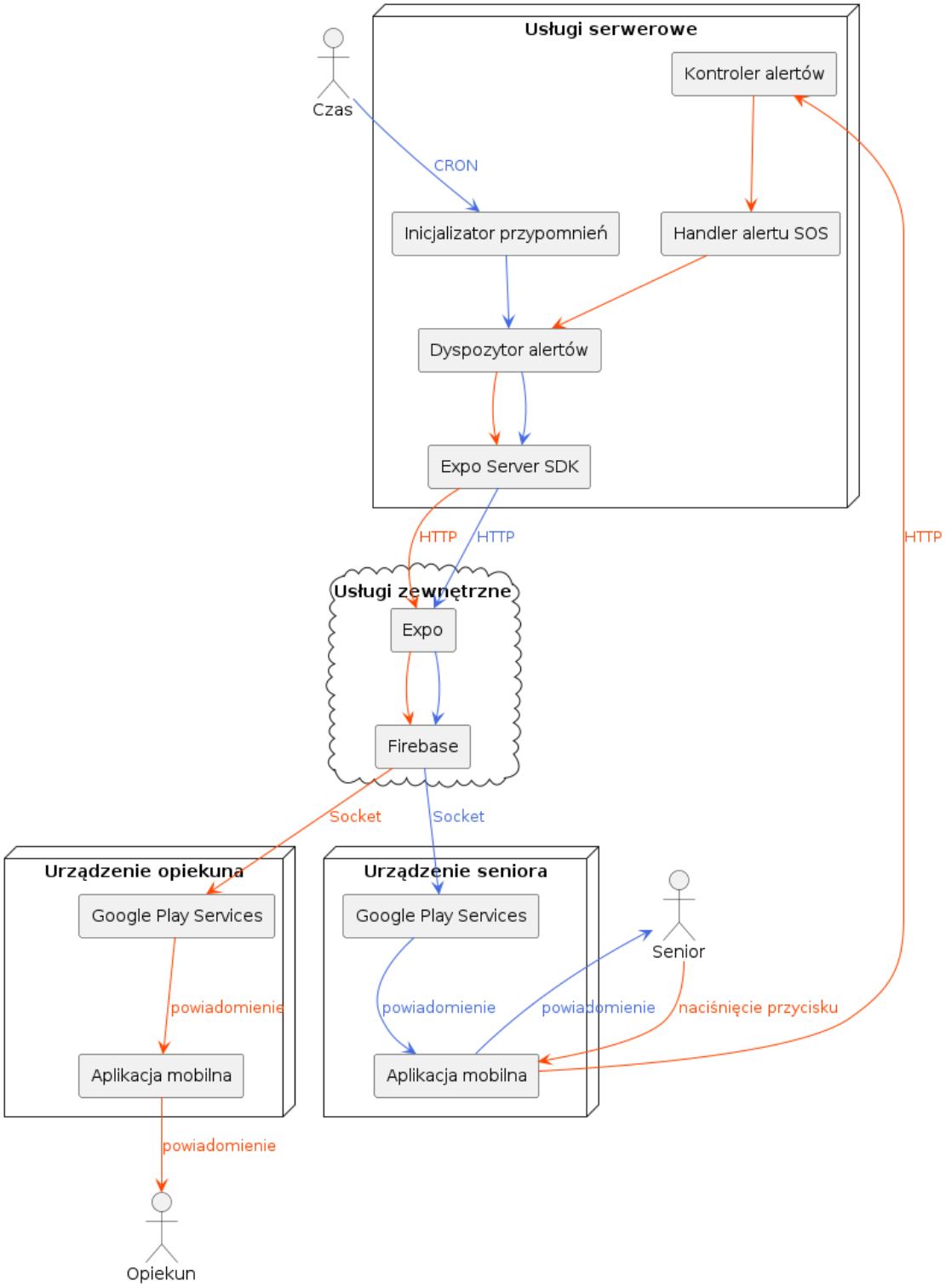
Większość alertów jest zgłaszana automatycznie przez system, wyjątkiem jest alert SOS, do którego powstał osobny punkt końcowy, do którego wysyłane jest żądanie po naciśnięciu przez seniora przycisku SOS znajdującego się w rogu panelu. Diagram sekwencji dla tego przypadku użycia został przedstawiony na Rys. 26.



Rysunek 26: Diagram sekwencji wysłania alertu SOS.

Każdemu alertowi o zapomnianych lekach odpowiada wyrażenie CRON. Przy każdej dacie i godzinie pasującej do wyrażenia wysyłane jest powiadomienie do seniora. Jeżeli w ciąg 5 minut od wysłania powiadomienia nie zostanie ono oznaczone jako przeczytane, wysyłane jest kolejne powiadomienie, tym razem do opiekuna. Za obsługę tego procesu odpowiada osobny serwis, który jest uruchamiany przy starcie aplikacji. Wewnętrznie używa on narzędzia do harmonogramowania Hangfire [20].

Całościowy przepływ danych pomiędzy poszczególnymi elementami systemu, przy wysyłaniu powiadomień obrazuje diagram Rys. 27. Kolorem *pomarańczowym* został oznaczony przepływ danych przy wysyłaniu alertu SOS, natomiast kolorem *niebieskim* został oznaczony przepływ danych przy wysyłaniu automatycznych przypomnień o lekach.



Rysunek 27: Diagram przepływu danych przy wysyłaniu powiadomień.

7.8. Algorytmy dotyczące gier

Najciekawszą pod względem implementacji grą w proponowanym systemie jest Sudoku. Został tam zastosowany algorytm oparty o technikę *backtracking*. Generacja planszy polega na losowym wypełnieniu pierwszego wiersza, a następnie wypełnianiu kolejnych pustych pól kolejnymi liczbami spośród tych, które nie występują w wierszu, kolumnie i kwadracie o boku 3. W przypadku, gdy nie można wypełnić pola żadną liczbą, następuje wycofywanie poprzednich wypełnień i ponowne próby.

Na uwagę zasługuje również dobór słów do gry Wordle. Lista została określona ręcznie, zachowując przystępne słowa o długości 5 znaków spośród najpopularniejszych występujących w danym języku. Lista dostosowuje się do obecnie wybranego języka w aplikacji.

8. Testy produktu programowego

8.1. Plan testowania

Z uwagi na naturę aplikacji, jak i dobre praktyki programistyczne, zapewnienie poprawnego działania systemu jest kluczowe. Poprawne działanie systemu zapewniono poprzez wprowadzenie i realizację planu testowania, który podzielony był na dwie główne gałęzie: testy jednostkowe i badanie środowiskowe.

Testy jednostkowe były pisane w trakcie implementacji i uruchamiane przed każdym wprowadzeniem zmian w celu upewnienia się, czy nie zepsuły one istniejących modułów. Badanie środowiskowe zostało przeprowadzone pod koniec projektu, gdy aplikacja była już gotowa. Pomogło ono zweryfikować spełnienie wymagań w obrębie dostępności i użyteczności oraz zareagować na informację zwrotną, wprowadzając odpowiednie poprawki.

8.2. Testy jednostkowe

Testy jednostkowe były pisane na równi z kodem aplikacji. Pozwoliły one weryfikować implementację i znajdować ewentualne błędy, zanim staną się one problemem. Testy zostały też włączone do systemu CI/CD z pomocą GitHub Actions, co sprawiło, że niemożliwe było wdrożenie kodu, dla którego testy nie kończyły się sukcesem. Testy dla każdej funkcjonalności pisane były w taki sposób, aby w miarę możliwości sprawdzić wszystkie możliwe ścieżki. Przykładowo, jeśli sprawdzano możliwość dodawania leku, sprawdzano ją dla seniora, opiekuna danego seniora i osoby niezalogowanej, lub opiekuna innego seniora. Bardzo ułatwiło to pracę w przypadkach, gdy trzeba było sprawdzić dodatkowe warunki (np. opiekun może zobaczyć tylko publiczne notatki swojego seniora), gdyż w takich przypadkach szybko wykrywano błędy. Samo pisanie testów pozwoliło również weryfikować założenia i pierwotne projekty systemu. Zdarzało się, że podczas pisania testów jednostkowych znajdowano pewne przypadki skrajne (ang. *edge case*), które nie zostały wcześniej uwzględnione i wymagały drobnych modyfikacji.

8.3. Badanie środowiskowe

Badanie środowiskowe było krokiem, który od samego początku był kluczowy. Z uwagi na grupę docelowych odbiorców aplikacji niezbędna była weryfikacja wymagań na podstawie ich opinii. W tym celu sporządzony został formularz użytkownika. Był on skierowany zarówno do seniorów, jak i do opiekunów. Składał się z kilku zadań dla konkretnych grup, które miały przeprowadzić ich przez aplikację, oraz z pytań po każdym z nich.

Zadania były skonstruowane w minimalistyczny sposób, prosząc użytkownika o wykonanie pewnej czynności, ale bez podawania instrukcji, aby sprawdzić, czy aplikacja faktycznie jest tak intuicyjna, jak założono. Niektóre zadania były skierowane w stronę konkretnego typu profilu, jednak wszyscy wypełniający formularz zostali zachęceni, aby spróbować wykonać wszystkie z nich.

Pełna lista zadań znajduje się poniżej:

Zadanie 1. Stwórz konto w aplikacji i profil seniora.

Zadanie 2. (dla opiekuna) Wyświetl kod QR seniora (z poziomu profilu seniora) i stwórz profil opiekuna (inny profil na tym samym koncie).

Zadanie 3. (dla seniora) Wyślij w aplikacji sygnał SOS.

Zadanie 4. (dla opiekuna) Dodaj do panelu seniora nowy gadżet, który jeszcze się na nim nie znajduje.

Zadanie 5. Dodaj nowy lek.

Zadanie 6. (dla seniora) Zapisz notatkę zwykłą i prywatną oraz odsłuchaj jedną z nich.

Zadanie 7. (dla seniora) Zagraj w każdą z trzech gier i sprawdź swój wynik w rankingu graczy.

Zadanie 8. Spróbuj zmienić w ustawieniach język aplikacji (polski-angielski).

Po każdym z zadań zadane seniorom zostały następujące pytania:

- Jak oceniasz skomplikowanie tego zadania?
- W jakim stopniu poznana część aplikacji spełnia Twoje oczekiwania w kwestii ...? (gdzie ... to wymagania realizowane w danym zadaniu, np. „dostosowania wyglądu panelu" w zadaniu 4)
- Jak oceniasz wygląd (przejrzystość) tej części aplikacji?
- Czy masz jakieś komentarze związane z poznaną częścią aplikacji?

Pierwsze trzy pytania były oceną w skali od 1 do 5, gdzie 1 to najgorsza ocena, a 5 to najlepsza. Pod ostatnim pytaniem znajdowało się wolne pole na tekst z komentarzem.

Na samym końcu zadane zostały też pytania o ocenę ogólnych doświadczeń z aplikacją i jej szaty graficznej (również w skali 1-5) oraz wszelkie końcowe komentarze. Podczas przeprowadzania testów członkowie zespołu byli obecni przy seniorach, aby pomóc im w razie problemów i obserwować zachowania seniorów, które trudno sprawdzić z pomocą samego formularza.

8.4. Wyniki badania środowiskowego

Wstęp

W badaniu wzięła udział grupa 11 użytkowników, w tym 5 seniorów i 6 opiekunów. Mimo niewielkiej ilości osób zaczęto zauważać pewne trendy w ich odpowiedziach, które sugerowały konkretne problemy. Analiza wyników została przeprowadzona pytanie po pytaniu z podziałem na role (tj. najpierw omawiane będą odpowiedzi na pytanie o skomplikowanie dla wszystkich zadań, uwzględniając rolę opowiadającego).

Skomplikowanie zadań

Pierwsze z pytań dotyczyło skomplikowania zadania. Wyższe wartości oznaczały w tym przypadku, że zadanie jest prostsze. Zgodnie z przewidywaniami – w większości przypadków opiekunowie uznawali zadania za prostsze niż seniorzy. Była to grupa młodsza, więc też bardziej obeznana ze smartfonami, jednak nawet oceny seniorów były raczej wysokie. Najtrudniejsze dla seniorów okazały się zadania 1 (zakładanie konta/profilu) oraz 4 (dodanie gadżetu do panelu, zadanie opiekuna). Paradoksalnie jest to dobra wiadomość, gdyż pierwsze zadanie jest zadaniem, przy którym raczej opiekun będzie pomagał seniorowi. Zadanie czwarte i tak jest przeznaczone głównie dla opiekunów, a wśród tej grupy było ono drugim najwyżej punktowanym zadaniem.

Za najprostsze zadania seniorzy uznali zadanie 3 (wysłanie sygnału SOS) oraz 5 (dodanie nowego leku). W zadaniu piątym średnia ocena seniorów jest nawet wyższa niż ocena opiekunów. Jest to bardzo dobra informacja, gdyż są to najważniejsze dla seniorów funkcje, które zapewniają ich bezpieczeństwo.

Spełnienie oczekiwań użytkowników przez zadania

Drugie z pytań dotyczyło spełnienia oczekiwań użytkowników przez konkretne elementy aplikacji. Wyniki w tym pytaniu były ogólnie wyższe, niż w poprzednim, co oznacza, że pomimo drobnych problemów aplikacja spełnia oczekiwania użytkowników. Same wyniki były też dużo bardziej zbliżone między seniorami a opiekunami, jedyne większe różnice dało się zauważyć w zadaniach 6 (notatki) i 7 (gry). Były one też najgorzej oceniane przez seniorów, co może sugerować, że wymagają one odrobiny dodatkowej pracy w przyszłości. Na równi z zadaniem 6 był też wynik zadania 3 (sygnał SOS), które mimo najwyższej prostoty w wykonaniu zadania, nie spełniło tak dobrze oczekiwań seniorów, co zostało opisane w późniejszej sekcji.

Wszystkie pozostałe zadania dzielą między sobą dwa najwyższe wyniki, z zadaniem 2 (parowanie z opiekunem) z najwyższą możliwą oceną. Jest to kolejna dobra wiadomość, gdyż oznacza to, że seniorom spodobał się poziom zabezpieczeń i mimo lekkich problemów w wykonaniu zadania, docenili oni jego zamysł. Opiekunowie również wysoko ocenili poziom spełnienia oczekiwań, najniższa średnia ocena to 4.5 (w skali od 1 do 5), co jest bardzo dobrym wynikiem.

Przejrzystość interfejsu przy zadaniach

Trzecie z pytań dotyczyło przejrzystości interfejsu w konkretnych elementach aplikacji. Wyniki w tym pytaniu były bardziej różnorodne niż w poprzednich. Niektóre fragmenty zostały dużo lepiej ocenione przez seniorów, a inne przez opiekunów, nie było tutaj silnej przewagi jak przy pierwszym pytaniu. Seniorzy tutaj ponownie najgorzej ocenili zadania 6 i 7, w odróżnieniu od opiekunów, u których miały one średnie (względem reszty zadań) oceny. Pojawienie się takich ocen tych zadań w dwóch pytaniach może oznaczać, że wymagają one pewnych poprawek w przyszłości.

Najlepiej oceniane przez seniorów były zadania 3 oraz 8. Zarówno w najlepiej, jak i najgorzej ocenianych zadaniach, różnice między opiniami seniorów i opiekunów były największe. Może to pozwolić w przyszłości lepiej zidentyfikować preferencje obu grup i dostosować do nich wybrane elementy aplikacji.

Komentarze do zadań

Użytkownicy mogli też zostawić komentarze dotyczące fragmentów aplikacji po każdym zadaniu. Komentarze te pomogły zrozumieć przyczyny części ocen seniorów i opiekunów.

Przy zadaniu 3 seniorzy mimo pozytywnej reakcji na system zastanawiali się, czy możliwe jest, aby sygnał SOS mógł być aktywowany, nawet gdy nie będą mieli przy sobie telefonu lub nie będą fizycznie w stanie sami aktywować sygnał. W zadaniu 4 seniorzy mieli pewne problemy z widocznością ograniczenia liczby gadżetów, czy faktem, że nie do końca rozumieli, czym jest panel. W zadaniu 6 problemem był rozmiar przycisku do odsłuchania notatek, powinien on być większy i bardziej widoczny. Problemem zadania 7 był brak przedstawienia zasad gier, które nie zawsze były jasne dla seniorów.

Ogólne doświadczenia z aplikacją

Zarówno seniorzy, jak i opiekunowie ocenili swoje doświadczenia z aplikacją na 5 albo 4, ze średnią w okolicach 4.5. Ogólna szata graficzna aplikacji została również oceniana pozytywnie, poza jedną odpowiedzią opiekuna, znaczna większość odpowiedzi to 5, z pojedynczymi odpowiedziami 4. Jedynym komentarzem był tutaj jeden komentarz seniora mówiący, że aplikacja jest dobrym pomysłem, ale trzeba jakoś przeszkolić seniorów, aby mogli z niej sprawnie korzystać.

8.5. Zgodność ze standardami dostępności

Została sprawdzona zgodność projektu ze standardem WCAG 2.1, który jest zbiorem rekomendacji dotyczących dostępności systemów cyfrowych [21].

Spełnione zostały wszystkie wymagania z filaru „1. Postrzegalność", w szczególności dotyczące rozmiaru i czytelności tekstu, czy też dostępności multimediów. Obie szaty graficzne aplikacji spełniają minimalne wytyczne dotyczące kontrastu kolorów. Na ekranach z rozmiarem zbliżonym do przeciętnego smartfona, zostały zachowane odpowiednie odstępy między elementami interfejsu.

Z sekcji drugiej „2. Funkcjonalność" nie został spełniony jeden warunek, dotyczący możliwości nawigacji za pomocą klawiatury. Warto zauważyć, że odnosi się to do sytuacji, w której użytkownik podłącza do swojego urządzenia klawiaturę zewnętrzną za pomocą kabla, a nie klawiatury ekranowej. Jest to oczywiście możliwa sytuacja, jednakże mało prawdopodobna. Pozostałe wymagania z tej sekcji zostały spełnione.

Sekcja trzecia „3. Zrozumiałość" dotyczy jakości tłumaczeń, m.in. czy nie zawierają one trudnych do zrozumienia zwrotów, czy też nie są zbyt długie. Jest to bardzo subiektywna kwestia, która wymagałaby głębszej analizy. Jednakże badania środowiskowe nie wykazały tego typu problemów.

Filar „4. Solidność" dotyczy poprawności zastosowanych w kodzie znaczników. Powinno to zostać zapewnione przez wykorzystywaną bibliotekę do budowy interfejsu użytkownika, zgodnie z jej dokumentacją.

Aplikacja mobilna nie była testowana na urządzeniach z większymi rozmiarami ekranu jak np. tablety, jednakże powinna prezentować się poprawnie, ponieważ interfejs użytkownika został zaprojektowany z myślą o responsywności.

Standard WCAG 2.1 dotyczy głównie stron internetowych, jednakże większość jego wytycznych była możliwa do zastosowania również w aplikacji mobilnej. Wspomniane powyżej naruszone wytyczne miały wszystkie kryteria sukcesu na poziomie AAA, co oznacza, że aplikacja spełnia wymagania na poziomie AA. Warto zauważyć, że specyfikacja odradza wyznaczania poziomu AAA jako wymaganego do spełnienia w 100%, ponieważ może być to niewykonalne dla niektórych systemów.

9. Podsumowanie

9.1. Stopień realizacji założeń

Proponowane rozwiązanie spełnia wszystkie wymagania funkcjonalne i niefunkcjonalne z kategorii Must Have, Should Have i Could Have. Jednocześnie warto pamiętać, że zaimplementowany system jest jedynie prototypem i wymaga dalszego rozwoju przed wdrożeniem.

Zostały zaimplementowane wszystkie moduły systemu, możliwe jest pełne zarządzanie kontami i profilami, w tym parowanie seniora i opiekuna. Użytkownik może zarządzać swoimi lekami i notatkami, a także edytować swój panel. Dostępne są trzy gry stymulujące umysł seniora. Wdrożony został system powiadomień, jednakże wymaga on jeszcze drobnych poprawek.

W kilku aspektach udało się wyjść ponad początkowe wymagania, przykładowo została dodana możliwość przeglądu listy swoich opiekunów oraz ewentualnego odpinania ich z widoku tej listy.

Najwięcej zadań projektowych zostało zrealizowanych w epikach „Maintenanace" i „Documentation", co oznacza, że spora część efektów zadań nie jest widoczna dla użytkownika końcowego. Są to między innymi konfiguracja środowiska deweloperskiego, testy jednostkowe, dokumentacja kodu na potrzeby wewnętrzne, statyczna analiza kodu, czy też konfiguracja CI/CD. Najbardziej obszernym epikiem funkcjonalnym był „Accounts and profiles", co podkreśla złożoność tej części systemu.

9.2. Perspektywy rozwoju

Istnieje wiele możliwości rozwoju proponowanego systemu. Część z nich jest czysto techniczna i wiąże się m.in. z refaktoryzacją obecnych rozwiązań, poprawą wydajności, czy też zwiększeniem pokrycia kodu testami. Jednakże istnieją też możliwości rozwoju funkcjonalnego, stanowiące wnioski z przeprowadzonego badania środowiskowego.

Zmiany techniczne

Należy rozważyć zmianę JWT na system sesji. Jest to łatwe do wykonania, jako że w finalnej wersji usługi serwerowe stanowią monolit i nie są rozproszone. Zmiana ta pozwoli na rozwój bezpieczeństwa systemu, m.in. poprzez wprowadzenie możliwości wylogowania użytkownika ze wszystkich urządzeń [18].

Poprawek wymaga również system powiadomień. W obecnej wersji jest on w pełni funkcjonalny, ale wymaga przechowywania w pamięci obiektu dla każdego istniejącego w systemie powiadomienia, co nie jest rozwiązaniem skalowalnym do setek czy tysięcy użytkowników.

Zmiany funkcjonalne

W aplikacji brakuje wielu częstych funkcji związanych z zarządzaniem kontami, które nie zostały uwzględnione w wymaganiach. Są to między innymi zmiana hasła, resetowanie hasła, czy też edycja swojej nazwy wyświetlanej.

Istnieje możliwość dodania kolejnych gier, do systemu. Warto rozważyć również utworzenie tablic wyników dla znajomych, a nie tylko globalnych, które pozwoliłyby na rywalizację ze znanymi sobie bezpośrednio użytkownikami.

Zwiększenie dostępności

System mógłby być dostosowany tak, aby spełniać wymagania standardu WCAG 2.1 na poziomie AAA. Obecnie rozwiązane w notatkach rozwiązanie odczytywania tekstu z ekranu mogłoby zostać również zastosowane w innych miejscach.

Należy też zwrócić uwagę na komentarze zgłoszone przez seniorów w trakcie badania środowiskowego, większość ze zgłoszonych błędów czy też drobnych poprawek została naprawiona jeszcze przed publikacją rozwiązania, jednakże część z nich wymaga dalszej pracy.

9.3. Perspektywy wdrożenia

Wdrożenie systemu jest możliwe, jednakże problematyczne ze względu na wymagania prawne. Dane dotyczące zażywanych leków są traktowane jako dane medyczne, których przetwarzanie jest ściśle regulowane, wymaga dodatkowych zabezpieczeń i zgód [22].

Dla projektu nie przewidziano również sposobu monetyzacji, co byłoby niezbędne do utrzymania kosztów operacyjnych systemu, w tym kosztów utrzymania infrastruktury, czy też opłat za ekspertyzy prawne i audyty bezpieczeństwa i dostępności.

W związku z powyższym, autorzy pracy nie widzą możliwości na samodzielne wdrożenie systemu, w jego obecnej formie. Potencjalnym rozwiązaniem jest jednak sprzedaż rozwiązania bądź przekazanie go organizacji non-profit.

DOKUMENTACJA UŻYTKOWNIKA

10. Wprowadzenie

Aplikacja mobilna Senso jest przeznaczona dla osób starszych, które poszukują nowoczesnych rozwiązań trudów życia codziennego. A także dla opiekunów osób starszych, którzy chcą aktywnie uczestniczyć w życiu seniorów i sprawić, aby ich życie było jeszcze łatwiejsze, dzięki uproszczeniu komunikacji między nimi. Aplikacja pozwala na zarządzanie lekami, przypominanie o ich regularnym zażywaniu, czy śledzeniu ilości posiadanych leków. Aplikacja posiada system alarmowania opiekunów np. w przypadku zapomnienia o zażyciu leku, czy sytuacji alarmowej zgłoszonej przez seniora. Aplikacja pozwala na prowadzenie notatek, które opcjonalnie mogą być odczytywane i edytowane przez opiekunów. Senso posiada również gry, które mają na celu stymulowanie umysłu seniora i sprawiają, że codzienne życie staje się jeszcze ciekawsze i bardziej urozmaicone. Ważną cechą aplikacji jest rozbudowane zarządzanie profilem seniora, przez opiekunów, co pozwala na dostosowanie aplikacji do potrzeb seniora i sprawne – zdalne – rozwiązywanie napotkanych przez osobę starszą problemów i trudności.

11. Instalacja produktu programowego

Zainstalowanie aplikacji na platformie android wymaga pobrania pliku `.apk` z ostatniej wersji aplikacji. Aby zainstalować aplikację pobraną manualnie – spoza sklepu Google Play – należy włączyć opcję instalacji z nieznanych źródeł w ustawieniach systemu Android [23]. A następnie wybrać pobrany plik `.apk` i zainstalować aplikację.

W przypadku systemu iOS należy pobrać plik `.ipa`, a następnie wybrać zainstalowany plik i go uruchomić, należy jednak pamiętać, że aby zainstalować aplikację z nieznanego źródła, należy włączyć tryb deweloperski na urządzeniu [24]. Co wymaga zapisania się do programu deweloperskiego Apple [25].

12. Wymagania systemowe

Aplikacja dostępna jest na urządzenia mobilne z systemem Android w wersji 5.0 lub nowszej i iOS w wersji 13.0 lub nowszej. Aplikacja nie została opublikowana w sklepie Google Play ani App Store, więc wymagana jest manualna instalacja. Ze względu na ograniczenia systemu iOS, aplikacja nie może być zainstalowana na urządzeniach z systemem iOS, które nie są w trybie deweloperskim [24]. Na urządzeniach z systemem Android nie występuje taka blokada i wymaga jedynie włączenia opcji instalacji z nieznanych źródeł [23].

13. Opis realizacji typowych zadań

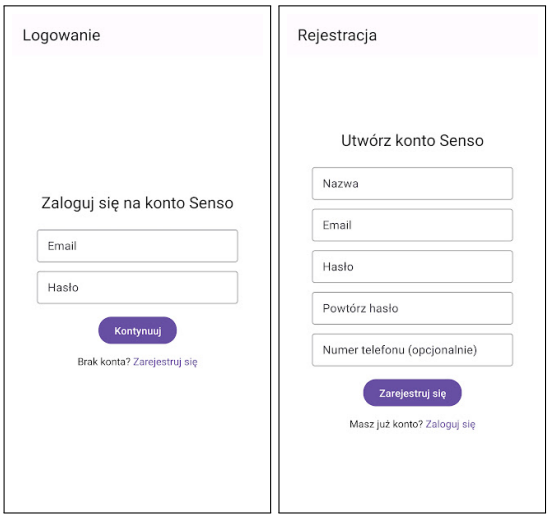
Typowe zadania w aplikacji zostały opisane poniżej w formie kroków do wykonania. W przypadku braku opisu kroku zostało uznane, że jest on wystarczająco intuicyjny. Przy każdym zadaniu zostały umieszczone zrzuty ekranu, stanowiące jednocześnie dokumentację projektu interfejsu użytkownika.

Zalogowanie się

1. Otwórz aplikację mobilną.
2. Wprowadź poprawny adres e-mail i hasło.
3. Kliknij przycisk „Kontynuuj”.

Zarejestrowanie się

1. Otwórz aplikację mobilną.
2. Kliknij link „Zarejestruj się” obok napisu „Brak konta?”.
3. Wprowadź poprawny adres imię, e-mail, hasło oraz opcjonalnie numer telefonu.
4. Kliknij przycisk „Zarejestruj się”.

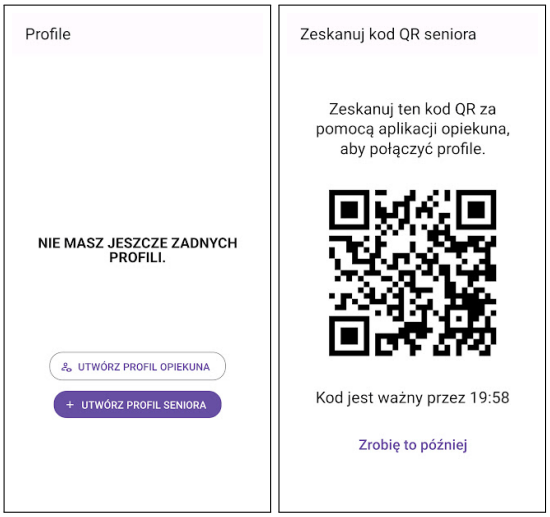


Rysunek 28: Ekrany dla zadań „Zalogowanie się” i „Zarejestrowanie się”.

Utworzenie profilu seniora

Uznaje się, że użytkownik ogląda listę profili.

1. Wybierz przycisk „Utwórz profil seniora”.
2. Opcjonalnie połącz profil seniora z kontem opiekuna przy pomocy kodu QR.

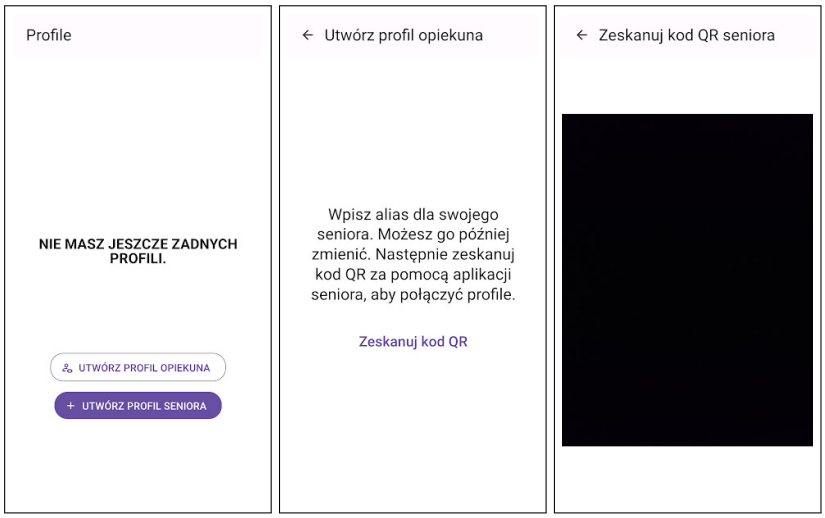


Rysunek 29: Ekrany dla zadania „Utworzenie profilu seniora”.

Utworzenie profilu opiekuna

Uznaje się, że użytkownik ogląda listę profili.

1. Wybierz przycisk „Utwórz profil opiekuna”.
2. Zeskanuj kod QR z aplikacji seniora. Kod QR znajduje się w sekcji „Paruj z opiekunem”.

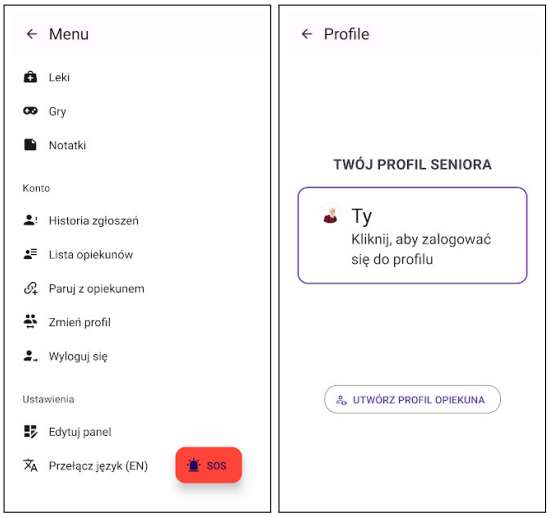


Rysunek 30: Ekrany dla zadania „Utworzenie profilu opiekuna”.

Zmiana profilu

Uznaje się, że użytkownik ogląda menu.

1. Wybierz przycisk „Zmień profil”.
2. Wybierz profil, na który chcesz się przełączyć.

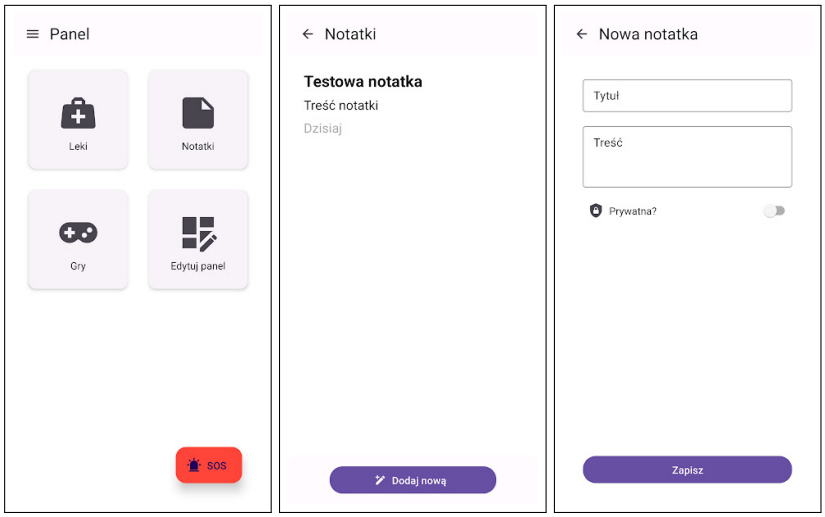


Rysunek 31: Ekrany dla zadania „Zmiana profilu”.

Dodanie notatki

Uznaje się, że użytkownik ogląda panel lub menu.

1. Wybierz przycisk „Notatki” w panelu lub menu.
2. Wybierz przycisk „Dodaj nową”.
3. Wprowadź tytuł i treść notatki i kliknij przycisk „Zapisz”.

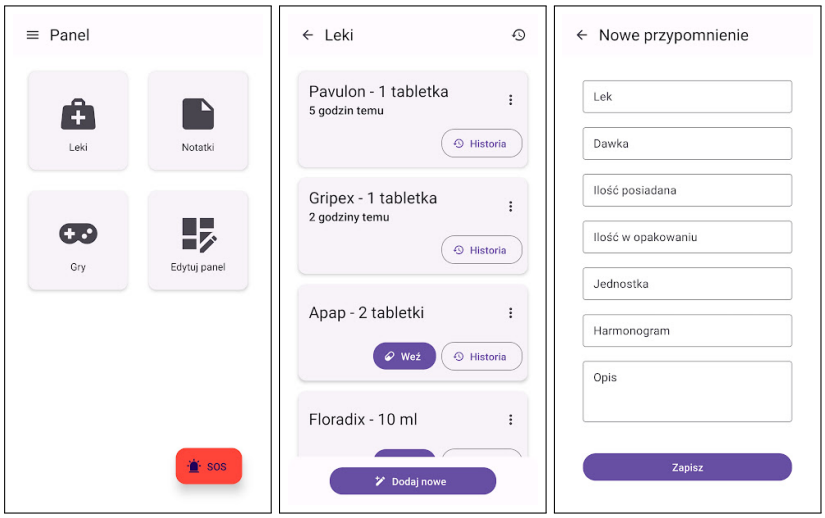


Rysunek 32: Ekrany dla zadania „Dodanie notatki”.

Dodanie leku

Uznaje się, że użytkownik ogląda panel lub menu.

1. Wybierz przycisk „Leki”.
2. Wybierz przycisk „Dodaj nowe”.
3. Uzupełnij wszystkie dane w formularzu i kliknij „Zapisz”.

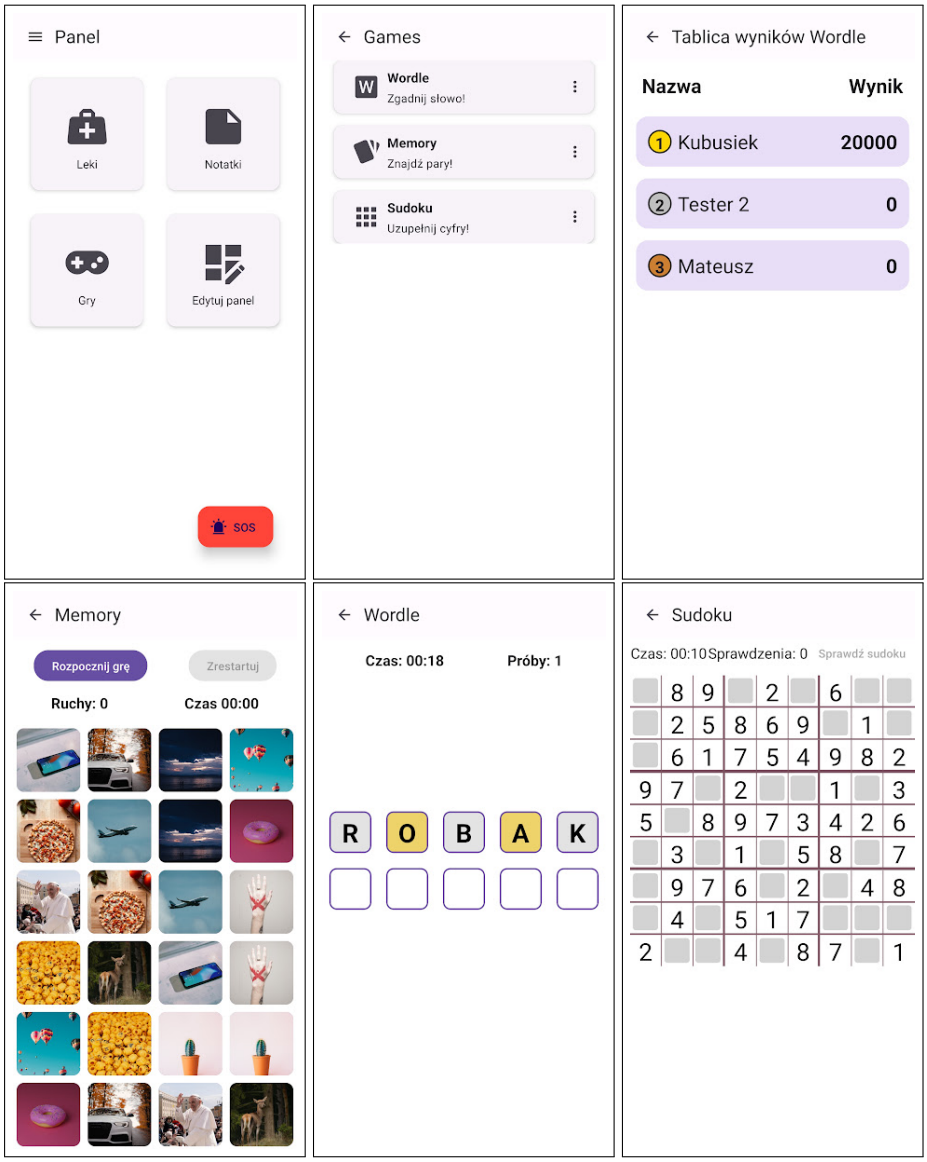


Rysunek 33: Ekrany dla zadania „Dodanie leku”.

Zagranie w grę i sprawdzenie wyników

Uznaje się, że użytkownik ogląda panel lub menu.

1. Wybierz przycisk „Gry”.
2. Wybierz grę, w którą chcesz zagrać.
3. Po zakończeniu gry, w panelu gier wybierz z menu opcję „Tablica wyników” przy grze, w której chcesz sprawdzić wyniki.

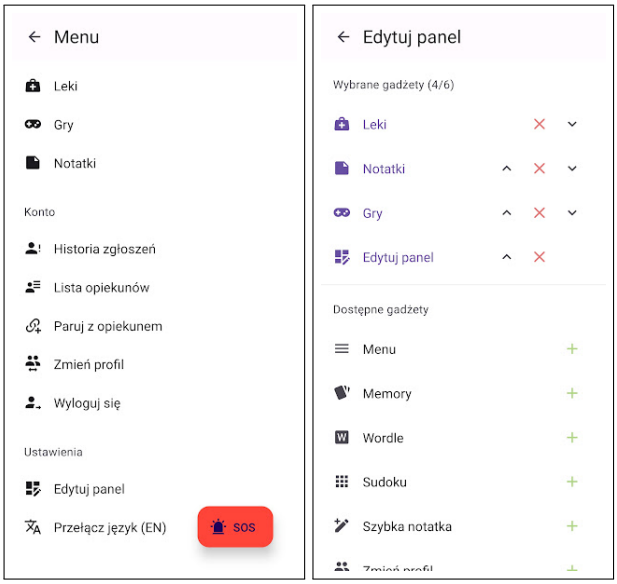


Rysunek 34: Ekrany dla zadania „Zagranie w grę i sprawdzenie wyników”.

Edycja gadżetów w panelu

Uznaje się, że użytkownik ogląda panel lub menu.

1. Wybierz przycisk „Edytuj panel”.
2. Dodaj, usuń lub zmień kolejność wyświetlanych gadżetów za pomocą ikon. Pamiętaj, że możesz wybrać maksymalnie 6 gadżetów z listy.

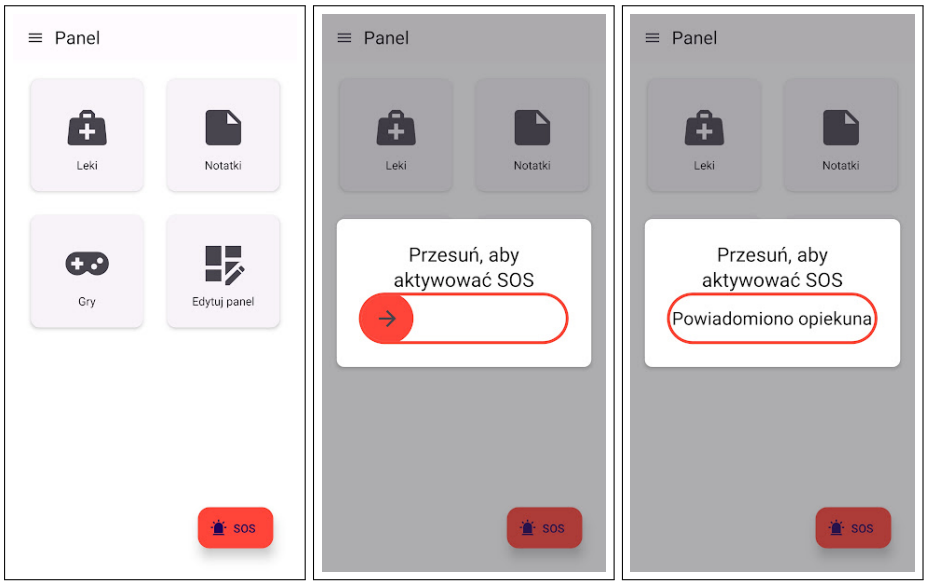


Rysunek 35: Ekrany dla zadania „Edycja gadżetów w panelu”.

Wysłanie sygnału SOS

Uznaje się, że użytkownik ma wybrany profil seniora i ogląda panel lub menu.

1. Kliknij przycisk „SOS”.
2. Potwierdź wysłanie sygnału SOS, przeciągając suwak.

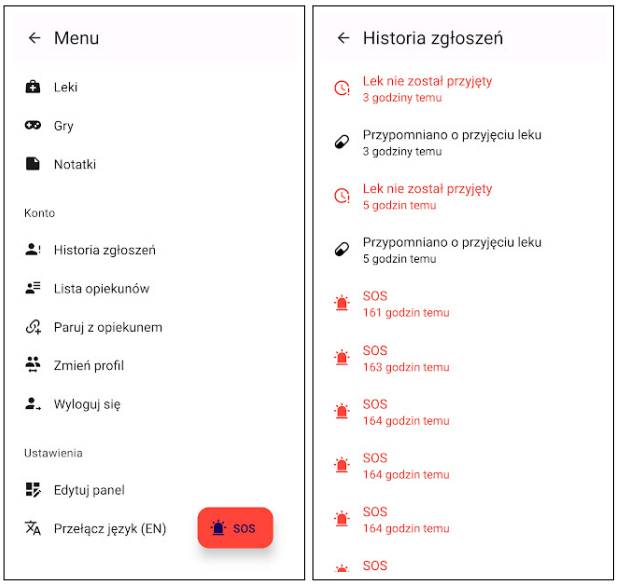


Rysunek 36: Ekrany dla zadania „Wysłanie sygnału SOS”.

Wyświetlenie historii alertów

Uznaje się, że użytkownik ogląda menu.

1. Kliknij przycisk „Historia zgłoszeń”.



Rysunek 37: Ekrany dla zadania „Wyświetlenie historii alertów”.

Bibliografia

[1] *TOP 5 aplikacji dla seniora. Co powinno znaleźć się na jego smartfonie?* Ortomedico.pl. 2023. URL: https://ortomedico.pl/blog/dla-seniora-i-jego-opiekuna/top-5-aplikacji-dla-seniora-co-powinno-znalezc-sie-na-jego-smartfonie (term. wiz. 10. 12. 2023).

[2] Anna Wróblewska-Zawadzka. *12 najlepszych aplikacji (nie)tylko dla seniorów*. Sektor 3.0. 2023. URL: https://sektor3-0.pl/blog/aplikacje-dla-seniorow (term. wiz. 10. 12. 2023).

[3] Wenhao Wu. „React Native vs Flutter, Cross-platforms mobile application frameworks". W: (2018).

[4] Veikko Lintujärvi. „Backend designing with Entity Framework Core". Tampere University of Applied Sciences, 2019.

[5] Wisal Khan i in. „SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review". W: *Big Data and Cognitive Computing* 7.2 (2023), s. 97.

[6] *GitHub docs – About Projects*. GitHub. 2023. URL: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects (term. wiz. 13. 12. 2023).

[7] *GitHub Actions documentation*. GitHub. 2023. URL: https://docs.github.com/en/actions (term. wiz. 13. 12. 2023).

[8] Waldemar Łabuda. „Podejście zwinne a tradycyjne do projektów wytwarzania oprogramowania". W: *Zeszyty Naukowe Warszawskiej Wyższej Szkoły Informatyki* (2015).

[9] Konrad Gos i Wojciech Zabierowski. „The comparison of microservice and monolithic architecture". W: *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE. 2020.

[10] Omar Al-Debagy i Peter Martinek. „A comparative review of microservices and monolithic architectures". W: *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE. 2018.

[11] Daniel Dusiński. „Wzorce projektowe w .NET". W: *Software 2.0* (2004).

[12] *Common web application architectures*. Microsoft. 2023. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (term. wiz. 11. 12. 2023).

[13] Jason Taylor. „Clean architecture with ASP.NET Core 3.0". W: GOTO Copenhagen. 2019. URL: https://gotocph.com/2019/sessions/984/clean-architecture-with-asp-net-core-3-0 (term. wiz. 11. 12. 2023).

[14] Ian Michael Ratner i Jack Harvey. „Vertical Slicing: Smaller is Better". W: *2011 Agile Conference*. IEEE. 2011.

[15] Paul Stovell. *Horizontal vs. Vertical Project Structure*. 2010. URL: https://paulstovell.com/horizontal-vertical-project-structure (term. wiz. 12. 12. 2023).

[16] Prosper Evergreen. „Selecting a state management stragegy for modern web frontend applications". Tampere University of Applied Sciences, 2023.

[17] *ESLint – Core Concepts*. OpenJS Foundation. URL: https://eslint.org/docs/latest/use/core-concepts (term. wiz. 12. 12. 2023).

[18] Sven Slootweg. *Stop using JWT for sessions*. 2016. URL: http://cryto.net/%7Ejoepie91/blog/2016/06/13/stop-using-jwt-for-sessions (term. wiz. 13. 12. 2023).

[19] Sven Slootweg. *Stop using JWT for sessions, part 2: Why your solution doesn't work*. 2016. URL: http://cryto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why-your-solution-doesnt-work (term. wiz. 13. 12. 2023).

[20] *Hangfire – Background jobs and workers for .NET and .NET Core*. Hangfire OÜ. 2013. URL: https://www.hangfire.io (term. wiz. 14. 12. 2023).

[21] *Web Content Accessibility Guidelines 2.1*. W3C World Wide Web Consortium. 2023. URL: https://www.w3.org/TR/WCAG21 (term. wiz. 14. 12. 2023).

[22] *Ochrona danych medycznych (cz. I)*. Barta & Kaliński sp. j. URL: https://bartakalinski.pl/artykuly/ochrona-danych-medycznych-cz-i (term. wiz. 14. 12. 2023).

[23] *How to install Unknown Sources applications in Android?* Appaloosa Store. URL: https://www.appaloosa.io/blog/guides/how-to-install-apps-from-unknown-sources-in-android (term. wiz. 15. 12. 2023).

[24] *Enabling Developer Mode on a device*. Apple Inc. URL: https://developer.apple.com/documentation/xcode/enabling-developer-mode-on-a-device (term. wiz. 15. 12. 2023).

[25] *Apple Developer Program*. Apple Inc. URL: https://developer.apple.com/programs (term. wiz. 15. 12. 2023).

Spis rysunków

Spis tabel

Spis listingów