



**Politechnika
Śląska**

PROJEKT INŻYNIERSKI

„System wspomagający zarządzanie codziennymi aktywnościami”

Patryk Czakon
Nr albumu pc305960

Kierunek: Informatyka
Specjalność: Bazy Danych i Inżynieria Systemów

PROWADZĄCY PRACĘ
Dr hab. Inż. Bożena Małysiak-Mrozek
KATEDRA Systemów Rozproszonych i Urządzeń Informatyki
Wydział Automatyki, Elektroniki i Informatyki

OPIEKUN, PROMOTOR POMOCNICZY (jeśli został powołany)
<stopień naukowy oraz imię i nazwisko>

GLIWICE 2025

Tytuł pracy:

System wspomagający zarządzanie codziennymi aktywnościami

Streszczenie:

(Streszczenie pracy –odpowiednie pole w systemie APD powinno zawierać kopię tego streszczenia. Streszczenie, wraz ze słowami kluczowymi.)

Słowa kluczowe:

(2-5 słów (fraz) kluczowych, oddzielonych przecinkami)

Thesis title:

System supporting management of daily activities

Abstract:

(Thesis abstract – to be copied into an appropriate field during electronic submission, in English.)

Keywords:

(2-5 keywords, separated with commas, in English.)

Spis treści

Rozdział 1	Wstęp.....	1
Rozdział 2	Analiza tematu.....	3
2.1.	Istniejące rozwiązania rynkowe.....	4
Rozdział 3	Wymagania i narzędzia	8
3.1.	. Wymagania funkcjonalne	8
3.2.	Wymagania нефункционалне	11
3.3.	Diagram przypadków użycia	12
3.4.	Wykorzystane narzędzia i technologie	15
3.5.	Metodyka pracy nad projektowaniem i implementacją.....	16
3.6.	Zastosowane algorytmy	16
Rozdział 4	Specyfikacja zewnętrzna	19
4.1.	Wymagania sprzętowe i programowe	19
4.2.	Sposób instalacji	20
4.3.	Sposób aktywacji.....	20
4.4.	Kategorie użytkowników.....	21
4.5.	Sposób obsługi.....	21
4.6.	Administracja systemu	38
4.6.1	Panel Moderacyjny	38
4.6.2.	Panel Administracyjny	40
Rozdział 5	Specyfikacja wewnętrzna	42
5.1.	Przedstawienie idei	42
5.2.	Architektura systemu	42
5.3.	Struktury danych i organizacji bazy danych.....	44
5.4.	Implementacja wybranych fragmentów	47
5.4.1.	Algorytm sugerujący aktywności	47
5.4.2.	Algorytm sugerujący umiejscowienie aktywności	50
5.4.3.	Przykładowy endpoint	52
Rozdział 6	Weryfikacja i walidacja.....	54
Rozdział 7	Podsumowanie i wnioski.....	56
	Bibliografia.....	58
	Spis skrótów i symboli	61
	Lista dodatkowych plików, uzupełniających tekst pracy	62
	Spis rysunków	63

Rozdział 1

Wstęp

Współczesne tempo życia sprawia, że sprawne zarządzanie czasem własnym stało się cenną umiejętnością. Dlatego rosnące na popularności są wszelakiego rodzaju rozwiązania ułatwiające taką organizację i planowanie. Tradycyjne kalendarze i dzienniki nie oferują funkcjonalności analizujących zachowania użytkownika czy jakiegokolwiek zintegrowania z osobami trzecimi. W związku z powyższym użytkownicy poszukują wygodnych w użyciu narzędzi dzięki którym będą mieli możliwość uprościć sobie zarządzanie obowiązkami, analizować własne nawyki, oraz mieć możliwość synchronizacji z osobami trzecimi.

Jedną z najpopularniejszych odpowiedzi na takie problemy są systemy informatyczne klasy web, które oferują wsparcie w organizacji i zarządzaniu czasem własnego. Systemy tego typu wykorzystują technologie internetowe, działające poprzez przeglądarkę internetową na dowolnym nowoczesnym urządzeniu, w większości oparte o architekturę klient – serwer. Dostępne z dowolnego miejsca o ile posiadamy połączenie internetowe, pozwalają nie tylko na zwykłe zarządzanie, ale na interakcję i integrację czynności z innymi użytkownikami czy grupami osób bez konieczności uprzedniego fizycznego spotkania. Dodatkowym atutem rozwiązań internetowych jest możliwość przechowywania danych na serwerze w chmurze dzięki czemu użytkownik nie straci dostępu do swoich danych nawet przy utracie urządzenia.

Celem pracy inżynierskiej jest zaprojektowanie i implementacja systemu webowego skierowanego do osób prywatnych umożliwiającego łatwą organizację zadań i obowiązków użytkownika. System ma umożliwiać zakładanie przez użytkowników kont a następnie tworzenie, modyfikowanie, usuwanie i wizualizację wydarzeń na interaktywnym kalendarzu. Analizować przeszłe zachowania użytkownika i oferować spersonalizowane sugestie kolejnych działań oraz pomagać w umiejscawianiu nowych aktywności. Kluczowym założeniem jest również możliwość integracji z innymi użytkownikami – poprzez listę znajomych i współdzielenia z nimi wybranych aktywności. Dodatkowymi funkcjonalnościami jest obsługa tworzenia powiadomień, możliwość ekstrakcji osi czasu do formatów drukowalnych, jak i wgląd w historię aktywności w określonym przez użytkownika okresie. Dostęp do aplikacji może mieć wielu użytkowników jednocześnie oraz

obsługuje poziomy dostęp do funkcjonalności, aby tylko konta posiadające odpowiednie uprawnienia mogły np. usuwać nie stosowne treści. Cała aplikacja została opracowana w intuicyjny, responsywny interfejs, aby prawidłowo wyświetlała się na każdym urządzeniu.

Zakres pracy obejmuje:

- zaprojektowanie i implementację części serwerowej
- zaprojektowanie i implementację części klienckiej
- implementację systemu autoryzacji oraz stopni dostępu
- zaprojektowanie i implementację algorytmu sugerującego umiejscowienie aktywności
- zaprojektowanie i implementację algorytmu sugerującego aktywność na podstawie historii wydarzeń użytkownika

Całość pracy została podzielona na 7 następujących rozdziałów:

1. Wprowadzenie – wstęp do tematu, omówienie problematyki, obecnych rozwiązań oraz celu projektu.
2. Analiza tematu – sformułowanie problemu, osadzenie rozwiązania w kontekście obecnego rynku, omówienie algorytmów.
3. Wymagania i narzędzia – spis wymagań funkcjonalnych i нефункциональных, diagramy UML, wykorzystane technologie, metodyka pracy i implementacji projektu.
4. Specyfikacja zewnętrzna – spis wymagań sprzętowych i programowych, sposób instalacji oraz uruchamiania, sposób obsługi programu, panel administracyjny.
5. Specyfikacja wewnętrzna – przedstawienie idei, architektura systemu, struktury danych i organizacja bazy danych oraz implementacja wybranych fragmentów.
6. Weryfikacja i walidacja – opis sposobów testowania i walidacji aplikacji.
7. Podsumowania i wnioski

Rozdział 2

Analiza tematu

Problem braku czasu jest jednym z kluczowych zmagień, często borykamy się z nadmiarem obowiązków w pracy, studiach, szkole jak i w życiu prywatnym. Skutkiem tego często bywa zwiększony współczynnik stresu, a co za tym idzie – gorsze samopoczucie. Umiejętne zarządzanie czasem wpływa na ogólny dobrostan, w szczególności na satysfakcję z życia [1][3], dodatkowo poprawia samodyscyplinę [2]. W regionach życia takich jak praca, nauka można zauważyć poprawę w zaangażowaniu i efektach [2][4]. Jasno zdefiniowane ramy czasowe mogą działać motywująco i skłaniać do systematycznej pracy oraz redukować zjawisko prokrastynacji.

Rozwój technologii webowych i mobilnych spowodował, że to właśnie aplikacje oferujące dostęp z urządzeń mobilnych i desktopowych są obecnie najczęściej wybieranym rozwiązaniem na rynku. Największymi atutami jest prostota dostępu i intuicyjna wizualizacja oraz możliwość interakcji z innymi użytkownikami. Dzięki nowoczesnym narzędziom do responsywnego projektowania aplikacje oferują najlepsze doświadczenie dla użytkownika [5]. Minusem takich rozwiązań jest fakt, iż nie są darmowe w porównaniu do tradycyjnych form organizacji czasu. Dostępne darmowe rozwiązania zazwyczaj posiadają ograniczenia funkcjonalności bądź agresywne reklamy co czyni je mniej pożądanymi przez użytkowników. Struktura większości aplikacji składa się z dwóch głównych części:

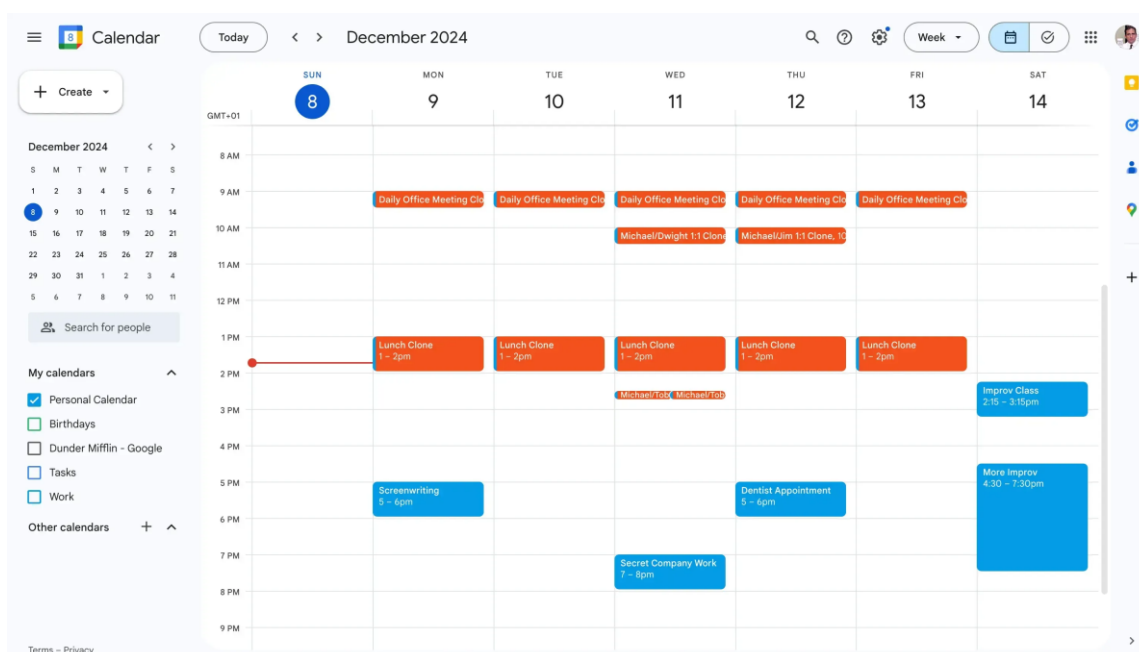
- frontendowej – tej którą widzą użytkownicy i która odpowiada za komunikację między użytkownikiem a serwerem.
- backendowej – czyli serwera i bazy danych.

Dodatkowo nieodłącznym elementem stały się serwisy chmurowe odpowiedzialne za synchronizację i dostęp do danych z wielu urządzeń. Wraz z biegiem czasu market rozbił się na wyspecjalizowane aplikacje zarządzające czasem i zadaniami skierowane do osób prywatnych i osobno dla firm, gdzie zazwyczaj wymagane jest więcej funkcjonalności i obsługa dużych ilości osób w ramach organizacji. Do najpopularniejszych dostępnych dla osób prywatnych rozwiązań należą aplikacje takie jak Google Calendar, Todoist czy Trello.

2.1. Istniejące rozwiązania rynkowe

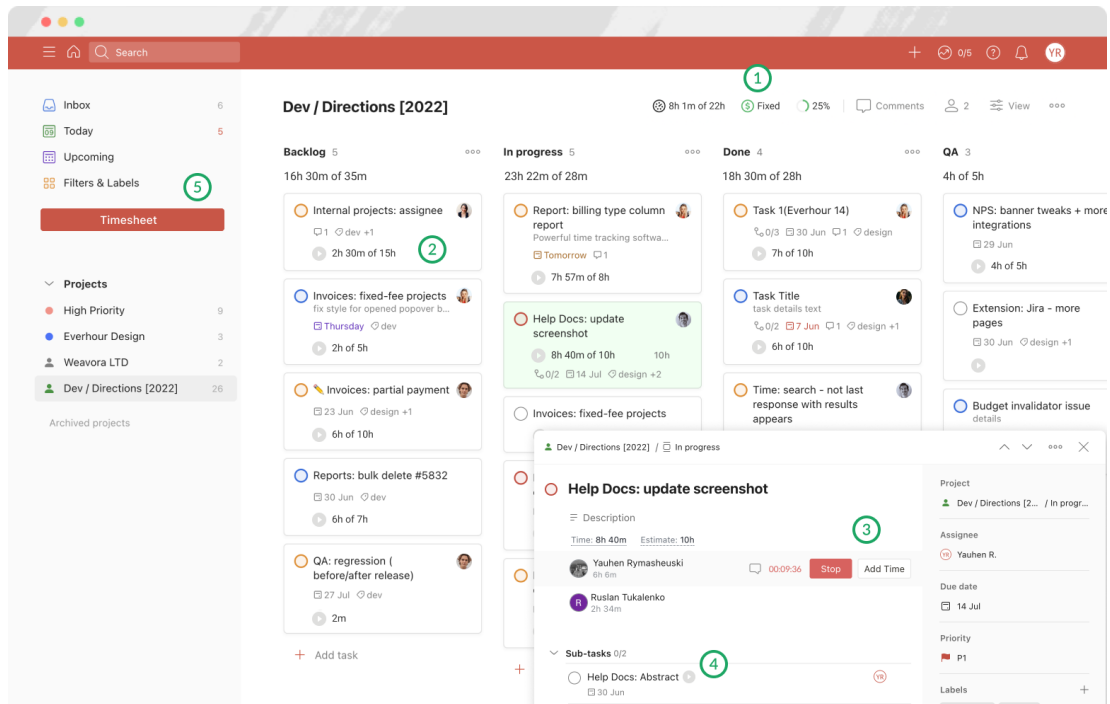
Na obecnym rynku dostępna jest niezliczona ilość aplikacji oferujących funkcjonalności organizacyjne, podrozdział ten jest dedykowany zestawieniu najpopularniejszych dostępnych rozwiązań dominujących wspomnianą kategorię.

Google Calendar [6] – aplikacja (Rysunek 2.1.1) firmy google oferująca proste udostępnianie zdarzeń innym użytkownikom jak i integrację z innymi serwisami tej firmy. Istnieją jednak limity pod względem zaawansowanych funkcji zadaniowych czy mały stopień personalizacji interfejsu.



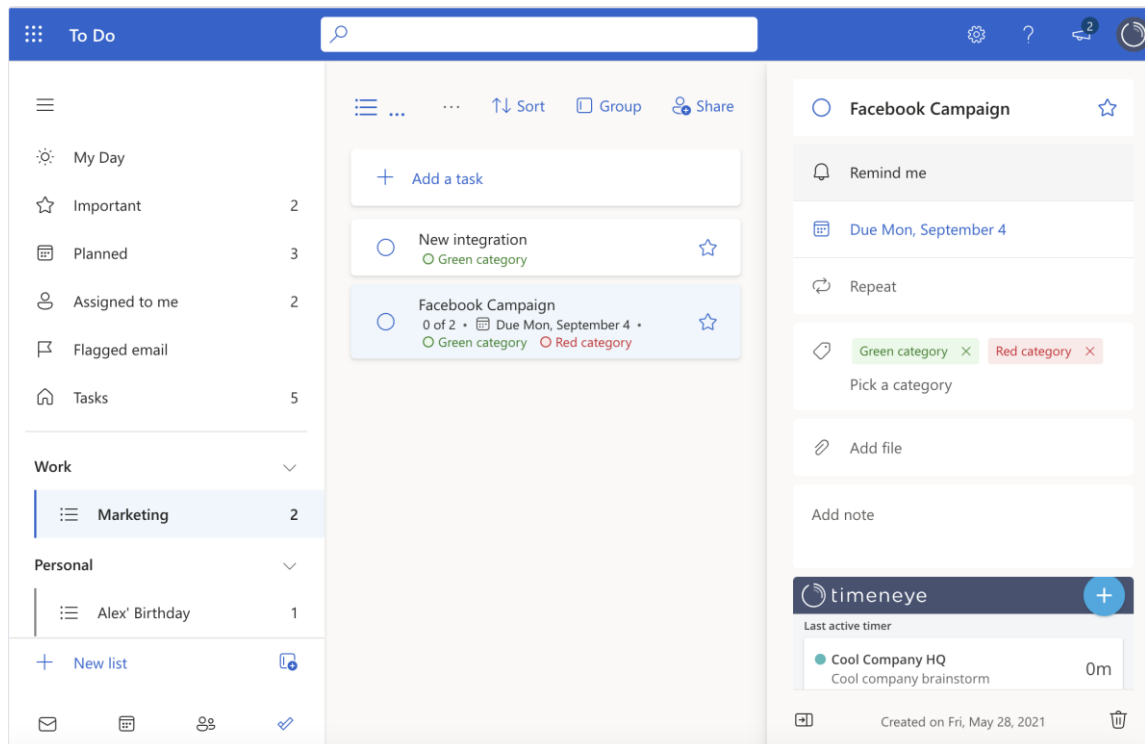
Rysunek 2.1.1 Interfejs aplikacji „Google Calendar”

Todoist [7] – aplikacja (Rysunek 2.1.2) firmy doist oferująca narzędzia obsługujące dodawanie zadań, ustawianie przypomnień i tworzenie projektów, wspiera integrację z wieloma aplikacjami używanymi przez korporacje takimi jak Slack, Google Calendar. Aplikacji brakuje zaawansowanych opcji do raportowania i w wersji darmowej część funkcjonalności jest niedostępna.



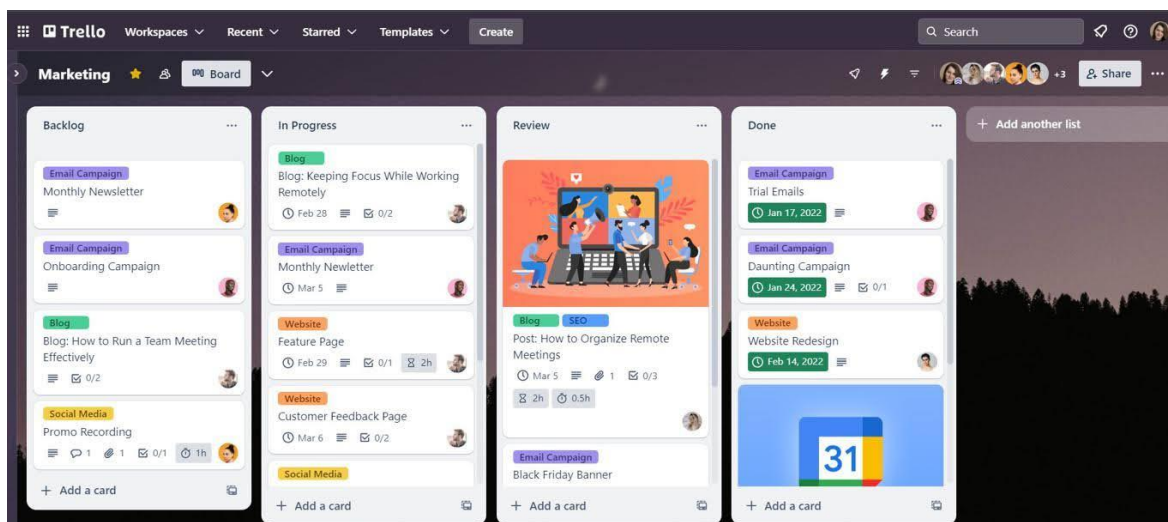
Rysunek 2.1.2 Interfejs Aplikacji „Todoist”

Microsoft To-Do [8] – aplikacja (Rysunek 2.1.3) firmy Microsoft oferująca proste narzędzia do tworzenia zdarzeń, oferuje automatyczną synchronizację z pocztą Outlook. Narzędzie jest stosunkowo proste i brakuje nim bardziej zaawansowanych funkcji, współpraca z innymi użytkownikami również jest ograniczona.



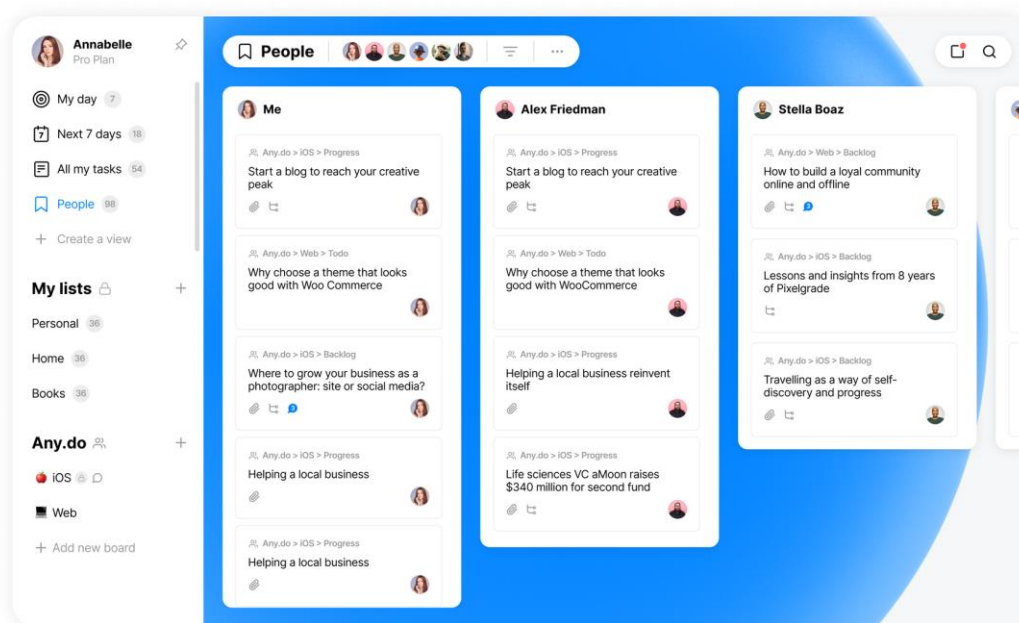
Rysunek 2.1.3 Interfejs Aplikacji „Microsoft To-Do”

Trello [9] – aplikacja (Rysunek 2.1.4) do zarządzania zadaniami i projektami na metodzie Kanban. Oferuje intuicyjny interfejs i dużą elastyczność tworzenia. Duża liczba zadań czy tablic skutkuje bardzo długimi tablicami przez co staje się mniej intuicyjna. Niestety niektóre podstawowe funkcje, takie jak widok kalendarzowy nie są dostępne w darmowej wersji aplikacji.



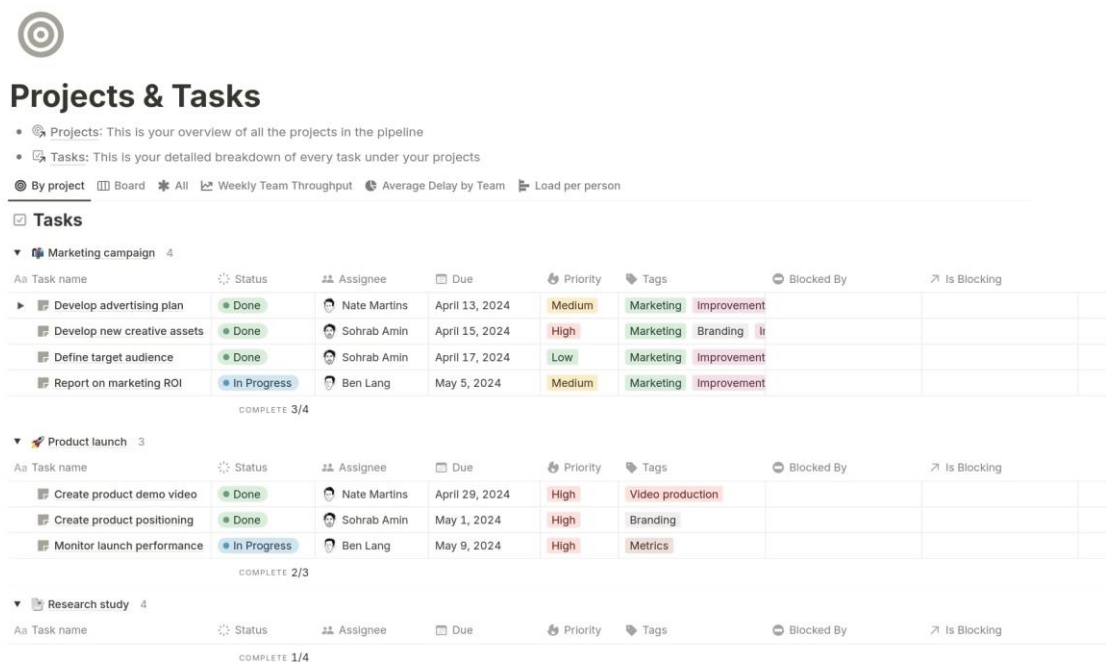
Rysunek 2.1.4 Interfejs aplikacji „Trello”

Any.do [10] – aplikacja (Rysunek 2.1.5) do organizacji zadań, oferuje takie funkcjonalności jak listy zadań, przypomnienia i asystenta planowania. Posiada bardzo minimalistyczny i prosty interfejs przez co trafia w gusta wielu użytkowników.



Rysunek 2.1.5 Interfejs aplikacji „Any.do”

Notion [11] – aplikacja (Rysunek 2.1.6) posiadająca wiele zastosowań, pozwalająca na prowadzenie notatek, zadań, tworzenie kalendarzy, projektów. Bardzo zaawansowany system do monitorowania prywatnego życia, cechuje się ogromną elastycznością – użytkownik może projektować własne kalendarze, systemy planowania, panel podsumowania czy nawet bazy danych powiązane ze sobą. Wynikiem dużej ilości opcji jest konieczność nauczenia się interfejsu i funkcjonalności, dodatkowo przy dużej ilości danych aplikacja potrzebuje dużo zasobów, aby działać płynnie.



Rysunek 2.1.6 Interfejs aplikacji „Notion”

Rozdział 3

Wymagania i narzędzia

W tym rozdziale przedstawione są wymagania funkcjonalne oraz нефункционалне, wraz z diagramami przypadków, wykorzystanymi narzędziami, technologiami. Dodatkowo opisane metody modelowania podczas projektowania oraz implementacji systemu wraz z teoretycznym opisem używanych w aplikacji algorytmów.

3.1. Wymagania funkcjonalne

System zakłada spełnianie następujących wymagań funkcjonalnych.

Funkcjonalności konta użytkownika

- System musi umożliwiać rejestrację konta z użyciem adresu email oraz hasła.
- System musi umożliwiać logowanie oraz wylogowywanie użytkownika.
- System musi umożliwiać zmianę danych użytkownika.
- System musi umożliwiać modyfikację ustawień prywatności użytkownika.
- System musi umożliwiać permanentne usunięcie konta użytkownika.

Funkcjonalności związane z aktywnościami

- System musi umożliwiać tworzenie aktywności
- System musi umożliwiać edycję aktywności
- System musi umożliwiać tworzenie zasad powtarzalności dla aktywności
- System musi umożliwiać tworzenie pojedynczych instancji aktywności.
- System musi umożliwiać tworzenie wyjątków od reguł powtarzalności.

- System musi umożliwiać usuwanie aktywności.
- System musi umożliwiać przekonwertowanie aktywności na publiczną.
- System musi umożliwiać przekonwertowanie aktywności na tylko-znajomi.

Funkcjonalności związane z aktywnościami online

- System musi umożliwiać przekonwertowanie aktywności na prywatną.
- System musi umożliwiać wysłanie zaproszeń do aktywności znajomym.
- System musi umożliwiać odrzucenie/akceptację zaproszenia do aktywności.
- System musi umożliwiać wgląd w listę uczestników danej aktywności.
- System musi umożliwiać wyrzucenie danego użytkownika z aktywności.
- System musi umożliwiać zablokowanie możliwości ponownego dołączenia.
- System musi obsługiwać dołączanie do aktywności poprzez kod dołączenia.
- System musi obsługiwać podgląd do wysłanych, otrzymanych zaproszeń do aktywności.
- System musi obsługiwać możliwość cofnięcia zaproszenia.

Funkcjonalności części socjalnej aplikacji

- System musi obsługiwać możliwość wysłania zaproszenia do znajomych.
- System musi obsługiwać akceptację, odrzucenie zaproszenia do znajomych.
- System musi obsługiwać cofnięcie wysłania zaproszenia do znajomych.
- System musi obsługiwać podgląd listy znajomych.
- System musi obsługiwać podgląd wysłanych i otrzymanych zaproszeń.
- System musi obsługiwać podgląd profilu znajomego.
- System musi obsługiwać usunięcie znajomego.
- System musi obsługiwać zablokowanie użytkownika.
- System musi obsługiwać odblokowanie użytkownika.
- System musi obsługiwać podgląd zablokowanych użytkowników.
- System musi obsługiwać utworzenie powiadomienia.

Funkcjonalności systemu powiadomień

- System musi obsługiwać edycję powiadomienia.
- System musi obsługiwać usunięcie powiadomienia.
- System musi obsługiwać oznaczenie powiadomienia jako przeczytane.
- System musi obsługiwać masowe usuwanie przeczytanych powiadomień.

Funkcjonalności administracji systemu

- System musi umożliwiać modyfikację danych użytkownika moderatorom i administracji.
- System musi umożliwiać usunięcie danych użytkownika moderatorom i administracji.
- System musi obsługiwać możliwość utworzenia konta moderatora przez administratora.
- System musi obsługiwać permanentne usuwanie kont użytkowników przez administratora.
- System musi obsługiwać permanentne usuwanie aktywności użytkowników przez administratora.
- System musi umożliwiać administratorowi wgląd w historię aktywności moderacyjnych.
- System musi wyświetlać kontom moderacyjnym i administracyjnym odpowiedni panel.

Inne funkcjonalności implementowane przez system

- System musi obsługiwać widok tygodniowy osi czasu.
- System musi obsługiwać możliwość podglądu statystyk w danym czasie.
- System musi obsługiwać możliwość eksportu danego tygodnia do pdf.
- System musi obsługiwać zmianę motywu aplikacji.
- System musi oferować możliwość zasugerowania umiejscowienia aktywności.
- System musi oferować możliwość zaproponowania aktywności.
- System musi ograniczać funkcjonalności na podstawie ról użytkowników.

3.2 Wymagania niefunkcjonalne

System zakłada spełnianie następujących wymagań niefunkcjonalnych.

Wydajność

- Czas ładowania strony poniżej jednej sekundy.
- Czas wykonania endpointów poniżej jednej sekundy.

Skalowalność

- Dostęp do aplikacji przez wielu użytkowników jednocześnie.
- Możliwość dodawania nowych funkcjonalności bez wpływu na obecne działanie.

Bezpieczeństwo

- Uwierzytelnianie użytkowników za pomocą tokenu JWT.
- Szyfrowanie haseł za pomocą Bcrypt.
- Autoryzacja RBAC – kontrola dostępu i funkcjonalności ze względu na role.
- Walidacja danych wejściowych.

Dostępność

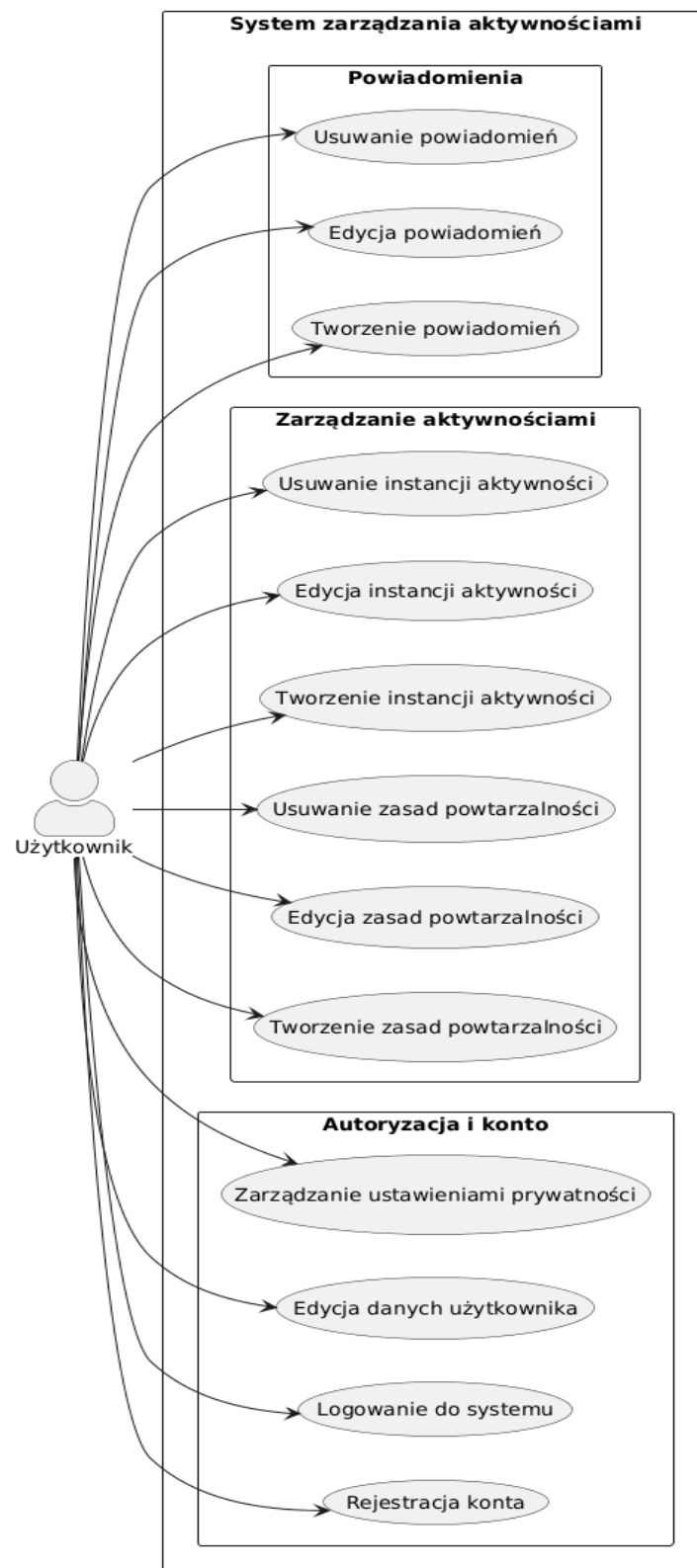
- Responsywny interfejs użytkownika.
- Tryb wysokiego kontrastu interfejsu.
- Dostęp i synchronizacja danych na wielu urządzeniach.

3.3. Diagram przypadków użycia

Diagramy przypadków użycia (Rysunek 3.3.1 oraz 3.3.2) reprezentują funkcjonalności z perspektywy użytkownika, czyli osoby posiadającej konto, realizowane przez niniejszy projekt. Funkcjonalności zostały podzielone na moduły w celu lepszej wizualizacji.

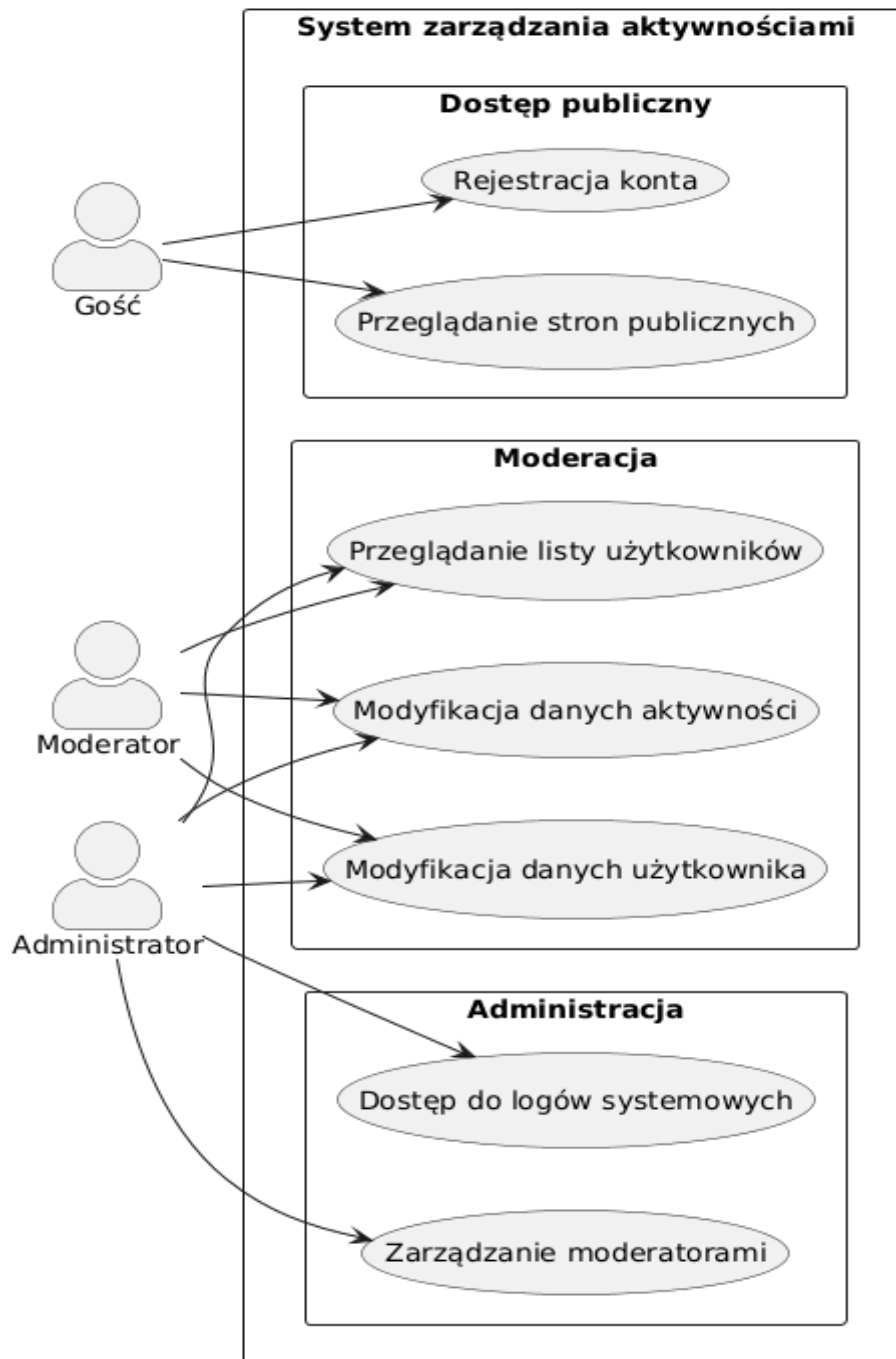


Rysunek 3.3.1 Diagram przypadków użycia użytkownika



Rysunek 3.3.2 Drugi diagram przypadków użycia użytkownika

Diagram funkcjonalności pozostałych ról systemowych (Rysunek 3.3.3) ilustruje możliwe interakcje wszystkich pozostałych aktorów zaimplementowanych w systemie. Gdzie „Gość” to osoba nie posiadająca konta, a „Moderator” oraz „Administrator” to konta z specjalnym dostępem.



Rysunek 3.3.3 Diagram przypadków użycia pozostałych ról systemowych

3.4. Wykorzystane narzędzia i technologie

W niniejszym podrozdziale przedstawione i opisane zostały narzędzia oraz technologie wykorzystane podczas implementacji i projektowania systemu. Dobór zastosowanych rozwiązań został podyktowany wymaganiami systemu oraz niezawodnością rozwiązań.

Środowisko programowe

- Visual Studio Code [12] – rozbudowane środowisko programistyczne. Oferuje wsparcie IntelliSense, podpowiedzi składni i integrację z Git [13]. W projekcie zostało użyte głównie jako środowisko do części frontendowej aplikacji.
- Visual Studio [14] – IDE firmy Microsoft przeznaczone do rozwoju aplikacji .NET [15]. W projekcie zostało użyte jako środowisko do tworzenia backendowej strony aplikacji.
- SQL Server Management Studio (SSMS) [16] – narzędzie firmy Microsoft służące do zarządzania bazą danych SQL server [17].

Część backendowa systemu

- ASP.NET Core, wersja 9.0.4 – framework do tworzenia API. Pełni rolę warstwy logiki biznesowej i komunikacji z bazą danych.
- Swagger wersja 9.0.9 [18] – interfejs dokumentujący istniejące endpointy, umożliwiający proste testowanie.
- Microsoft Entity Framework Core, wersja 9.0.9 [19] – odpowiedzialny za mapowanie encji C# na tabele SQL.
- SQL Server, wersja 9.0.9 – baza danych przechowująca informacje.
- QuestPDF, wersja 2025.7.4 [20] – biblioteka .NET do generacji dokumentów PDF.
- ASP.NET Core JWT Authentication, wersja 9.0.9 [21] – mechanizm autoryzujący oparty o tokeny JWT.

Część frontendowa systemu

- React, wersja 19.1.1 [22] – biblioteka javascriptowa do budowania interfejsów użytkownika, odpowiada za struktury komponentów, logikę widoków i dynamiczne renderowanie danych.
- Vite, wersja 7.1.2 [23] – serwer deweloperski do uruchamiania projektów react.

- Tailwind CSS, wersja 3.4.18 [24] – framework CSS, umożliwia tworzenie responsywnych interfejsów.
- Jwt-decode, wersja 4.0.0 [25] – służące do dekodowania tokenów jwt.
- Lucide-react, wersja 0.554.0 [26] – biblioteka ikon.
- React-router-dom, wersja 7.9.6 [27] – router wykorzystywany do nawigacji między stronami.
- Recharts 3.5.0, wersja [28] – biblioteka do obsługi wykresów

3.5. Metodyka pracy nad projektowaniem i implementacją

Implementacja kolejnych funkcjonalności projektu następowała systematycznie dla każdej funkcjonalności. Aby zapewnić integralność systemu przed wprowadzeniem zmian tworzony był zrzut kodu na poprzez platformę GitHub. Następnie implementacja nowych funkcjonalności i kolejny zrzut kodu. Następnie testowanie i sprawdzenie kodu, po końcowej weryfikacji i wprowadzeniu poprawek następuje kolejny zrzut kodu.

Pierwsza została w pełni napisana część backendowa systemu i przetestowana za pomocą interfejsu swagger. Część frontendowa korzysta tylko z endpointów więc jedyne ewentualne ingerencje w kod backendowy wynikały w potrzeby dodania jakiegoś pola do wyświetlania bądź lekkich zmian przy wyświetlaniu niektórych danych.

3.6. Zastosowane algorytmy

W niniejszej pracy projektowej zaimplementowane zostały algorytmy do sugestii aktywności oraz umiejscowienia aktywności, które korzystają z teorii zbiorów rozmytych (Wzór 1). Zbiór rozmyty A na przestrzeni X to zbiór par:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \quad (1)$$

gdzie:

$\mu_A(x)$ – funkcja przynależności

$\mu_A(x) \in [0,1]$ – stopień przynależności elementu x do zbioru A

Algorytm sugerujący aktywność

Algorytm sugerujący pobiera od użytkownika termin, godziny i ewentualną kategorię następnie za pomocą wag dla zmiennych oblicza stopień dopasowania z puli aktywności użytkownika na jego podstawie przeszłych wydarzeń. Na podstawie liczby wystąpień danej aktywności w historii obliczany jest współczynnik (Wzór 2):

$$stabilityFactor = \min \left(1, \frac{\log_{10}(N + 1)}{2} \right) \quad (2)$$

gdzie:

N – liczba wystąpień w historii

Wynik (Wzór 3) dopasowania na podstawie podanych danych jest obliczany poprzez:

$$Score = (0,6 \cdot \mu_d) + (0,25 \cdot \mu_t) + (0,15 \cdot \mu_a) \quad (3)$$

gdzie:

μ_d – czas trwania aktywności

μ_t – przedział godzinowy aktywności

μ_a – dzień tygodnia

Końcowy wynik dopasowania (Wzór 4) otrzymujemy poprzez pomnożenie przez współczynnik stabilności nawyku:

$$FinalScore = Score \cdot stabilityFactor \quad (4)$$

Algorytm zwraca listę pozycji aktywności jako wynik wraz z wagą z przedziału $[0,1]$.

W aplikacji zaimplementowana została też druga wersja niniejszego algorytmu biorąca pod uwagę przeszłe aktywności z puli 100 losowych użytkowników, którzy wyrazili na to zgodę. Główna różnica polega na dodaniu dodatkowego czynnika przy okazji mnożenia dopasowania (Wzór 3), aby odzwierciedlić ilość wystąpień danej aktywności w społeczności. Zmodyfikowany wzór dopasowania (Wzór 5) prezentuje się następująco:

$$Score = (0,6 \cdot \mu_d) + (0,25 \cdot \mu_t) + (0,15 \cdot \mu_a) + (0,1 \cdot p) \quad (5)$$

gdzie:

μ_d – czas trwania aktywności

μ_t – przedział godzinowy aktywności

μ_a – dzień tygodnia

p – liczba wystąpień aktywności w społeczności. Nasycy się przy 25 wystąpieniach.

Następnie analogicznie liczony jest końcowy wynik dopasowania (Wzór 4). Ten algorytm również zwraca listę pozycji aktywności jako wynik wraz z wagą z przedziału $[0,1]$.

Algorytm sugerujący umiejscowienie aktywności

Kolejnym algorytmem obecnym w projekcie jest algorytm odpowiedzialny za podpowiadanie użytkownikowi umiejscowienia aktywności. Pobierana od użytkownika jest aktywność, termin, ewentualne widełki godzinowe i dzień tygodnia. Na podstawie tych czynników przeszukiwana jest oś czasu użytkownika w celu znalezienia odstępu czasowego równego czasowi trwania aktywności + 20minut buforu. Algorytm zwraca listę takich terminów z propozycją automatycznego utworzenia instancji w wybranym terminie.

Drugi wariant algorytmu sugerującego umiejscowienie umożliwia ingerencję w istniejącą oś czasu. Wyszukuje odstępów między istniejącymi aktywnościami, które są maksymalnie 30minut krótsze aniżeli podany przez użytkownika czas aktywności. Algorytm zwraca listę takich terminów i prosi użytkownika o podanie czasów o które ma skrócić aktywność poprzedzającą, tą którą chcemy wstawić, bądź przesunąć rozpoczęcie następującej. Następnie automatycznie modyfikuje istniejące instancje po podaniu przez użytkownika wartości.

Wartym dodania jest, że w całym projekcie czasy trwania aktywności są adaptacyjne – obliczana jest średnia (Wzór 6) z ostatnich 20 wystąpień danej aktywności:

$$AvgDuration = \frac{1}{N} \sum_{i=1}^N D_i \quad (6)$$

gdzie:

N – liczba ostatnich wystąpień

D_i – czas trwania i-tego wystąpienia

Rozdział 4

Specyfikacja zewnętrzna

W niniejszym rozdziale opisane zostały niezbędne wymagania sprzętowe oraz programowe niezbędne do poprawnego działania aplikacji po stronie serwera jak i klienta. Dodatkowo opisany został sposób instalacji oraz aktywacji aplikacji. Rozdział zawiera również opis ról użytkowników zaimplementowanych w systemie jak i obszernie opisany sposób obsługi z perspektywy użytkownika i osoby administracyjnej.

4.1. Wymagania sprzętowe i programowe

Aby aplikacja działała w pełni poprawie należy spełnić następujące wymagania:

Strona serwerowa

- System operacyjny: Windows 10/11 bądź Windows Server
- Procesor: minimum 4 rdzenie
- Pamięć Ram: minimum 2GB
- Przestrzeń dyskowa: 2GB + rozmiar bazy danych
- Środowisko uruchomieniowe: .NET SDK (ASP.NET Core 9/10)

Strona kliencka

- Dostęp do urządzenia obsługującego przeglądarki: Chrome/Firefox
- Dostęp do internetu

4.2. Sposób instalacji

Niniejszy podrozdział zawiera pełny opis instalacji oraz pierwszego uruchomienia aplikacji.

Strona backendowa systemu

1. Przygotować środowisko .NET SDK zgodny z wersją ASP.NET Core 9/10
2. Pobrać aplikację z repozytorium githubowskiego
3. Przejść do folderu backendowego, wykonać komendę **dotnet restore**
4. W pliku appsettings.json skonfigurować ConnectionString odpowiadający bazie
5. W terminalu wykonać migrację nowej bazy – **dotnet ef migrations add <nazwa>**
6. W terminalu wykonać aktualizację bazy – **dotnet ef database update**
7. Następnie można włączyć aplikację – **dotnet run**

Strona frontendowa systemu

1. Przejść do folderu frontendowego
2. W terminalu wykonać **npm install**
3. Włączyć aplikację poprzez **npm run dev**

Strona backendowa jest dostępna pod adresem localhost:5268, strona frontendowa pod localhost:5173. Aby podejrzeć endpointy można wejść pod localhost:5268/swagger aby za pośrednictwem interfejsu swagger zobaczyć strukturę endpointów.

4.3. Sposób aktywacji

W systemie, przy pierwszym uruchomieniu, domyślnie tworzone jest konto administracyjne z danymi: admin@example.com, admin123!@#. W celu administracji systemem wystarczy wejść pod adres localhost:5173 i zalogować się podanymi danymi, aplikacja automatycznie uruchomi panel administracyjny oferujący dalsze funkcjonalności.

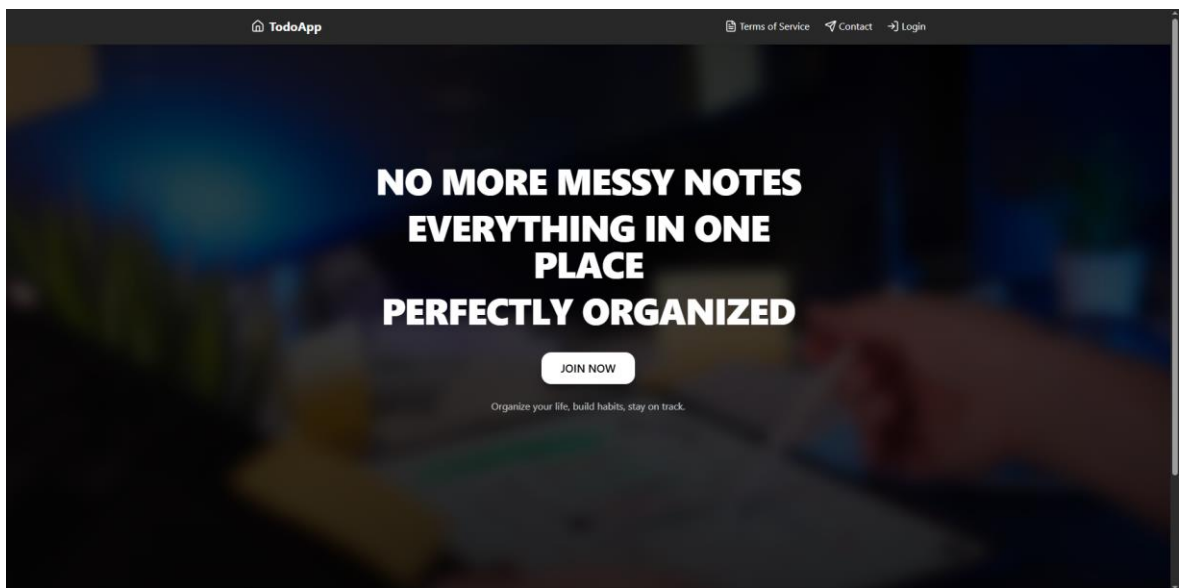
4.4. Kategorie użytkowników

W systemie zostały zaimplementowane następujące kategorie użytkowników:

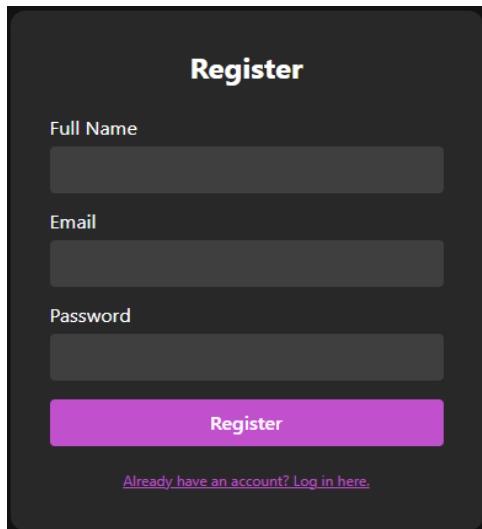
- Gość – użytkownik niezalogowany posiadający możliwość założenia konta i zalogowania się na nie, oraz przeglądania podstawowych stron.
- Użytkownik – użytkownik zalogowany, posiadający konto, ma dostęp do wszelkich funkcjonalności systemu.
- Moderator – użytkownik tworzony przez administratora, może przeglądać użytkowników i aktywności oraz edytować ich dane, ale nie usuwać permanentnie.
- Administrator – posiada wgląd do logów moderacyjnych, może tworzyć, edytować oraz usuwać konta moderacyjne oraz trwale usuwać użytkowników bądź aktywności.

4.5. Sposób obsługi

Aby założyć konto użytkownik może dołączyć poprzez centralny przycisk „JOIN NOW” (Rysunek 4.5.1) wyświetlany na stronie startowej. Jeżeli użytkownik posiada już konto i chce się zalogować, wystarczy, że wybierze zakładkę „login” (Rysunek 4.5.1) z panelu nawigacyjnego. Zostanie wtedy przekierowany na stronę z logowaniem (Rysunek 4.5.3) bądź rejestracją (Rysunek 4.5.2).



Rysunek 4.5.1 Widok strony głównej



Register

Full Name

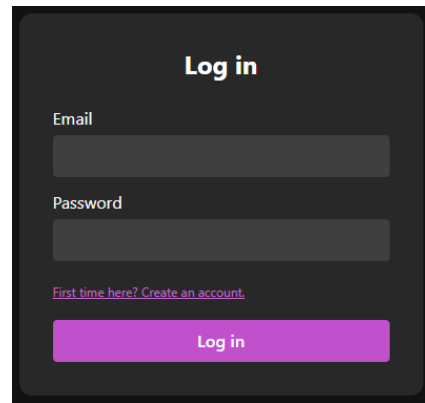
Email

Password

[Already have an account? Log in here.](#)

Register

Rysunek 4.5.2 Widok panelu rejestracyjnego



Log in

Email

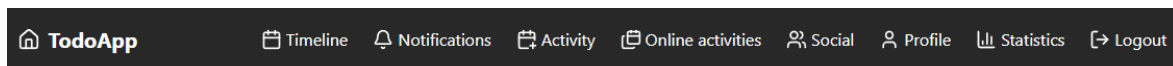
Password

[First time here? Create an account.](#)

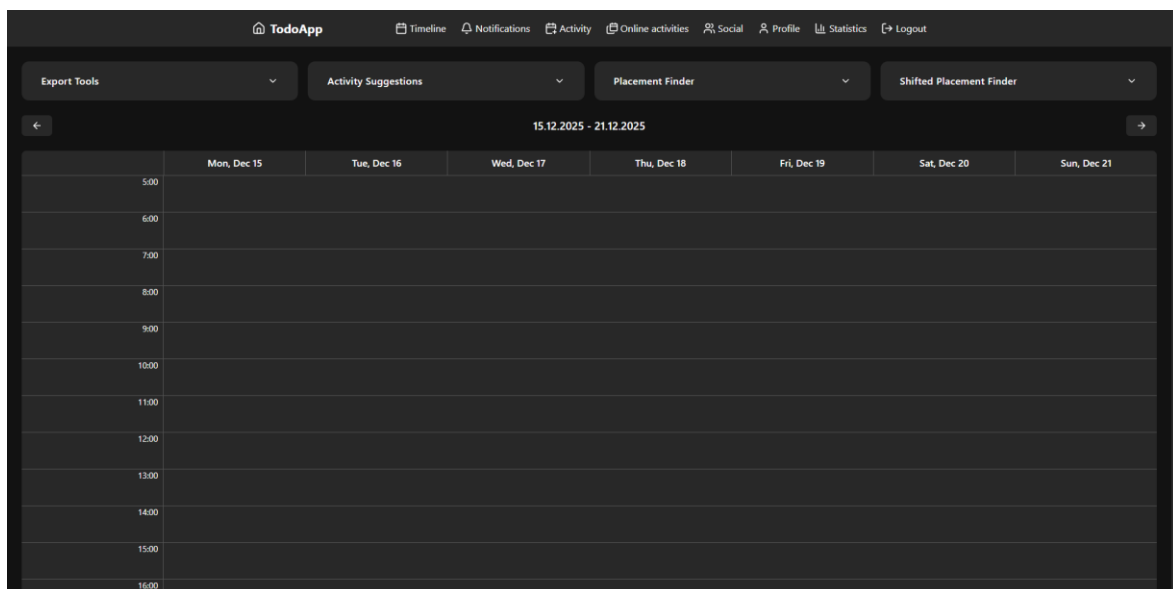
Log in

Rysunek 4.5.3 Widok panelu do zalogowania

Po zalogowaniu użytkownik jest przenoszony do widoku kalendarzowego obecnego tygodnia (Rysunek 4.5.5). Obecnie żadna aktywność nie jest wyświetlana, ponieważ użytkownik żadnej nie stworzył. Aby utworzyć aktywność należy z nagłówka nawigacyjnego (Rysunek 4.5.4) wybrać opcję „Activity”.

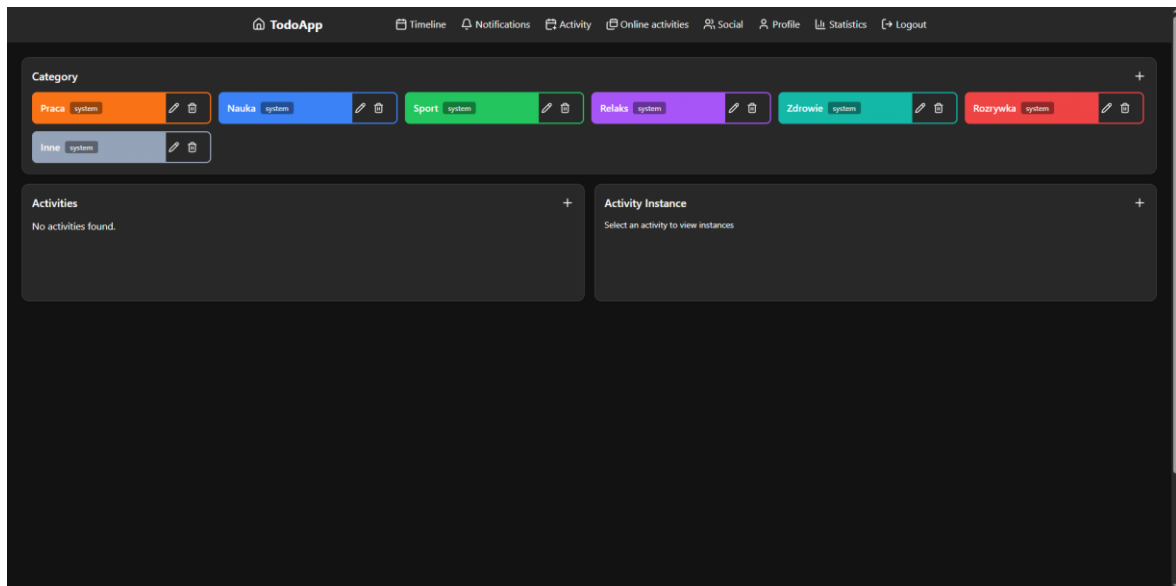


Rysunek 4.5.4 Nagłówek nawigacyjny użytkownika



Rysunek 4.5.5 Widok kalendarzowy

Na panelu Aktywności (Rysunek 4.5.6) użytkownik może przeglądać, tworzyć, edytować oraz usuwać swoje aktywności, kategorie, zasady powtarzalności aktywności oraz pojedyncze instance.

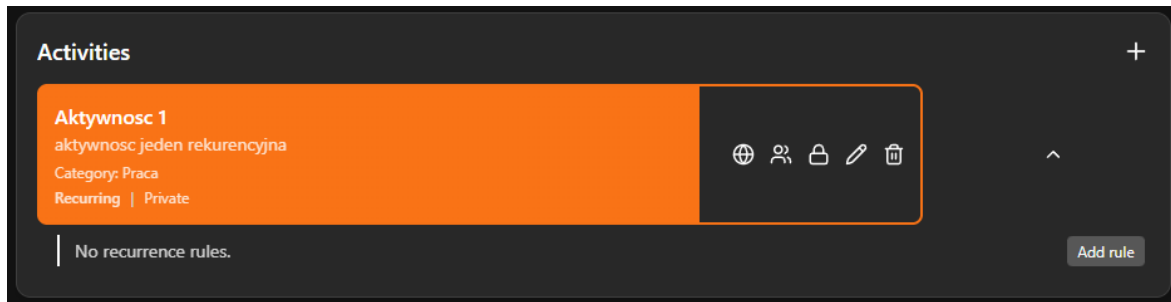


Rysunek 4.5.6 Widok panelu aktywności

Aby utworzyć aktywność należy kliknąć „+” z prawego górnego rogu panelu Activities (Rysunek 4.5.6) wyświetli się wtedy modal (Rysunek 4.5.7) dodawania aktywności.

Rysunek 4.5.7 modal dodawania aktywności

Po dodaniu aktywności, będzie się wyświetlać (Rys 4.5.8) w panelu. Można ją edytować (ikonka ołówka) czy usunąć (ikonka kosza), bądź zmienić tryb Online (globus), Friends-Only (sylwetki ludzi), Offline (kłódka). Dodatkowo do każdej aktywności zdefiniowanej jako powtarzalna podczas tworzenia można dodać zasady powtarzalności.



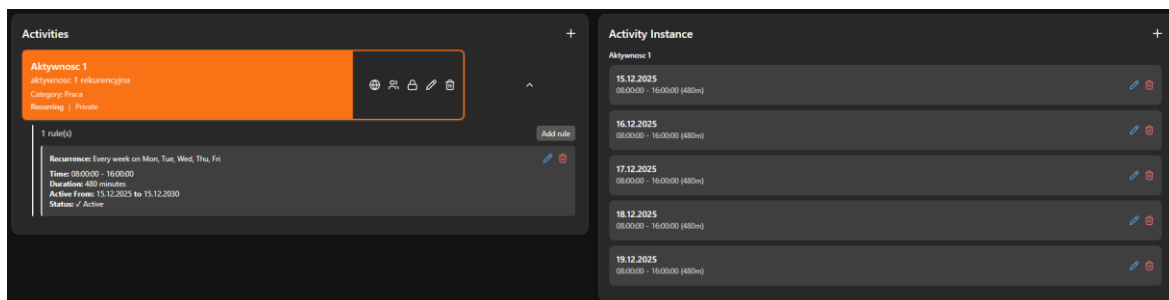
Rysunek 4.5.8 Widok aktywności

Użytkownik chce dodać aktywność „praca” do kalendarza. W tym celu używa trybu powtarzalności „Weekly” zaznacza wybrane dni, oraz uzupełnia resztę danych – czas rozpoczęcia, zakończenia, zakres i interwał tygodni (Rysunek 4.5.9).

The image shows a 'Add Recurrence Rule' modal form. It has a dark background with white text. The form includes several sections: 'Type' with a dropdown menu set to 'Weekly'; 'Interval' with a text input set to '1'; 'Days of Week' with checkboxes for Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday; 'Start Time' with a time picker set to '08:00'; 'End Time' with a time picker set to '16:00'; 'Duration (minutes)' with a text input set to '480'; 'Date Range Start' with a date picker set to '15.12.2025'; 'Date Range End' with a date picker set to '15.12.2030'; and an 'Active' checkbox which is checked. At the bottom, there are two buttons: 'Cancel' and 'Create'.

Rysunek 4.5.9 Widok modala dodawania zasad powtarzalności

Widok po dodaniu zasady, wraz z najbliższymi instancjami aktywności został przedstawiony na rysunku poniżej (Rysunek 4.5.10). Edycji bądź usuwania obydwóch można dokonać poprzez kliknięcie odpowiedniej ikonki przy wybranym rekordzie.

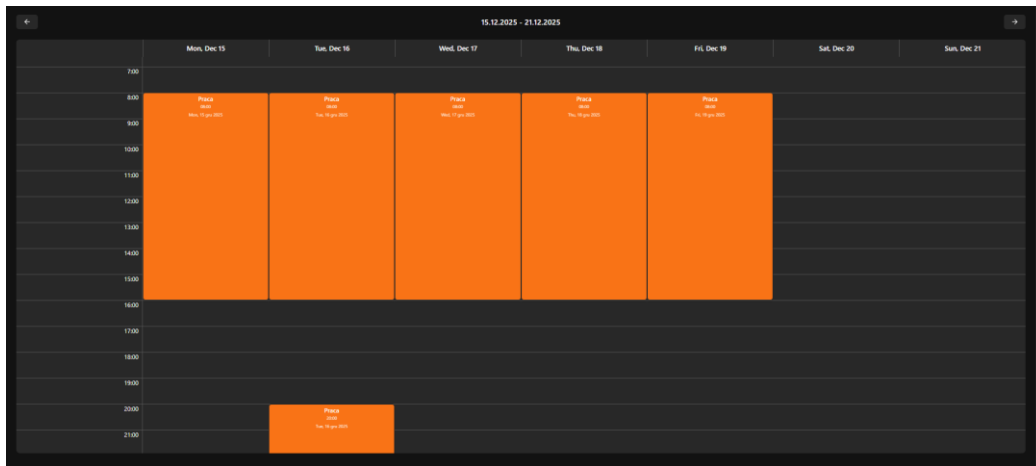


Rysunek 4.5.10 Widok aktywności, wraz z zasadą powtarzalności i instancjami

Użytkownik chce dodać dodatkową instancję pojedynczą aktywności, osobno od utworzonej wcześniej reguły powtarzalności. W tym celu należy wybrać „+” z panelu „Activity Instance” (Rysunek 4.5.10). Wyświetlony wtedy zostanie modal dodawania (Rysunek 4.5.11).

Rysunek 4.5.11 Modal dodawania pojedynczej instancji

Widok kalendarzowy po dodaniu aktywności oraz pojedynczej instancji (Rysunek 4.5.12).



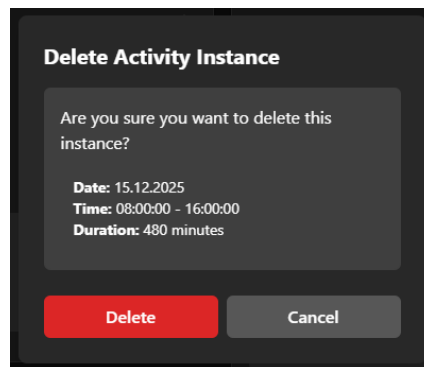
Rysunek 4.5.12 Widok osi czasu po zmianach

Użytkownikowi nie podoba się systemowy kolor kategorii „Praca” więc postanowił stworzyć własną kategorię i zmienić kategorię utworzonej poprzednio aktywności (Rysunek 4.5.8). W tym celu powinien wybrać „+” z panelu „Category” (Rysunek 4.5.6), wyświetlony zostanie modal dodawania kategorii (Rysunek 4.5.13). Aby zaaplikować zmianę kategorii należy wybrać ikonkę ołówka wyświetlaną obok aktywności (Rysunek 4.5.8) – wyświetlony zostanie modal edycyjny (Rysunek 4.5.14)

Rysunek 4.5.13 Modal tworzenia kategorii

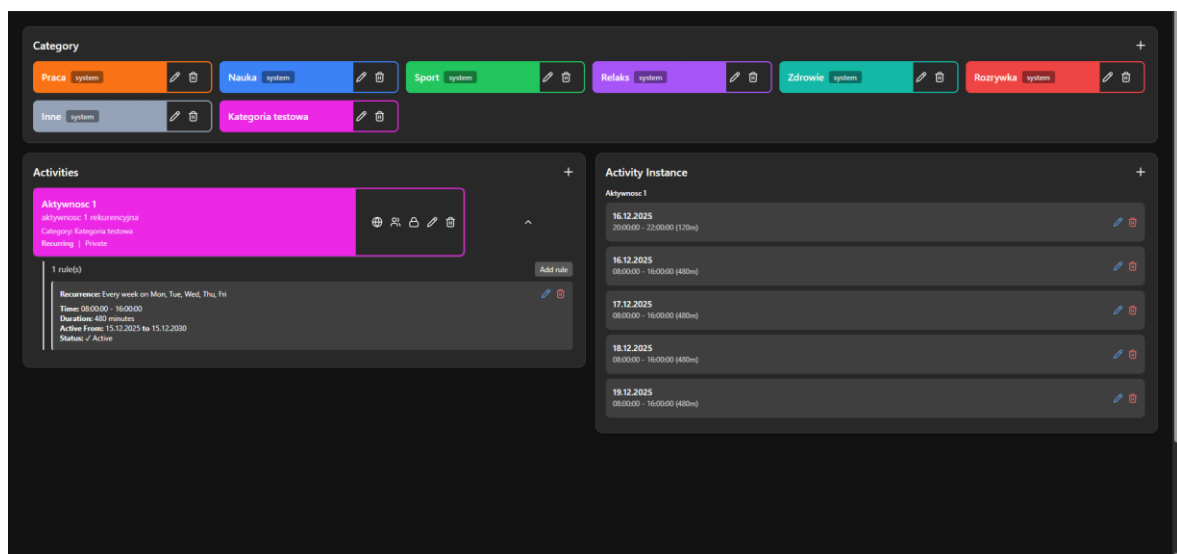
Rysunek 4.5.14 Modal edycji aktywności

Dodatkowo pragnie usunąć wystąpienie aktywności „Praca” w poniedziałek pomimo aktywnej reguły powtarzalności. Aby to zrobić należy kliknąć w ikonkę kosza wyświetlaną obok aktywności (Rysunek 4.5.8), zostanie wtedy wyświetlony modal potwierdzający (Rysunek 4.5.15).

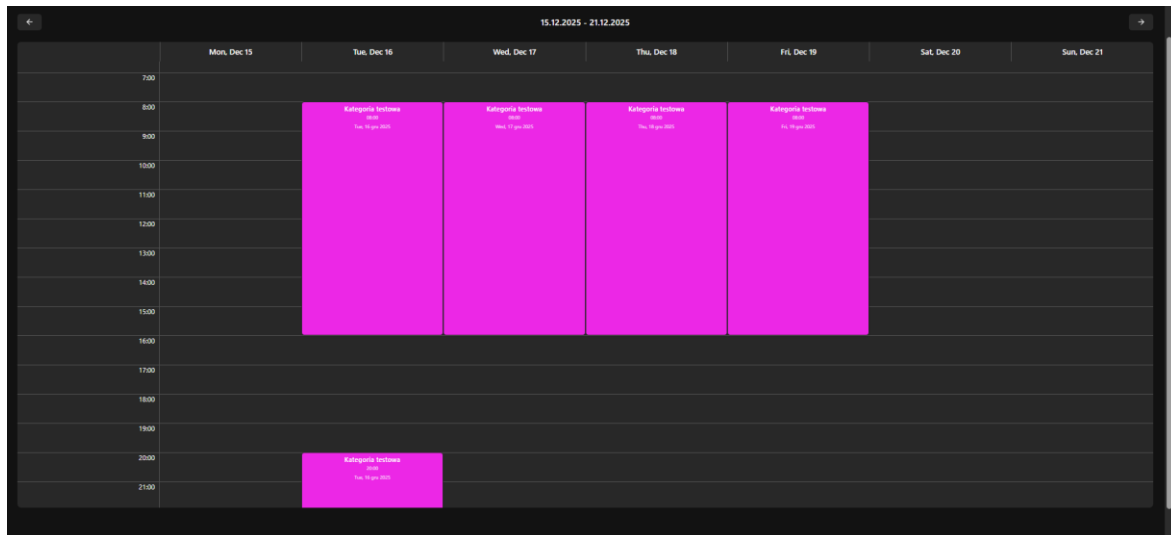


Rysunek 4.5.15 Modal usuwania instancji aktywności

Wyniki zmian można zauważyć na panelu aktywności - nową kategorię, inny kolor aktywności oraz jedną instancję mniej. (Rysunek 4.5.16) Zmiany są również automatycznie odzwierciedlane na widoku kalendarzowym. (Rysunek 4.5.17)

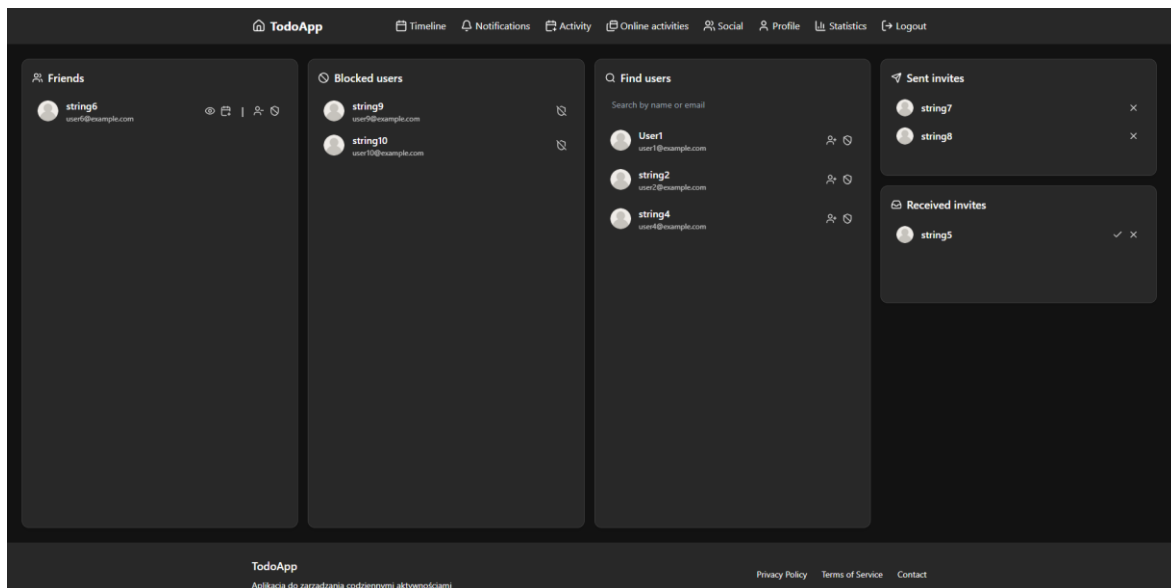


Rysunek 4.5.16 Widok panelu aktywności po zmianach



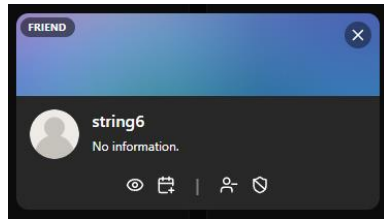
Rysunek 4.5.17 Widok kalendarzowy po zmianach

Użytkownik chciałby dodać znajomych oraz zablokować niektórych użytkowników. Takie funkcjonalności oferuje mu zakładka „Social”. Poniższy rysunek (Rysunek 4.5.18) przedstawia wygląd panelu po wykonaniu akcji dodawania i blokowania, otrzymaniu zaproszenia oraz posiadaniu już znajomego. Każdą z tych akcji wykonuje się poprzez kliknięcie w odpowiednią ikonkę, obok osoby której wobec której chcemy wykonać akcję.

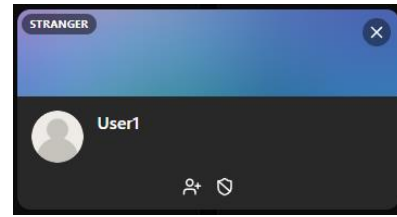


Rysunek 4.5.18 Widok panelu znajomych

Dodatkowo po kliknięciu w osobę wyświetla się nam jej tło miniprofilu (Rysunek 4.5.19, 4.5.20) które zawiera dodatkowo zdjęcie w tle, opis i status – czy jest się znajomym z danym użytkownikiem.

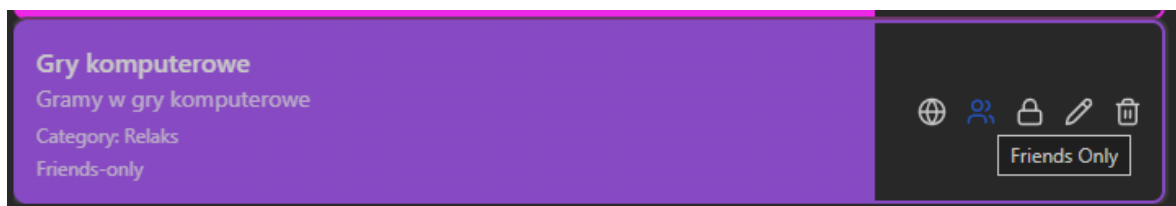


Rysunek 4.5.19 miniprofil znajomego



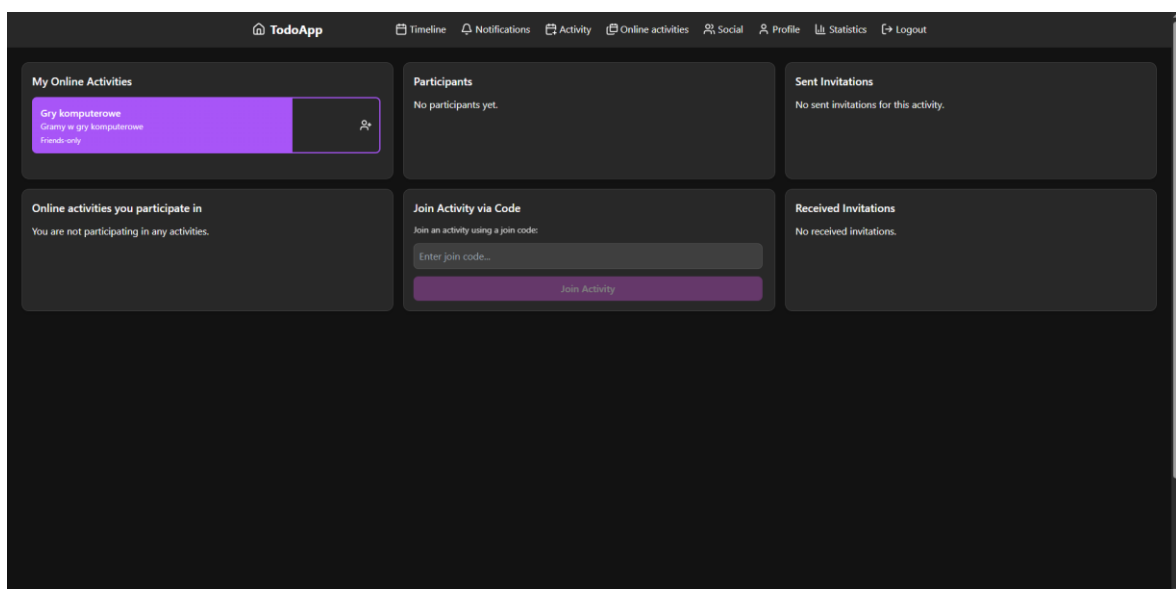
Rysunek 4.5.20 miniprofil nieznanego

Po dodaniu znajomych użytkownik chciałby stworzyć aktywność, którą mógłby dzielić ze znajomymi, analogicznie jak poprzednio (Rysunek 4.5.6/7, Rysunek 4.5.11) tworzymy aktywność oraz instancję, następnie w aktywności zmieniamy widoczność na „Friends-Only” poprzez kliknięcie odpowiedniej ikonki (Rysunek 4.5.21).



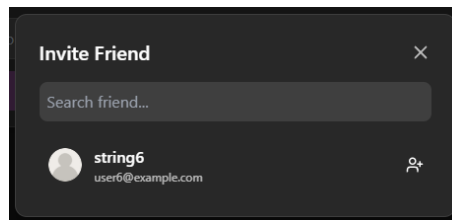
Rysunek 4.5.21 Widok aktywności z opcją „Friends-Only”

Po utworzeniu takiej aktywności możemy przejść do widoku aktywności online (Rysunek 4.5.22) – „Online activities” na pasku nawigacyjnym. Strona ta zawiera takie informacje jak wysłane, otrzymane zaproszenia, uczestnictwo w aktywnościach online, listy uczestników oraz możliwość dołączenia do aktywności poprzez kod dostępu.



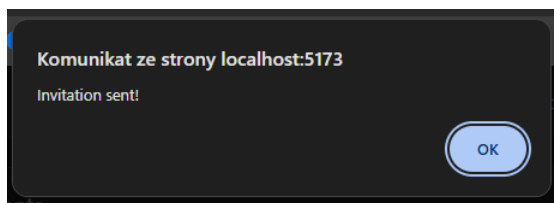
Rysunek 4.5.22 Widok strony aktywności online

Aby zaprosić znajomego do aktywności online której jesteśmy właścicielem należy kliknąć ikonkę dodawania, widoczną obok wybranej aktywności. Wyświetlony wtedy zostanie modal zapraszania znajomych (Rysunek 4.5.23).

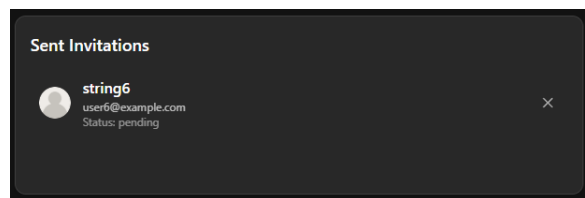


Rysunek 4.5.23 Modal zapraszania znajomych

Po wybraniu, którego znajomego chcemy zaprosić klikamy ikonkę obok znajomego i system wyświetli nam alert o sukcesie (Rysunek 4.5.24), jak i na panelu aktywności online w sekcji wysłanych zaproszeń będzie widoczne wysłane zaproszenie (Rysunek 4.5.25).

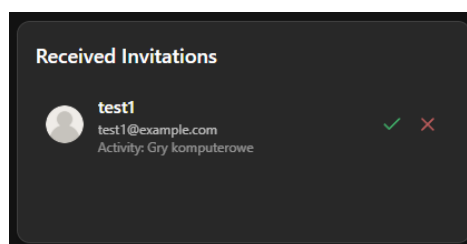


Rysunek 4.5.24 Alert sukcesu



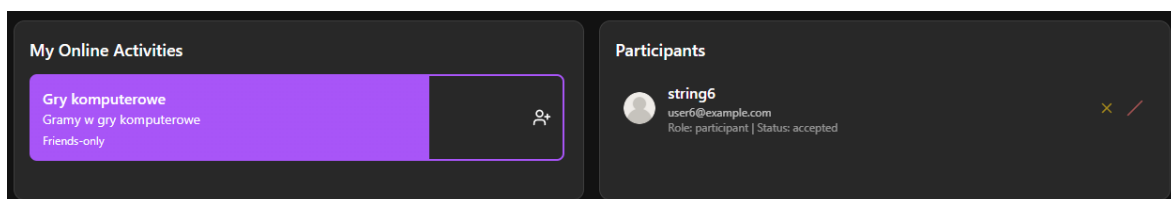
Rysunek 4.5.25 Panel wysłanych zaproszeń

Analogicznie w panelu z otrzymanymi zaproszeniami (Rysunek 4.5.26) są wyświetlane otrzymane zaproszenia z opcją akceptacji bądź odrzucenia reprezentowane poprzez ikony.



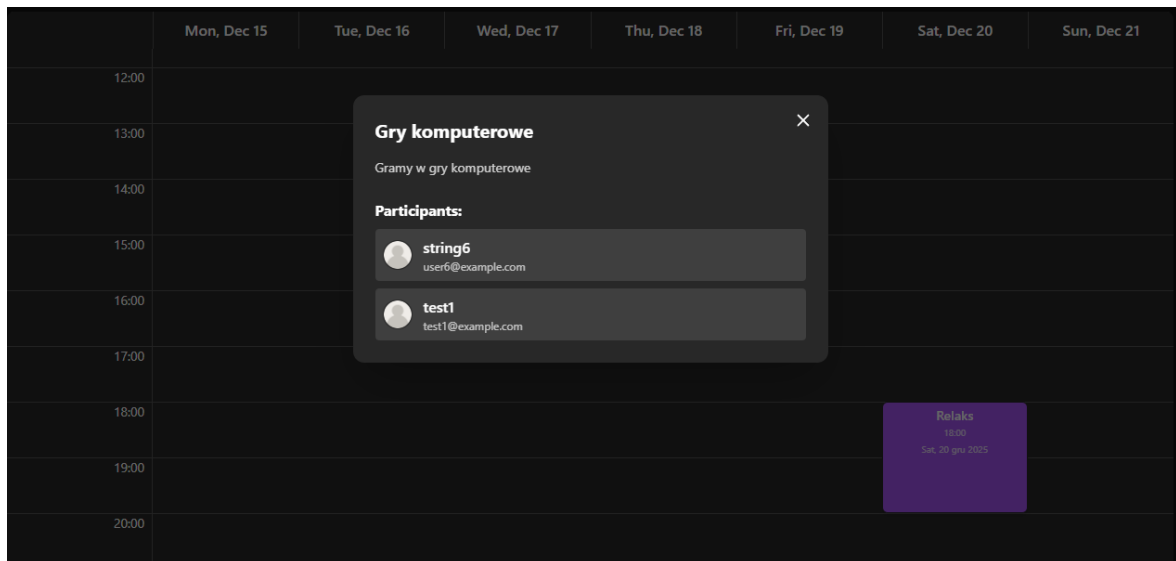
Rysunek 4.5.26 Panel otrzymanych zaproszeń

Gdy osoba zaproszona zaakceptuje zaproszenie to po kliknięciu na wybraną aktywność możemy ją zobaczyć jako uczestnika – wraz z możliwością wyrzucenia i zablokowania ponownego dołączenia (Rysunek 4.5.27).



Rysunek 4.5.27 Widok na uczestników aktywności online

Każdy uczestnik aktywności może zobaczyć jej instancję na swojej osi czasu, a po kliknięciu w daną instancję wyświetla się modal ze szczegółami oraz listą uczestników (Rysunek 4.5.28).



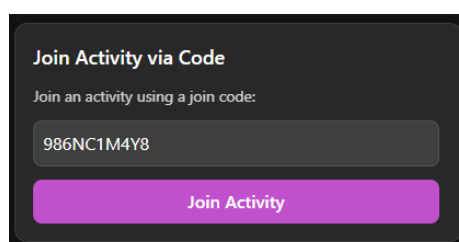
Rysunek 4.5.28 Widok osi czasu z aktywnością online

Użytkownik chciałby zorganizować ogólnodostępną aktywność, dla wszystkich osób posiadających kod dostępu. Najpierw analogicznie jak w dwóch poprzednich przypadkach tworzymy aktywność i instancję, następnie zmieniamy widoczność aktywności na publiczną (Rysunek 4.5.29).



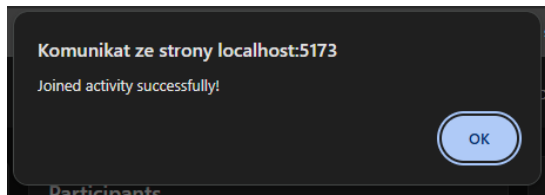
Rysunek 4.5.29 Widok aktywności publicznej

Dołączyć do takiej aktywności można z poziomu panelu aktywności online po wpisaniu kodu dostępu w odpowiednim panelu (Rysunek 4.5.30) strony aktywności online.

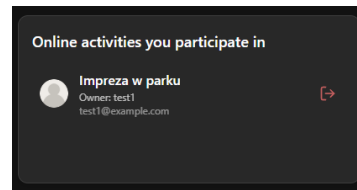


Rysunek 4.5.30 Widok kodu dostępu

O sukcesie dołączenia poinformuje nas system poprzez alert (Rysunek 4.5.31) oraz w widoku uczestnictwa pojawi się nowa aktywność (Rysunek 4.5.32).

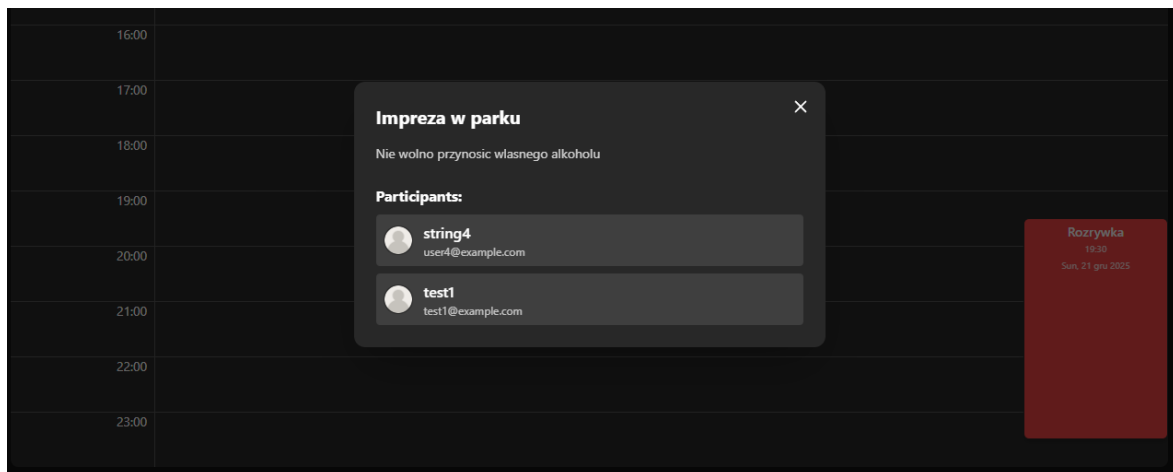


Rysunek 4.5.31 Alert sukcesu



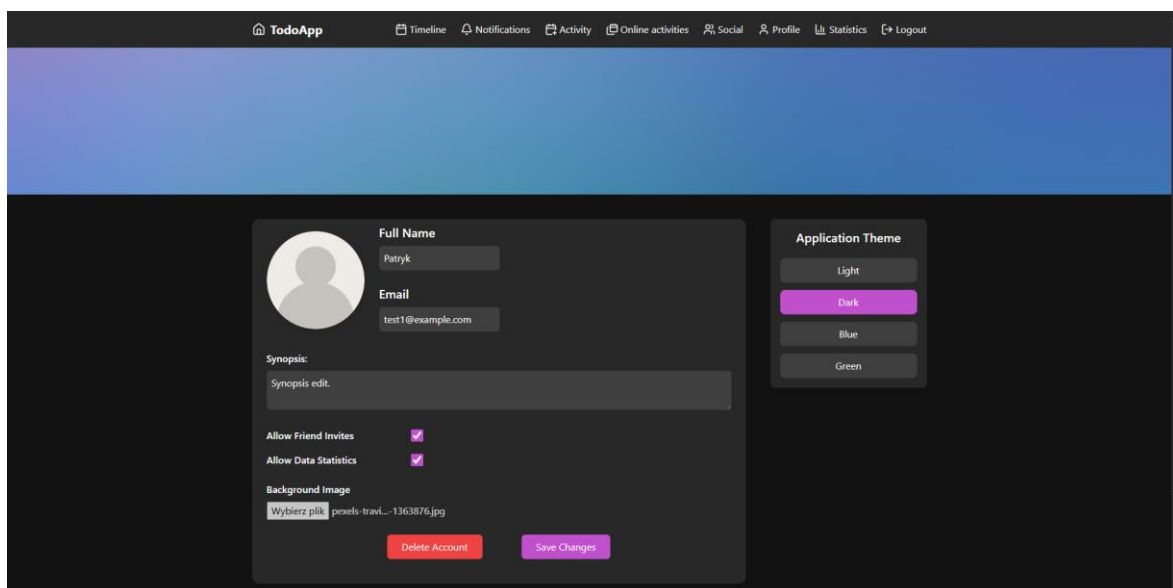
Rysunek 4.5.32 Widok uczestnictwa

Osoba dołączająca do takiej aktywności może zobaczyć jej instancję na swojej osi czasu, po kliknięciu wyświetla się lista uczestników oraz szczegóły (Rysunek 4.5.33).



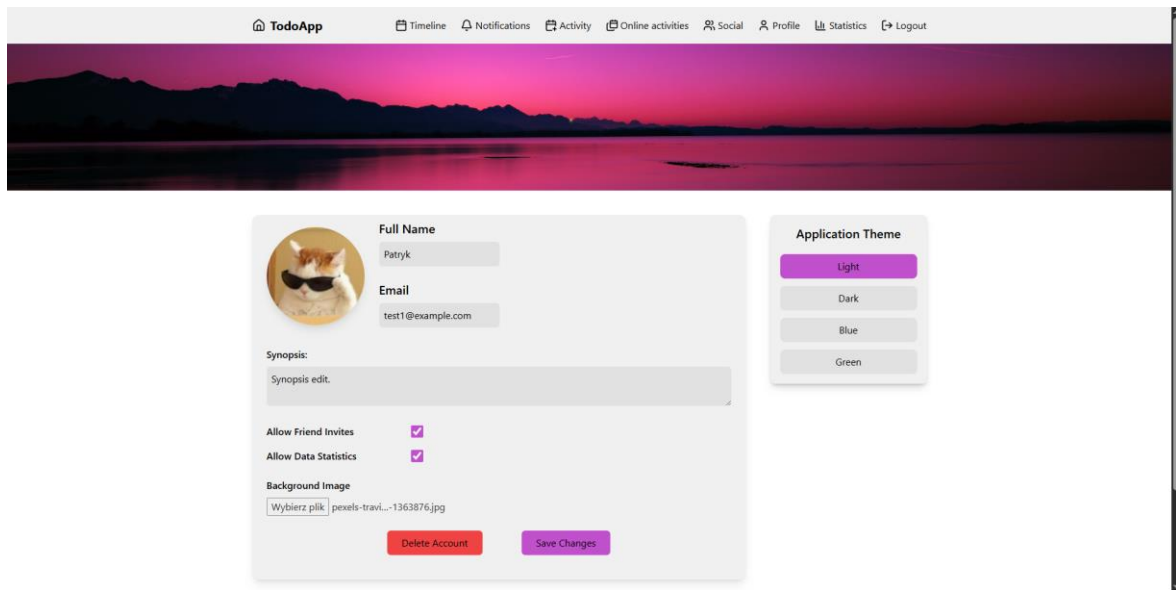
Rysunek 4.5.33 Widok osi czasu z aktywnością publiczną

Użytkownikowi pragnie dokonać zmian swojego profilu - zdjęcia, nazwy, opisu i motywu systemu. Aby tego dokonać należy wybrać opcję „Profile” z menu nawigacyjnego (Rysunek 4.5.4) i edytować widoczne na profilu wartości (Rysunek 4.5.34).



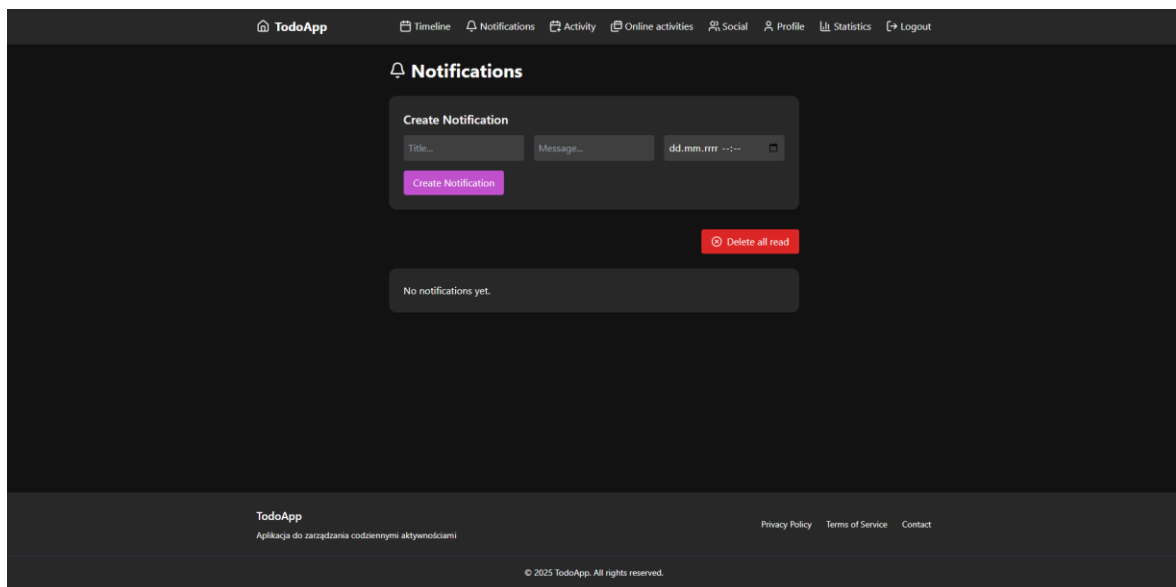
Rysunek 4.5.34 Profil użytkownika z niezatwierdzonymi zmianami

Po zapisaniu zmian profil użytkownika wygląda następująco (Rysunek 4.5.35):

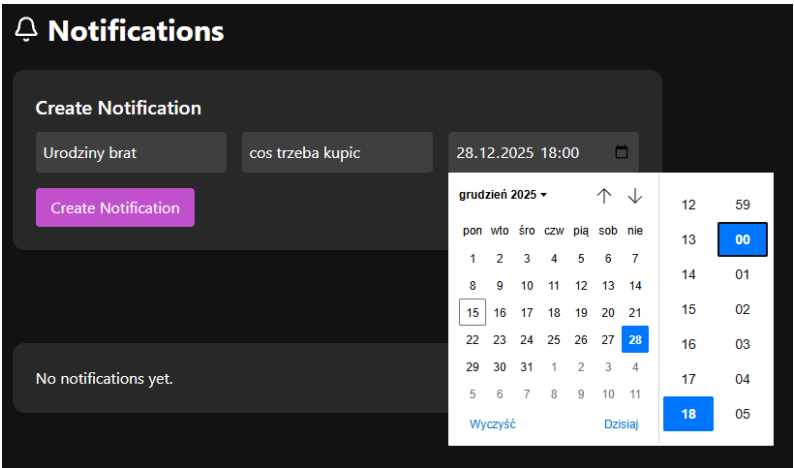


Rysunek 4.5.35 Profil użytkownika po zmianach

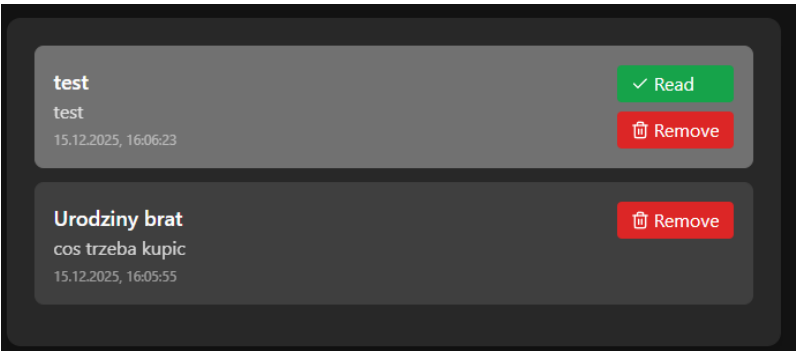
Użytkownik chce utworzyć przypomnienie o urodzinach swojego brata. Może tego dokonać za pomocą panelu „Notifications” (Rysunek 4.5.36). Należy wpisać wybrane wartości w polach panelu „Create Notification” (Rysunek 4.5.37), efekt będzie widoczny poniżej zaraz po utworzeniu (Rysunek 4.5.38).



Rysunek 4.5.36 Widok panelu powiadomień

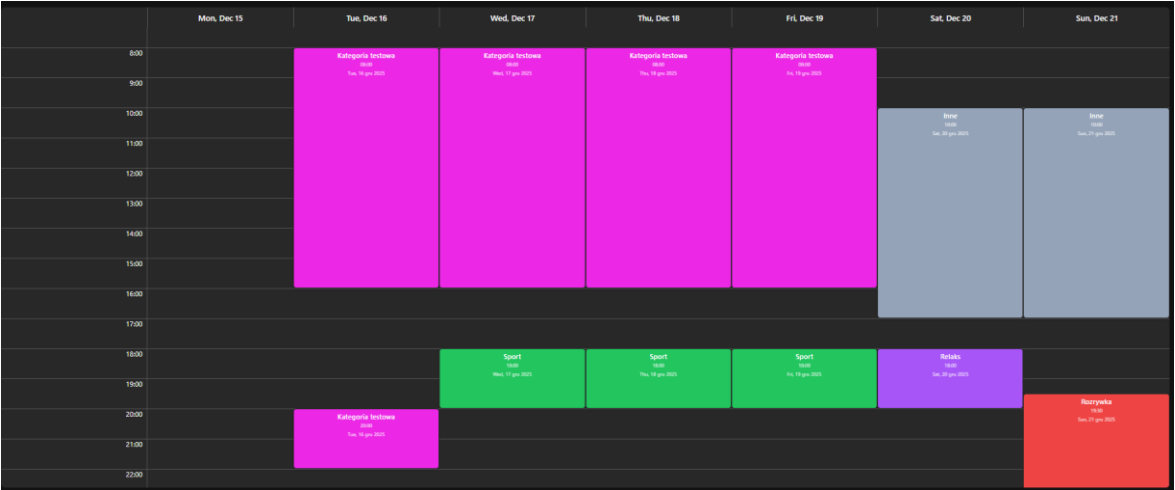


Rysunek 4.5.37 Dodawanie nowego powiadomienia

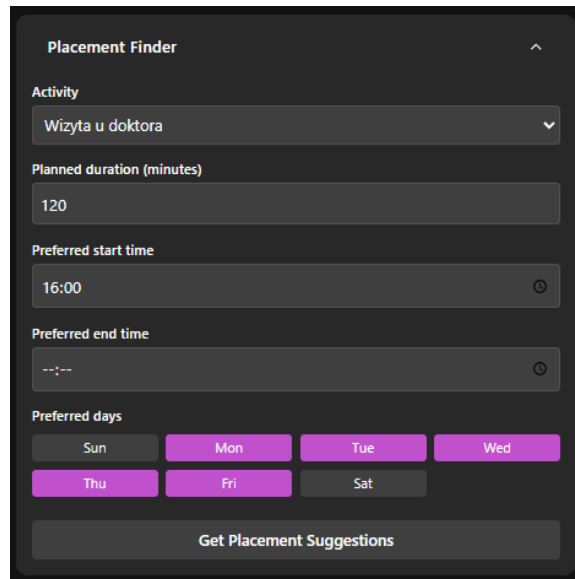


Rysunek 4.5.38 Widok istniejących powiadomień

Użytkownik musi udać się do lekarza, więc zamierza skorzystać z algorytmu umiejscowienia aktywności, aby dopasować wizytę do swojego planu tygodnia. Wybiera opcję „Placement Finder” na widoku kalendarzowym (Rysunek 4.5.5) wypełnia wyświetlony formularz (Rysunek 4.5.40) preferowanymi danymi oraz zatwierdza odpowiadający wynik (Rysunek 4.5.41). Oś czasu użytkownika przed tymi zmianami prezentuje się następująco (Rysunek 4.5.39):

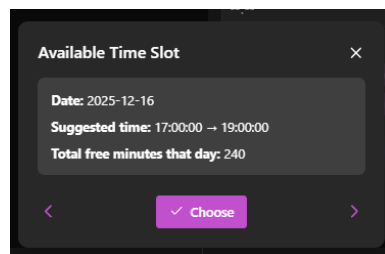


Rysunek 4.5.39 Oś tygodnia użytkownika



The 'Placement Finder' modal is a dark-themed form with a title bar at the top right containing an upward arrow. It contains several input fields: a dropdown menu for 'Activity' with 'Wizyta u doktora' selected, a text input for 'Planned duration (minutes)' with '120', and two time pickers for 'Preferred start time' (set to '16:00') and 'Preferred end time' (set to '--:--'). Below these is a section for 'Preferred days' with seven buttons: 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', and 'Sat'. The 'Mon', 'Tue', 'Wed', 'Thu', and 'Fri' buttons are highlighted in purple. At the bottom is a wide button labeled 'Get Placement Suggestions'.

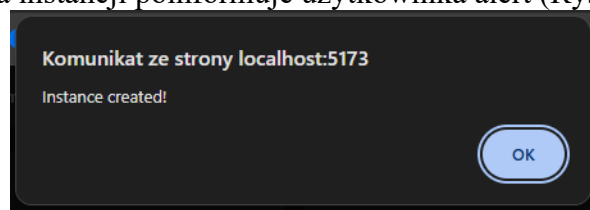
Rysunek 4.5.40 Modal algorytmu sugerującego



The 'Available Time Slot' modal is a dark-themed window with a close button (X) in the top right. It displays information about a specific time slot: 'Date: 2025-12-16', 'Suggested time: 17:00:00 → 19:00:00', and 'Total free minutes that day: 240'. At the bottom, there are left and right navigation arrows and a purple button with a checkmark and the text 'Choose'.

Rysunek 4.5.41 Jeden z wyników algorytmu sugerującego

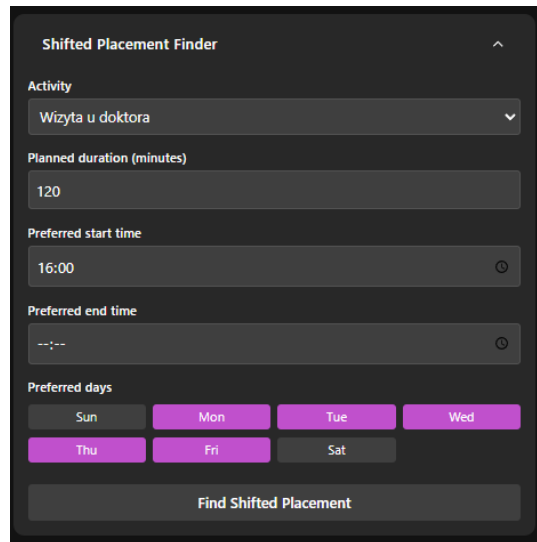
O sukcesie utworzenia instancji poinformuje użytkownika alert (Rysunek 4.5.42).



The alert dialog is a dark-themed box with a title 'Komunikat ze strony localhost:5173'. The message inside reads 'Instance created!'. In the bottom right corner, there is a blue button with the text 'OK'.

Rysunek 4.5.42 Alert sukcesu

Użytkownik chce wstawić kolejną wizytę u lekarza do planu tygodnia, ale tym razem preferowałby bardziej wydajne dopasowanie, w tym celu może skorzystać z algorytmu „Shifted Placement Finder” który proponuje ingerencję w istniejące instancje aktywności. W tym celu wybiera opcję „Placement shifted” na widoku kalendarzowym (Rysunek 4.5.5), wypełnia wyświetlany formularz (Rysunek 4.5.43). Następnie wybiera odpowiadający termin i uzupełnia dodatkowe informacje o przesunięciu aktywności (Rysunek 4.5.44).

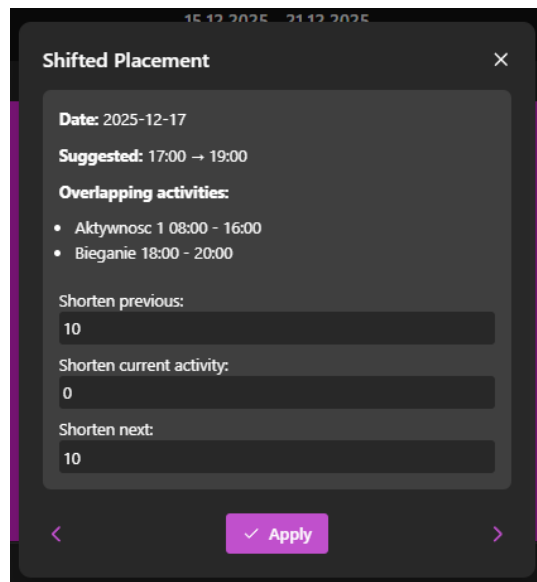


The 'Shifted Placement Finder' modal contains the following fields and controls:

- Activity:** A dropdown menu with 'Wizyta u doktora' selected.
- Planned duration (minutes):** A text input field containing '120'.
- Preferred start time:** A time picker showing '16:00'.
- Preferred end time:** A time picker showing '--:--'.
- Preferred days:** A grid of day buttons: Sun, Mon, Tue, Wed, Thu, Fri, Sat. 'Mon', 'Tue', 'Wed', 'Thu', and 'Fri' are highlighted in blue.
- Find Shifted Placement:** A large button at the bottom.

Rysunek 4.5.43 Modal algorytmu ingerującego

Użytkownik zdecydował się zakończyć poprzednią aktywność wcześniej i przesunąć rozpoczęcie następnej.



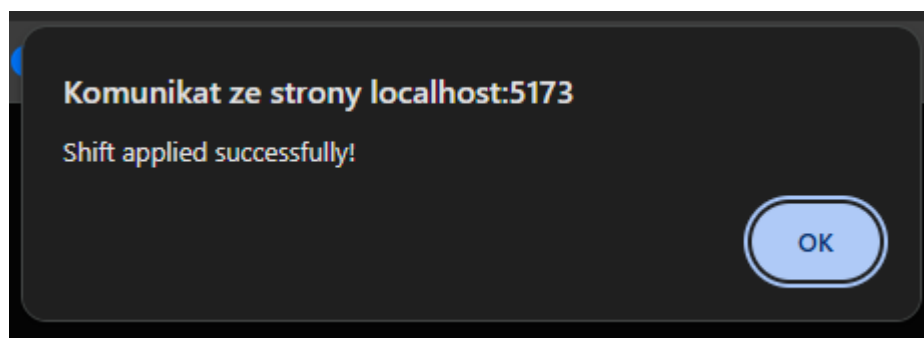
The 'Shifted Placement' modal displays the following information:

- Date:** 2025-12-17
- Suggested:** 17:00 → 19:00
- Overlapping activities:**
 - Aktywnosc 1 08:00 - 16:00
 - Bieganie 18:00 - 20:00
- Shorten previous:** 10
- Shorten current activity:** 0
- Shorten next:** 10

At the bottom, there is a blue 'Apply' button with a checkmark icon, flanked by left and right navigation arrows.

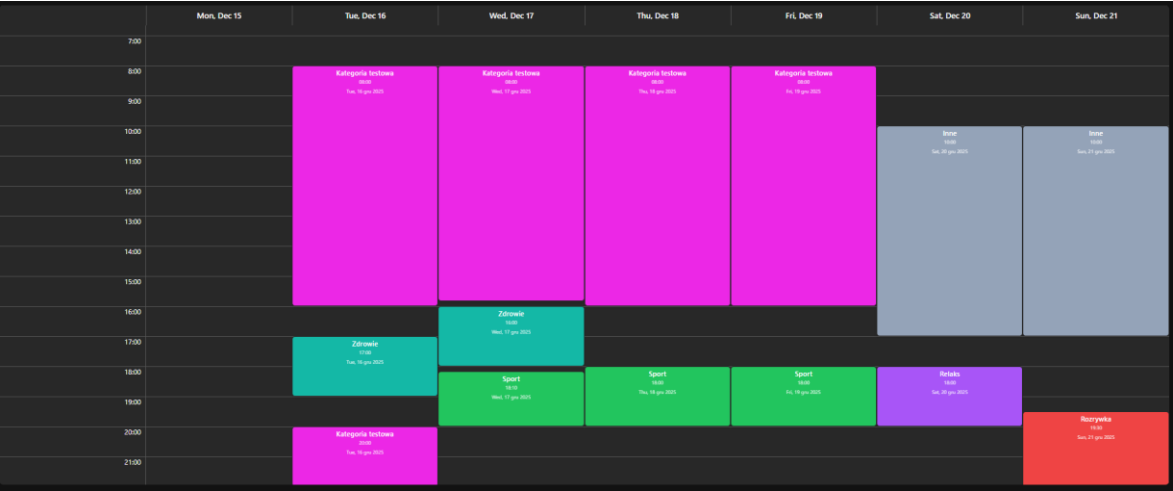
Rysunek 4.5.44 Jeden z wyników algorytmu ingerującego

O sukcesie informuje użytkownika alert systemowy (Rysunek 4.5.45).



Rysunek 4.5.45 Alert sukcesu

Wykonane zmiany użytkownik może zauważyć na swoim widoku kalendarzowym (Rysunek 4.5.46). Analogicznie do wybranego wyniku pierwszej wizyty (Rysunek 4.5.41) została automatycznie utworzona instancja w wybranym terminie. W przypadku drugiej wizyty algorytm automatycznie zmodyfikował istniejące aktywności zgodnie z podanymi przez użytkownika danymi i wybranym terminem (Rysunek 4.5.44).

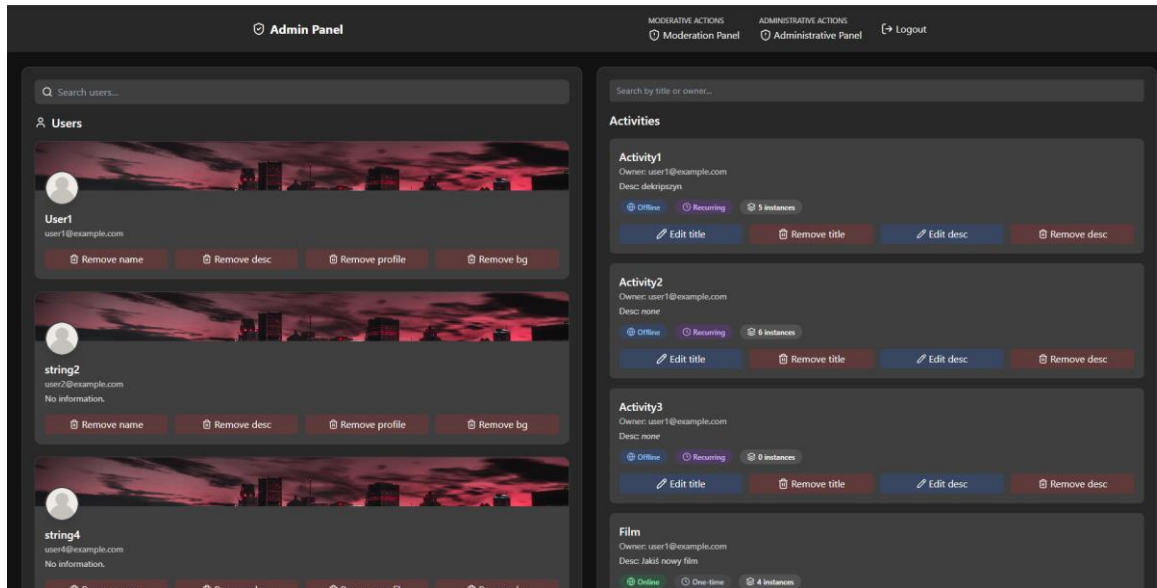


Rysunek 4.5.46 Widok osi czasu po zmianach

4.6. Administracja systemu

Niniejszy podrozdział skupia się na zobrazowaniu funkcjonalności kont administracyjnych, przedstawiony jest wygląd interfejsu wraz z opisem funkcjonalności oraz przykładem akcji moderacyjnych.

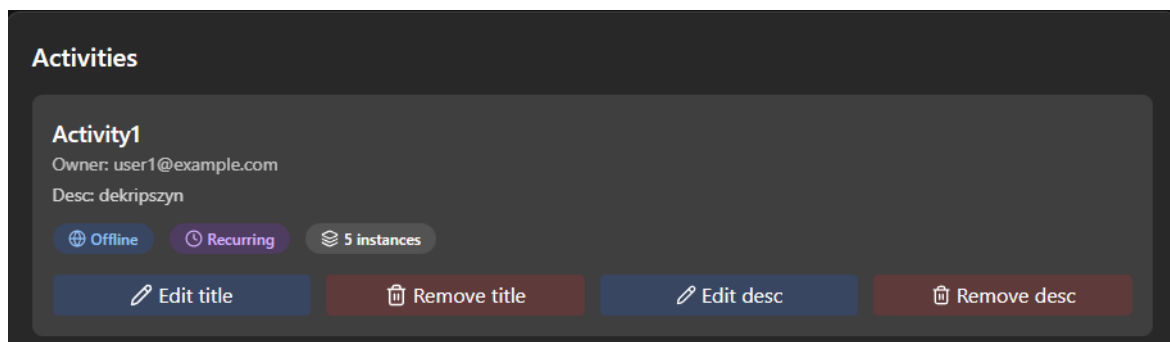
4.6.1 Panel Moderacyjny



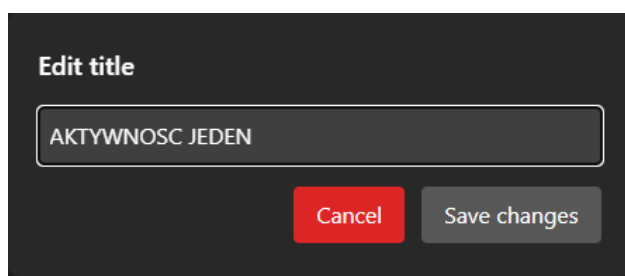
Rysunek 4.6.1 Wygląd interfejsu panelu moderacyjnego

Lewy panel strony „Users” (Rysunek 4.6.1) odpowiada za wyświetlanie kont użytkowników wraz z możliwością wyszukania, oraz oferuje akcje moderacyjne – zresetowanie nazwy, opisu, zdjęcia profilowego, zdjęcia w tle. Prawa strona „Activities” odpowiada za wyświetlanie aktywności wraz z możliwością wyszukiwania utworzonych przez użytkowników wraz z oferowaniem takich funkcjonalności jak edycja, reset tytułu i opisu.

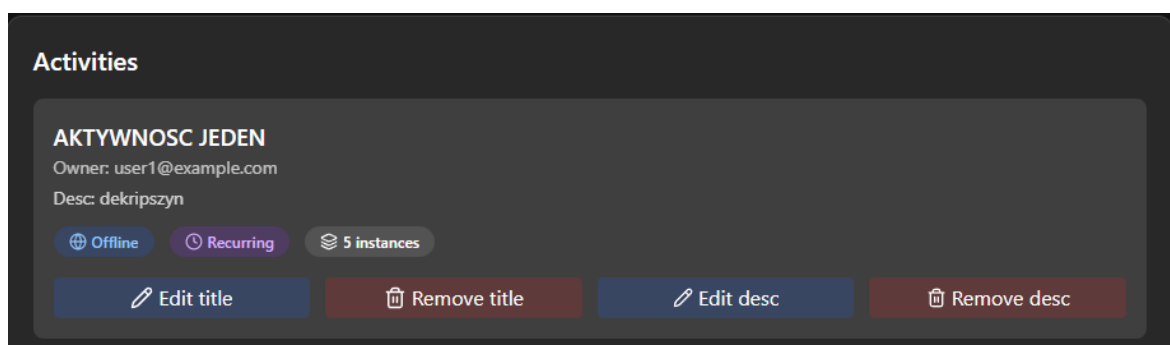
Przykładowa akcja moderacyjna obejmująca edycję tytułu aktywności została zobrazowana na rysunkach poniżej. Celem edycji danej aktywności wystarczy wybrać odpowiednią opcję z wyświetlanych przy każdej z nich (Rysunek 4.6.2). Po wybraniu opcji „Edit title” zostanie wyświetlony modal edycyjny (Rysunek 4.6.3). Po wpisaniu nowej wartości i zatwierdzeniu aktywność zostanie edytowana (Rysunek 4.6.4).



Rysunek 4.6.2 Aktywność przed edycją



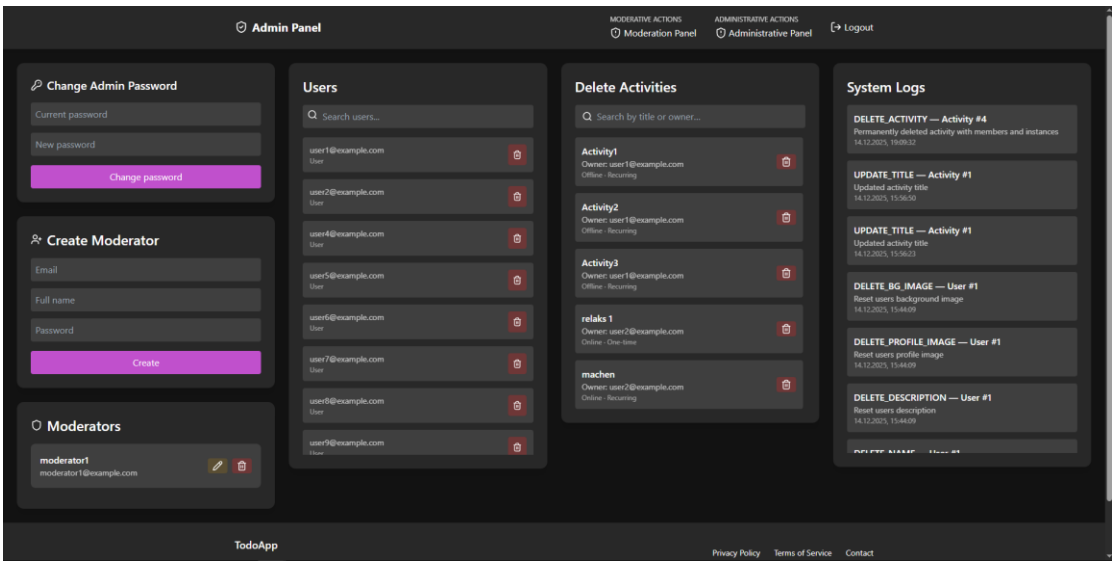
Rysunek 4.6.3 Modal edycji aktywności



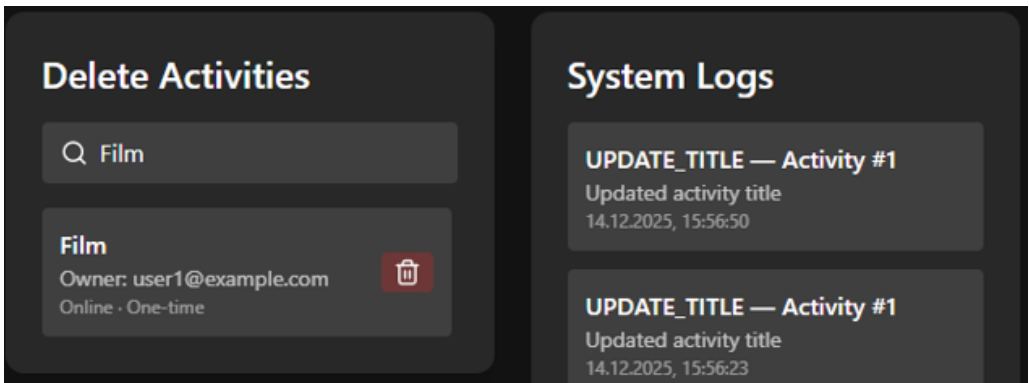
Rysunek 4.6.4 Widok aktywności po modyfikacji

4.6.2. Panel Administracyjny

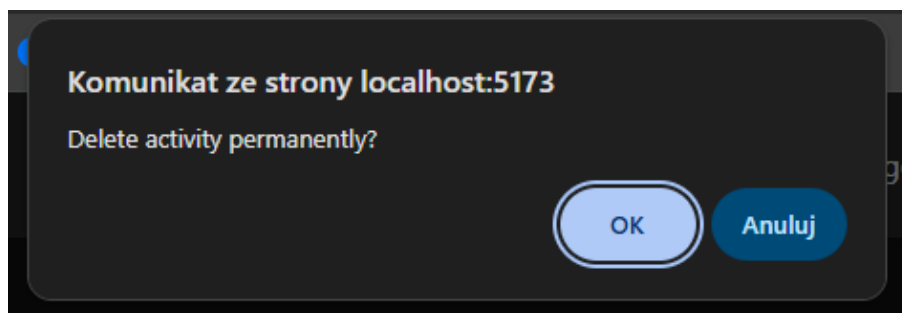
Panel administracyjny (Rysunek 4.6.5) implementuje takie funkcjonalności jak zmianę własnego hasła, tworzenie, usuwanie i edycję kont moderacyjnych. Trwałe usuwanie użytkowników i aktywności oraz przeglądanie logów systemowych. Przykładowa funkcjonalność usunięcia aktywności została przedstawiona za pomocą rysunków poniżej. Najpierw, z panelu „Delete Activities” wyszukujemy aktywność do usunięcia (Rysunek 4.6.6), następnie wybieramy ikonkę kosza, po potwierdzeniu (Rysunek 4.6.7) aktywność zostanie permanentnie usunięta i zostanie dodany nowy log systemowy (Rysunek 4.6.8).



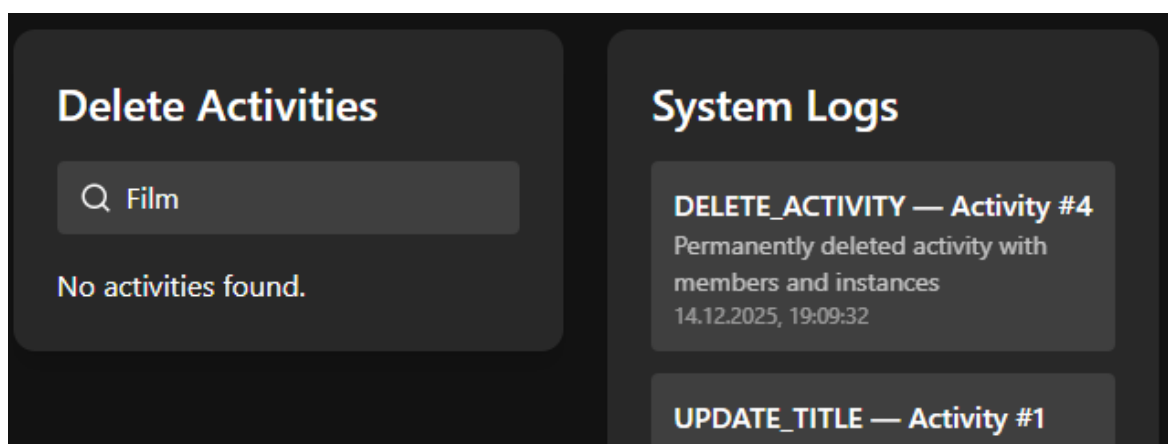
Rysunek 4.6.5 Widok panelu administracyjnego



Rysunek 4.6.6 Wyszukanie aktywności do usunięcia



Rysunek 4.6.7 Komunikat potwierdzający



Rysunek 4.6.8 Widok po usunięciu aktywności

Rozdział 5

Specyfikacja wewnętrzna

W niniejszym rozdziale przedstawiona jest idea aplikacji wraz z opisem architektury systemowej. Dodatkowo rozdział zawiera opis wykorzystanej w systemie bazy danych wraz z technicznym przedstawieniem i opisem zaimplementowanych w programie algorytmów.

5.1. Przedstawienie idei

Ideą systemu jest stworzenie intuicyjnego, prostego w obsłudze rozwiązania przeglądarkowego wspomagającego użytkownika w organizacji codziennych aktywności. System dostosowuje się do indywidualnych nawyków użytkownika, kluczowym założeniem projektowym jest interaktywna wizualizacja osi czasu oraz analiza wcześniejszych działań użytkownika w celu proponowania sugestii dotyczących kolejnych aktywności. Kolejnym równie ważnym aspektem jest możliwość integracji z innymi użytkownikami poprzez aktywności publiczne oraz współdzielone jak i możliwość odcięcia się od niepożądanych osób poprzez blokowanie. System skierowany jest do osób prywatnych, które szukają prostych w obsłudze, darmowych rozwiązań, które oferują wyżej wymienione kluczowe funkcjonalności.

5.2. Architektura systemu

Niniejszy podrozdział zawiera opis architektury systemu z podziałem na część backendową oraz frontendową.

Strona frontendowa

Odpowiedzialna za wyświetlanie danych w jasnej i czytelnej formie oraz interakcję z użytkownikiem. Interfejs użytkownika został zaprojektowany i zaimplementowany w pełni responsywnie, co umożliwia korzystanie z aplikacji na różnych urządzeniach i rozdzielczościach wyświetlaczy. Komunikacja z warstwą backendową odbywa się poprzez REST API, z wykorzystaniem żądań HTTP i danych w formacie JSON. Dodatkowo frontend realizuje również:

- walidację uprawnień użytkowników poprzez tokeny JWT
- walidację wprowadzanych przez użytkowników danych
- dynamiczne aktualizowanie widoków – strony zbudowane z komponentów

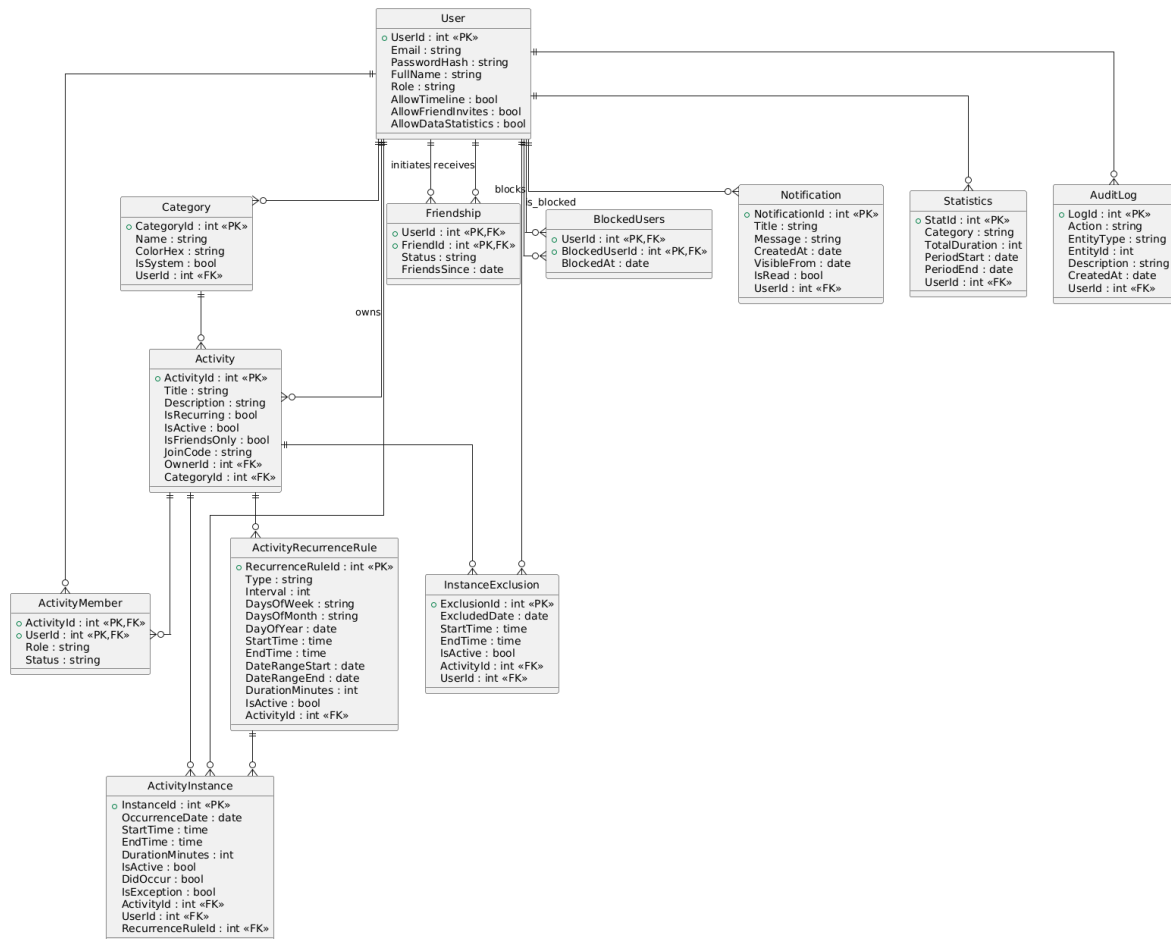
Strona backendowa

Strona backendowa systemu odpowiada za logikę biznesową, przetwarzanie danych otrzymanych z frontendu, kontrolę dostępu do zasobów i operacje na bazie danych. Backend udostępnia zestaw endpointów REST API, które następnie wykorzystuje frontend do komunikacji i realizacji wszelakich zapytań. Architektura systemu została oparta na warstwowym podziale odpowiedzialności, co pozwala na zachowanie czytelniejszej struktury aplikacji oraz łatwiejsze dalsze rozwijanie. Zastosowany podział obejmuje następujące warstwy:

- Kontroler – odpowiada za obsługę żądań http, podstawową walidację danych wejściowych oraz przekazywanie żądań do odpowiednich serwisów oraz wyświetlanie informacji zwrotnych otrzymanych od serwisów.
- Serwis – zawiera właściwą logikę biznesową, odpowiada za przetwarzanie danych oraz koordynację operacji na encjach i komunikację z bazą danych.
- DTO (Data Transfer Object) – służy do bezpiecznego przekazywania danych pomiędzy warstwami, aby nie operować bezpośrednio na encji. Daje możliwość wydzielenia wybranych pól do wyświetlania, ograniczając ekspozycję wewnętrznej struktury encji.
- Encja – reprezentacja tabeli w bazie danych, odpowiada bezpośrednio strukturze tabel i relacjom między nimi.

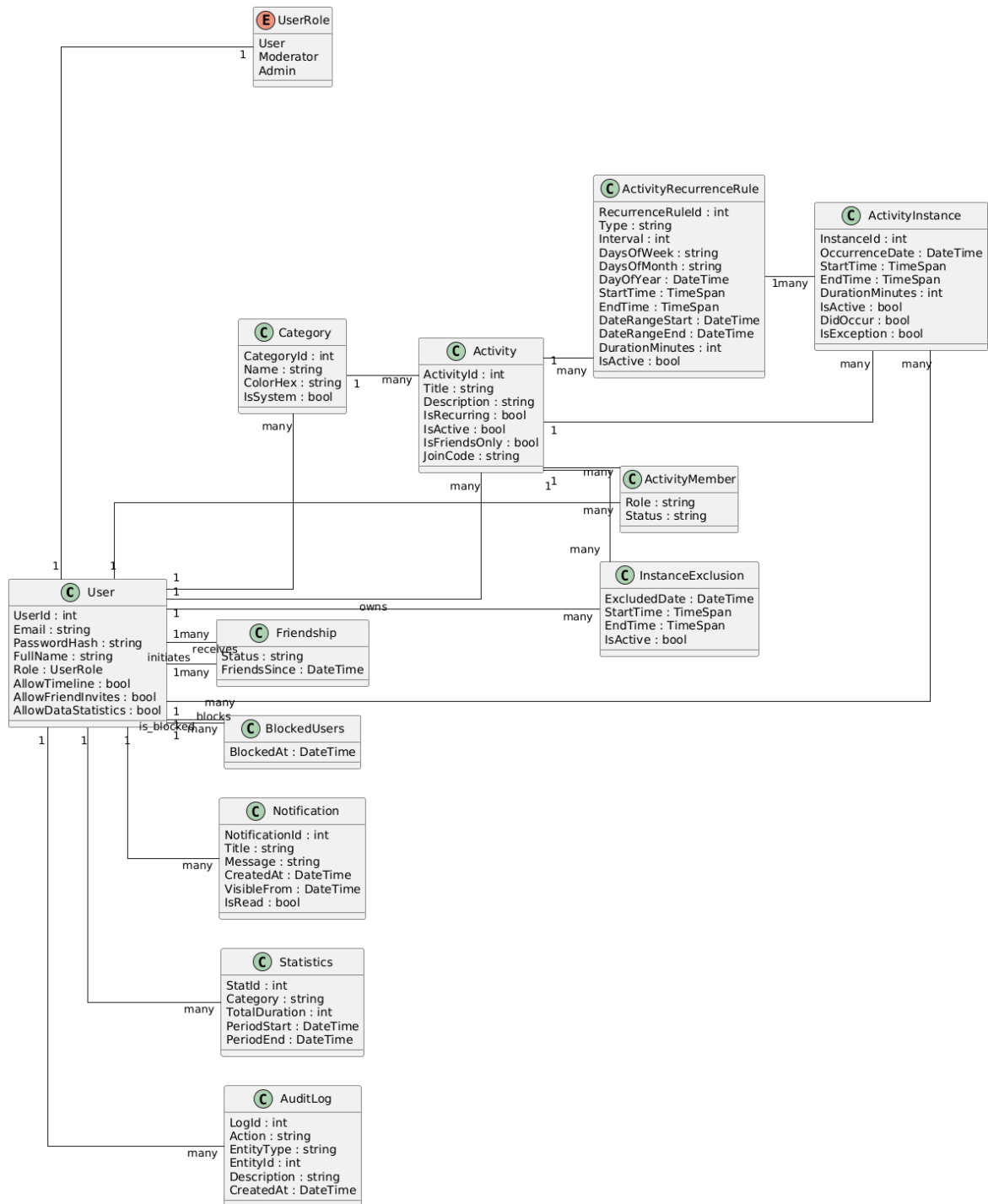
5.3. Struktury danych i organizacji bazy danych

Nie odłączą częścią każdej aplikacji jest baza danych, odpowiedzialna za przechowywanie oraz organizację informacji systemowych. W projekcie została wykorzystana relacyjna baza danych (Rysunek 5.3.1). Implementacja projektowa (Rysunek 5.3.2) została przedstawiona za pomocą diagramu klas znajdujących się w programie.



Rysunek 5.3.1 Diagram bazy danych

Tabele Activity, ActivityRecurrenceRule, ActivityInstance, InstanceExclusion, Category oraz ActivityMember są niezbędne do obsługi aktywności oraz współdzielenia. Tabela User to reprezentacja użytkownika, za funkcjonalność opcji socjalnych obecnych w projekcie odpowiadają tabele BlockedUsers, Friendship. Tabela Statistics przechowuje informacje o statystykach a AuditLog o logach systemowych (Rysunek 5.3.1).



Rysunek 5.3.2 Diagram klas

Diagram klas (Rysunek 5.3.2) odwzorowuje faktyczną strukturę projektową programu. Diagram przedstawia dodatkową strukturę względem poprzedniego (Rysunek 5.3.1) diagramu, jest to struktura typu enum przechowująca role użytkownika.

Szczegółowy opis zakresu funkcjonalności tabel bazy danych:

- User – reprezentuje pojedynczego użytkownika, przechowuje dane z nim związane oraz jego ustawienia prywatności.
- Statistics – przechowuje zagregowane dane statystyczne dotyczące przeszłych aktywności użytkownika.
- Notification – przechowuje powiadomienia stworzone przez użytkownika.
- InstanceExclusion – lista terminów powiązanych z aktywnościami, które algorytm generacji instancji ma pominąć.
- Friendship – reprezentuje relacje znajomości pomiędzy użytkownikami. Relacja jest definiowana poprzez pole statusu – znajomość, blokada, oczekujące.
- Category – reprezentuje kategorie aktywności. Kategorie mogą być systemowe, tworzone na starcie bądź utworzone przez użytkownika. Każda kategoria ma przypisany kolor wykorzystywany w warstwie wizualnej.
- BlockedUsers – przechowuje informacje o blokadach pomiędzy użytkownikami.
- AuditLogs – służy do rejestrowania działań administracyjnych i moderacyjnych w systemie. Zawiera informacje o wykonanej akcji, wykonawcy, identyfikatorze encji i czasie jej wykonania.
- ActivityRecurrenceRule – przechowuje reguły powtarzalności przypisane do aktywności. Określa sposób w cyklicznego generowania instancji, zakres dat, interwał, dni wystąpień, czas trwania
- ActivityMember – tabela pośrednia pomiędzy użytkownikami a aktywnościami współdzielonymi. Przechowuje informacje o roli użytkownika w danej aktywności oraz statusie zaproszenia.
- ActivityInstance – reprezentuje instancje, czyli pojedyncze wystąpienia danej aktywności w czasie. Każda instancja posiada konkretną datę, godzinę rozpoczęcia i zakończenia oraz rzeczywisty czas trwania. Umożliwia rozróżnianie instancji wygenerowanych automatycznie a stworzonych ręcznie oraz obsługuje wyjątki od reguł powtarzalności.
- Activity – przechowuje definicję aktywności tworzonych przez użytkowników. Zawiera podstawowe informacje takie jak tytuł, opis, właściciela oraz opcjonalną kategorię. Dodatkowo określa cykliczność oraz czy dana aktywność jest współdzielona z innymi użytkownikami.

5.4. Implementacja wybranych fragmentów

Podrozdział dedykowany praktycznej implementacji wykorzystanych w systemie algorytmów. Zobrazowane i omówione za pomocą fragmentów kodu są algorytmy omawiane w rozdziale 3.6, wraz z dodatkowym algorytmem odpowiadającym za obsługę generacji osi czasu użytkownika. Dodatkowo wyjaśniona jest budowa przykładowego endpointa wraz z logiką od strony backendowej.

5.4.1. Algorytm sugerujący aktywności

Algorytm sugestii aktywności ma na celu zaproponowanie użytkownikowi najbardziej adekwatną aktywność do umieszczenia na osi czasu, na podstawie jego historycznych zachowań, preferencji czasowych oraz stabilności nawyków. Rozwiązanie opiera się na ważonym modelu punktowym zawierającym elementy logiki rozmytej. Algorytm działa następująco:

1. Pobiera instancje aktywności w danym oknie czasowym (Rysunek 5.4.1)

```
var instanceQuery = _context.ActivityInstances
    .Include(i => i.Activity)
    .ThenInclude(a => a.Category)
    .Where(i =>
        i.UserId == userId &&
        i.IsActive &&
        i.OccurrenceDate >= windowFrom &&
        i.OccurrenceDate <= windowTo);
```

Rysunek 5.4.1 Pobranie instancji w danym oknie czasowym

2. Pobiera unikalne aktywności z historii (Rysunek 5.4.2)

```
var activities = instances
    .Select(i => i.Activity)
    .Where(a => a != null && a.IsActive && a.OwnerId == userId)
    .DistinctBy(a => a.ActivityId)
    .ToList();
```

Rysunek 5.4.2 Pobranie unikalnych aktywności z historii

3. Obliczenia średni czasu trwania i dopasowania do terminu użytkownika (Rysunek 5.4.3)

```
var avgDuration = actInstances.Average(i => (double)i.DurationMinutes);
double muDuration = 1.0;
if (dto.PlannedDurationMinutes.HasValue)
{
    var diff = Math.Abs(dto.PlannedDurationMinutes.Value - avgDuration);

    if (diff <= 15)
        muDuration = Math.Max(0.75, 1.0 - diff * 0.016);
    else
        muDuration = Math.Max(0.05, 0.75 - (diff - 15) * 0.05);

    muDuration = Math.Round(muDuration, 3);
}
```

Rysunek 5.4.3 Fragment kodu odpowiedzialny za średni czas trwania i dopasowanie

4. Sprawdza stabilność nawyku (Rysunek 5.4.4)

```
var totalOccurrences = candidateSlots.Count;
var stabilityFactor = Math.Min(1.0, Math.Log10(totalOccurrences + 1) / 2.0);
```

Rysunek 5.4.4 Warunek liczenia stabilności nawyku

5. Sprawdzenie preferowanych dni tygodnia i przedziału godzinowego (Rysunek 5.4.5)

```
double muDay = 1.0;
if (dto.PreferredDays != null && dto.PreferredDays.Count > 0)
    muDay = dto.PreferredDays.Contains(slot.day) ? 1.0 : 0.3;

double muTime = 1.0;
if (dto.PreferredStart.HasValue && dto.PreferredEnd.HasValue)
{
    var startDiff = Math.Abs((slot.time -
dto.PreferredStart.Value).TotalMinutes);
    var endDiff = Math.Abs((slot.time -
dto.PreferredEnd.Value).TotalMinutes);
    var diff = Math.Min(startDiff, endDiff);

    if (diff <= 15)
        muTime = Math.Max(0.85, 1.0 - diff * 0.01);
    else
        muTime = Math.Max(0.2, 0.85 - (diff - 15) * 0.03);
}

var weighted = 0.25 * muTime + 0.15 * muDay;
var bestWeighted = 0.25 * bestMuTime + 0.15 * bestMuDay;

if (weighted > bestWeighted)
{
    bestMuTime = muTime;
    bestMuDay = muDay;
    bestSlot = slot;
}
```

Rysunek 5.4.5 Fragment kodu odpowiedzialny za obliczenie dopasowania dnia i czasu

6. Podsumowanie wyniku (Rysunek 5.4.6)

```
double score = 0.6 * muDuration + 0.25 * bestMuTime + 0.15 * bestMuDay;
score *= stabilityFactor;
```

Rysunek 5.4.6 Obliczanie końcowej wartości punktowej dopasowania

W alternatywnej wersji algorytmu bazującej na aktywnościach innych osób główną zmianą jest pozbycie się współczynnika stabilności przy obliczaniu wyniku (Rysunek 5.4.6) i zastąpienie go współczynnikiem popularności, który nasycy się przy 25 wystąpieniach danej aktywności (Rysunek 5.4.7).

```
double popularity = Math.Min(1.0, g.Count / 25.0);
double score = 0.5 * muDuration + 0.2 * muDay + 0.2 * muTime + 0.1 *
popularity;
```

Rysunek 5.4.7 Obliczanie alternatywnej końcowej wartości punktowej dopasowania

5.4.2. Algorytm sugerujący umiejscowienie aktywności

Algorytm odpowiada za wyszukiwanie wolnych przedziałów czasowych między istniejącymi aktywnościami, w których możliwe jest umieszczenie nowej aktywności o zadanym czasie trwania. Algorytm wykonuje następujące działania:

1. Ogranicza zakres dat do analizy (Rysunek 5.4.8)

```
var start = dto.StartDate ?? DateTime.UtcNow.Date;
var end = dto.EndDate ?? start.AddDays(15);
```

Rysunek 5.4.8 Zakres dat do analizy, domyślnie 14dni do przodu

2. Pobiera oś czasu użytkownika (Rysunek 5.4.9)

```
await _timelineService.GenerateActivityInstancesAsync(userId, start, end);
var userTimeline = await _timelineService.GetTimelineForUserAsync(userId,
start, end);
```

Rysunek 5.4.9 Zapytanie do bazy danych pobierające oś czasu użytkownika

3. Iteruje dzień po dniu i zbiera luki czasowe (Rysunek 5.4.10)

```
for (var day = start.Date; day <= end.Date; day = day.AddDays(1))
{
    if (onlyDays != null && !onlyDays.Contains(day.DayOfWeek))
        continue;

    var windowStart = day + prefStart;
    var windowEnd = day + prefEnd;

    if (windowEnd <= start || windowStart >= end)
        continue;
    if (windowStart < start) windowStart = start;
    if (windowEnd > end) windowEnd = end;

    var overlaps = events
        .Where(e => e.End > windowStart && e.Start < windowEnd)
        .OrderBy(e => e.Start)
        .ToList();

    var gaps = new List<DateTime s, DateTime e>();

    if (!overlaps.Any())
    {
        // cały dzień (w zadanym oknie) jest wolny
        gaps.Add((windowStart, windowEnd));
    }
    else
    {
        // luka przed pierwszym wydarzeniem
        if (overlaps[0].Start > windowStart)
            gaps.Add((windowStart, overlaps[0].Start));

        // luki pomiędzy wydarzeniami
        for (int i = 0; i < overlaps.Count - 1; i++)
        {
            var gs = overlaps[i].End;
            var ge = overlaps[i + 1].Start;
            if (ge > gs)
                gaps.Add((gs, ge));
        }

        // luka po ostatnim wydarzeniu
        if (overlaps[^1].End < windowEnd)
            gaps.Add((overlaps[^1].End, windowEnd));
    }
}
```

Rysunek 5.4.10 Fragment kodu zbierający luki czasowe

4. Sprawdza czy luka czasowa spełnia kryterium (Rysunek 5.4.11)

```
foreach (var (gs, ge) in gaps)
{
    var gapMinutes = (int)(ge - gs).TotalMinutes;
    if (gapMinutes < minRequired)
        continue;
}
```

Rysunek 5.4.11 Warunek sprawdzający każdą znalezioną lukę

Wynikiem działania algorytmu jest lista dostępnych terminów - data i godzina w których zmieści się aktywność, którą użytkownik chce umiejscowić.

Alternatywna wersja algorytmu (Rysunek 5.4.12) różni się tym, że wyszukuje luk, które są maksymalnie 30 minut krótsze niż zadany przez użytkownika czas, ale przy zwracaniu listy wyników zwraca dodatkowo wydarzenia, które nachodzą na aktywność, którą chcemy wstawić.

```
var overlapping = dayEvents
    .Where(ev => ev.Start < sugEnd && ev.End > sugStart)
    .ToList();

var prev = dayEvents
    .Where(ev => ev.End <= sugStart)
    .OrderByDescending(ev => ev.End)
    .FirstOrDefault();

var next = dayEvents
    .Where(ev => ev.Start >= sugEnd)
    .OrderBy(ev => ev.Start)
    .FirstOrDefault();
```

Rysunek 5.4.12 Dodatkowa filtracja przy okazji szukania luk

Alternatywna wersja również zwraca listę luk czasu między aktywnościami, ale dodatkowo przy każdej z nich wyświetlane są nachodzące aktywności.

5.4.3. Przykładowy endpoint

Aby zachować logiczny porządek oraz przejrzystość w kodzie każdy endpoint składa się z interfejsu (Rysunek 5.4.13) definiującego funkcje serwisu, serwisu zawierającego logikę (Rysunek 5.4.14), oraz kontrolera (Rysunek 5.4.16) do weryfikacji oraz komunikacji. Aby endpoint był widoczny w programie należy go również zarejestrować w kontekście aplikacji (Rysunek 5.4.15).

```
namespace todo_backend.Services.CategoryService
{
    public interface ICategoryService
    {
        Task<CategoryDto?> CreateCategoryAsync(ModifyCategoryDto dto, int id);
    }
}
```

Rysunek 5.4.13 Interfejs serwisu kategorii

```
public async Task<CategoryDto?> CreateCategoryAsync(ModifyCategoryDto dto, int id)
{
    var exists = await _context.Categories
        .AnyAsync(c => c.UserId == id && c.Name == dto.Name);

    if (exists) return null;

    var category = new Category
    {
        UserId = id,
        Name = dto.Name,
        ColorHex = dto.ColorHex,
    };

    _context.Categories.Add(category);
    await _context.SaveChangesAsync();

    return new CategoryDto
    {
        Name = dto.Name,
        ColorHex = dto.ColorHex
    };
}
```

Rysunek 5.4.14 Serwis dodawania kategorii

```
builder.Services.AddScoped<ICategoryService, CategoryService>();
```

Rysunek 5.4.15 Rejestracja serwisu w kontekście

```
[Authorize]
[HttpPost("create-category")]
public async Task<IActionResult> CreateCategory(ModifyCategoryDto dto)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier);
    if (userIdClaim == null) return Unauthorized();
    int userId = int.Parse(userIdClaim.Value);

    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var category = await _categoryService.CreateCategoryAsync(dto, userId);
    return Ok(category);
}
```

Rysunek 5.4.16 Kontroler do serwisu dodającego kategorię

Rozdział 6

Weryfikacja i walidacja

Proces weryfikacji i walidacji obejmował testowanie poszczególnych komponentów aplikacji, integrację nowych funkcjonalności oraz sprawdzenie poprawności działania kluczowych scenariuszy użytkowych oraz weryfikację uprawnień dostępu.

Na etapie implementacji nowych funkcjonalności najpierw każda z funkcjonalności została przetestowana za pomocą interfejsu swagger do testowania endpointów pod kątem błędów logiki programu bądź niepoprawnych danych wejściowych. Podczas implementacji używana była kontrola wersji oprogramowania za pomocą silnika git, przed większymi zmianami obecna wersja systemu była zapisywana, aby zapobiec zepsuciu którejś z istniejących części systemu. Dopiero po potwierdzeniu poprawności działania danej funkcjonalności zostawała ona na stałe wprowadzana w istniejący system.

Dodatkowo zostały przeprowadzone scenariusze użytkowe polegające na wykonywaniu sekwencji operacji odpowiadającym rzeczywistym scenariuszom użytkownika systemu, takich jak rejestracja, tworzenie i edycja aktywności, tworzenie i edycja zasad rekurencyjnych, tworzenie powiadomień, prawidłowe wyświetlanie na widoku kalendarzowym. W podobnym tonie przetestowane zostały funkcjonalności socjalne – dodawanie znajomych, usuwanie znajomych, cofanie zaproszeń, blokowanie użytkowników. Każdy scenariusz przeprowadzany był w pełni po stronie frontendowej systemu aby dodatkowo zweryfikować poprawność wyświetlania wszystkich związanych formularzy i stron.

Weryfikacja poprawności wyświetlania interfejsu na wielu urządzeniach została zrealizowana poprzez otwieranie aplikacji na trzech monitorach o różnych wymiarach oraz poprzez narzędzia deweloperskie google – tutaj głównie tablety i telefony.

Poprawna weryfikacja ograniczeń funkcjonalności z podziałem na role została sprawdzona frontendowo poprzez próbę wejścia na panel administracyjny z poziomu użytkownika – zakończoną niepowodzeniem. Dodatkowo bezpośrednio po stronie backendowej próbowano wywołać endpointy administracyjne z poziomu innych ról, co również zakończyło się niepowodzeniem.

Testowanie algorytmów przebiegało na wykonywaniu ich na specjalnie spreparowanych zbiorach testowych aktywności i użytkowników w celu weryfikacji zwracanych wyników z oczekiwanymi.

Podczas testowania wykrytych zostało parę błędów głównie w logice generowania instancji aktywności, gdzie poprzez niepoprawny argument sprawdzający niemożliwe było utworzenie osobnej pojedynczej instancji danej aktywności w tym samym dniu co istniejąca instancja wygenerowana rekurencyjnie. Wtedy algorytm traktował tą stworzoną ręcznie jako pierwotną i nie generował kolejnej. Dzięki poprawie i doprecyzowaniu argumentu sprawdzającego udało się ten błąd naprawić.

Kolejnym wykrytym problemem był ciekawy przypadek przy powtarzalności ostatnich dni miesiąca, pierwotnie w kodzie jeżeli tryb powtarzalności był miesięczny to do kolejnego wystąpienia dodawana była funkcja `AddMonths(1)`. Jednak w przypadku ostatnich dni miesiąca gdy następujący miesiąc miał mniej dni niż poprzedni to instancje były tworzone niepoprawnie – zamiast w ostatni dzień, to w ten który był ostatni w poprzednim miesiącu. Rozwiązaniem tego problemu dodanie dodatkowego warunku sprawdzającego długość miesiąca i następnie dodatkową inkrementację dni o odpowiednią ilość.

Dodatkowo wykryte zostały błędy związane z niepoprawną walidacją danych wejściowych w paru przypadkach oraz jeden przypadek nadmiarowego usuwania danych przez funkcję czyszczącą nadmiarowe instancje.

Przeprowadzone testy potwierdzają poprawność działania kluczowych funkcji systemu oraz zgodność implementacji z założeniami projektowymi. System umożliwia realizację zaplanowanych scenariuszy użytkowych, a mechaniki kontroli dostępu działają zgodnie z oczekiwaniami. Otrzymane wyniki potwierdzają spełnianie wymagań funkcjonalnych zdefiniowanych w Rozdziale 3.1.

Rozdział 7

Podsumowanie i wnioski

Celem niniejszej pracy dyplomowej było zaprojektowanie oraz implementacja systemu wspomagającego zarządzanie codziennymi aktywnościami użytkownika, obsługę cykliczności aktywności, sugerowanie aktywności na podstawie przeszłych nawyków użytkownika i sugerowanie umiejscowienia aktywności wraz z wglądem na swoje statystyki. Dodatkowo system miał wspierać funkcje socjalne takie jak system znajomych i aktywności współdzielone z innymi użytkownikami. W trakcie produkcji wdrożone zostały dodatkowe funkcjonalności takie jak alternatywne wersje algorytmów, obsługa powiadomień, czy możliwość eksportu obecnego planu tygodnia do formatów drukowalnych (PDF). W ramach realizacji projektu zaprojektowano i wdrożono kompletną aplikację webową, obejmującą warstwę frontendową, backendową oraz relacyjną bazę danych.

Zrealizowany system spełnia wszystkie postawione na etapie analizy i deklaracji wymagania funkcjonalne oraz нефункционалне. Użytkownik ma możliwość rejestracji, logowania, swobodnego tworzenia i modyfikowania aktywności, powiadomień, definiowania reguł powtarzalności i wyświetlanie interaktywnej osi czasu. Zaimplementowane mechanizmy umożliwiają również obsługę aktywności współdzielonych i relacji pomiędzy użytkownikami. Zaimplementowane algorytmy umożliwiające sugerowanie aktywności oraz ich umiejscowienia działają zgodnie z oczekiwaniami. Na podstawie przeprowadzonych testów można stwierdzić, że system działa zgodnie z przyjętymi założeniami projektowymi.

Podczas realizacji projektu napotkano kilka problemów technicznych oraz projektowych. Do najistotniejszego należało zagadnienie powtarzalności aktywności, ponieważ okazało się, że pierwotny model bazy danych jest niewystarczający i wprowadzał duże ograniczenia w regułach powtarzalności. Problem udało się rozwiązać poprzez przemodelowanie części bazy danych związanej z powtarzalnością i aktywnościami oraz dodanie nowych tabel dla pojedynczych instancji oraz wyjątków od reguł. Jako że problem został zauważony podczas implementacji wiązało się to z re-faktoryzacją dużej ilości kodu aby zaadaptować zmiany dyktowane nową strukturą bazy danych. Wprowadzone zmiany rozwiązały problem ograniczeń przy zasadach powtarzalności.

Kolejny problem stanowiło wdrożenie logiki generacji instancji na podstawie zasad powtarzalności wraz z obsługą wyjątków od reguł. Początkowo program generował zdwojone instancje aktywności, bądź pomijał niektóre albo niepoprawnie liczył początki i końce danego tygodnia. Problemy udało się rozwiązać poprzez iteracyjne testowanie i modyfikowanie kodu. Szczególnie dobrym pomysłem okazało się wprowadzanie reguł powtarzalności pojedynczo – najpierw dzienne, potem tygodniowe, następnie miesięczne i roczne.

Obecna wersja systemu może stanowić dobrą podstawę do dalszej rozbudowy funkcjonalności, potencjalne kierunki rozwoju obejmują takie kwestie jak:

- Dodanie funkcjonalności socjalnych – zakładanie grup, komentowanie.
- Wprowadzenie bardziej złożonych algorytmów rekomendacji i sugestii
- Integracja z zewnętrznymi kalendarzami
- Rozbudowanie panelu statystyk o analizę trendów

Dalsze prace mogły również objąć kwestie optymalizacyjne aby zapewnić bezproblemowe doświadczenia z używania jak największej liczbie użytkowników.

Realizacja projektu pozwoliła na praktyczne zastosowanie wiedzy uzyskanej w toku nauczania z zakresu projektowania systemów informatycznych, programowania aplikacji webowych, pracy z relacyjnymi bazami danych oraz projektowania interfejsów. Systematyzacja pracy oraz częsta kontrola wersji podczas implementacji okazały się dwoma kluczowymi czynnikami sprawnego wdrożenia projektu.

Bibliografia

- [1] Brad Aeon, Aida Faber, Alexandra Panaccio,
<https://pmc.ncbi.nlm.nih.gov/articles/PMC7799745/> (dostęp: 21.12.2025)
- [2] Yangyang Fu, Qiuju Wang, Xiaofeng Wang, Haoxuan Zhong, Junqi Chen, Haoyu Fei, Yipeng Yao, Yao Xiao, Wnfu Li, Na Li,
<https://pmc.ncbi.nlm.nih.gov/articles/PMC11967054/> (dostęp: 21.12.2025)
- [3] Valerie P. Jackson, [https://www.jacr.org/article/S1546-1440\(08\)00581-4/fulltext](https://www.jacr.org/article/S1546-1440(08)00581-4/fulltext)
(dostęp: 21.12.2025)
- [4] Mário Nuno Mata, Mariam Sohail, Syed Arslan Haider, José Moleiro Martins,
https://www.researchgate.net/publication/354372403_THE_RELATIONSHIP_BETWEEN_TIME_MANAGEMENT_WORK_STRESS_AND_WORK_PERFORMANCE-A_QUANTITATIVE_STUDY_IN_PORTUGAL (dostęp: 21.12.2025)
- [5] Noorkaran Bhanakar,
https://www.researchgate.net/publication/370134359_Responsive_Web_Design_and_Its_Impact_on_User_Experience (dostęp: 21.12.2025)
- [6] <https://workspace.google.com/intl/pl/products/calendar/> (dostęp: 21.12.2025)
- [7] <https://www.todoist.com/pl> (dostęp: 21.12.2025)
- [8] <https://www.microsoft.com/pl-pl/microsoft-365/microsoft-to-do-list-app> (dostęp: 21.12.2025)
- [9] <https://trello.com/planner> (dostęp: 21.12.2025)
- [10] <https://www.any.do/personal> (dostęp: 21.12.2025)
- [11] <https://www.notion.com/> (dostęp: 21.12.2025)
- [12] <https://code.visualstudio.com/docs> (dostęp: 21.12.2025)
- [13] <https://git-scm.com/docs> (dostęp: 21.12.2025)
- [14] <https://learn.microsoft.com/en-us/visualstudio/windows/?view=visualstudio> (dostęp: 21.12.2025)
- [15] <https://learn.microsoft.com/en-us/dotnet/> (dostęp: 21.12.2025)
- [16] <https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms> (dostęp: 21.12.2025)
- [17] <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17> (dostęp: 21.12.2025)
- [18] <https://swagger.io/docs/> (dostęp: 21.12.2025)
- [19] <https://learn.microsoft.com/en-us/ef/> (dostęp: 21.12.2025)

-
- [20] <https://www.questpdf.com/> (dostęp: 21.12.2025)
 - [21] <https://learn.microsoft.com/enus/aspnet/core/security/authentication/?view=aspnetcore-10.0> (dostęp: 21.12.2025)
 - [22] <https://react.dev/reference/react> (dostęp: 21.12.2025)
 - [23] <https://vite.dev/> (dostęp: 21.12.2025)
 - [24] <https://v2.tailwindcss.com/docs> (dostęp: 21.12.2025)
 - [25] <https://www.npmjs.com/package/jwt-decode> (dostęp: 21.12.2025)
 - [26] <https://lucide.dev/guide/packages/lucide-react> (dostęp: 21.12.2025)
 - [27] <https://reactrouter.com/> (dostęp: 21.12.2025)
 - [28] <https://recharts.github.io/en-US/guide/> (dostęp: 21.12.2025)

Dodatki

Spis skrótów i symboli

<i>DTO</i>	Data transfer object
<i>HTTP</i>	Hypertext Transfer Protocol
<i>HTML</i>	Hypertext Markup Language
<i>CSS</i>	Cascading Style Sheets
<i>JSON</i>	JavaScript Object Notation
<i>API</i>	Application Programing Interface

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- dane testowe
- film pokazujący działanie opracowanego oprogramowania

Spis rysunków

Rysunek 2.1.1 Interfejs aplikacji „Google Calendar”	4
Rysunek 2.1.2 Interfejs Aplikacji „Todoist”	5
Rysunek 2.1.3 Interfejs Aplikacji „Microsoft To-Do”	5
Rysunek 2.1.4 Interfejs aplikacji „Trello”	6
Rysunek 2.1.5 Interfejs aplikacji „Any.do”	6
Rysunek 2.1.6 Interfejs aplikacji „Notion”	7
Rysunek 3.3.1 Diagram przypadków użycia użytkownika	12
Rysunek 3.3.2 Drugi diagram przypadków użycia użytkownika	13
Rysunek 3.3.3 Diagram przypadków użycia pozostałych ról systemowych	14
Rysunek 4.5.1 Widok strony głównej	21
Rysunek 4.5.2 Widok panelu rejestracyjnego	22
Rysunek 4.5.3 Widok panelu do zalogowania	22
Rysunek 4.5.4 Nagłówek nawigacyjny użytkownika	22
Rysunek 4.5.5 Widok kalendarzowy	22
Rysunek 4.5.6 Widok panelu aktywności	23
Rysunek 4.5.7 modal dodawania aktywności	23
Rysunek 4.5.8 Widok aktywności	24
Rysunek 4.5.9 Widok modala dodawania zasad powtarzalności	24
Rysunek 4.5.10 Widok aktywności, wraz z zasadą powtarzalności i instancjami	25
Rysunek 4.5.11 Modal dodawania pojedynczej instancji	25
Rysunek 4.5.12 Widok osi czasu po zmianach	26
Rysunek 4.5.13 Modal tworzenia kategorii	26
Rysunek 4.5.14 Modal edycji aktywności	26
Rysunek 4.5.15 Modal usuwania instancji aktywności	27
Rysunek 4.5.16 Widok panelu aktywności po zmianach	27
Rysunek 4.5.17 Widok kalendarzowy po zmianach	28
Rysunek 4.5.18 Widok panelu znajomych	28
Rysunek 4.5.19 miniprofil znajomego	29
Rysunek 4.5.20 miniprofil nieznajomego	29
Rysunek 4.5.21 Widok aktywności z opcją „Friends-Only”	29
Rysunek 4.5.22 Widok strony aktywności online	29
Rysunek 4.5.23 Modal zapraszania znajomych	30
Rysunek 4.5.24 Alert sukcesu	30
Rysunek 4.5.25 Panel wysłanych zaproszeń	30
Rysunek 4.5.26 Panel otrzymanych zaproszeń	30
Rysunek 4.5.27 Widok na uczestników aktywności online	30
Rysunek 4.5.28 Widok osi czasu z aktywnością online	31
Rysunek 4.5.29 Widok aktywności publicznej	31
Rysunek 4.5.30 Widok kodu dostępu	31
Rysunek 4.5.31 Alert sukcesu	32
Rysunek 4.5.32 Widok uczestnictwa	32

Rysunek 4.5.33 Widok osi czasu z aktywnością publiczną	32
Rysunek 4.5.34 Profil użytkownika z niezatwierdzonymi zmianami	32
Rysunek 4.5.35 Profil użytkownika po zmianach	33
Rysunek 4.5.36 Widok panelu powiadomień	33
Rysunek 4.5.37 Dodawanie nowego powiadomienia	34
Rysunek 4.5.38 Widok istniejących powiadomień	34
Rysunek 4.5.39 Oś tygodnia użytkownika	34
Rysunek 4.5.40 Modal algorytmu sugerującego	35
Rysunek 4.5.41 Jeden z wyników algorytmu sugerującego	35
Rysunek 4.5.42 Alert sukcesu	35
Rysunek 4.5.43 Modal algorytmu ingerującego	36
Rysunek 4.5.44 Jeden z wyników algorytmu ingerującego	36
Rysunek 4.5.45 Alert sukcesu	36
Rysunek 4.5.46 Widok osi czasu po zmianach	37
Rysunek 4.6.1 Wygląd interfejsu panelu moderacyjnego	38
Rysunek 4.6.2 Aktywność przed edycją	39
Rysunek 4.6.3 Modal edycji aktywności	39
Rysunek 4.6.4 Widok aktywności po modyfikacji	39
Rysunek 4.6.5 Widok panelu administracyjnego	40
Rysunek 4.6.6 Wyszukanie aktywności do usunięcia	40
Rysunek 4.6.7 Komunikat potwierdzający	41
Rysunek 4.6.8 Widok po usunięciu aktywności	41
Rysunek 5.3.1 Diagram bazy danych	44
Rysunek 5.3.2 Diagram klas	45
Rysunek 5.4.1 Pobranie instancji w danym oknie czasowym	47
Rysunek 5.4.2 Pobranie unikalnych aktywności z historii	47
Rysunek 5.4.3 Fragment kodu odpowiedzialny za średni czas trwania i dopasowanie	48
Rysunek 5.4.4 Warunek liczenia stabilności nawyku	48
Rysunek 5.4.5 Fragment kodu odpowiedzialny za obliczenie dopasowania dnia i czasu ..	49
Rysunek 5.4.6 Obliczanie końcowej wartości punktowej dopasowania	49
Rysunek 5.4.7 Obliczanie alternatywnej końcowej wartości punktowej dopasowania	49
Rysunek 5.4.8 Zakres dat do analizy, domyślnie 14dni do przodu	50
Rysunek 5.4.9 Zapytanie do bazy danych pobierające oś czasu użytkownika	50
Rysunek 5.4.10 Fragment kodu zbierający luki czasowe	50
Rysunek 5.4.11 Warunek sprawdzający każdą znaną lukę	51
Rysunek 5.4.12 Dodatkowa filtracja przy okazji szukania luk	51
Rysunek 5.4.13 Interfejs serwisu kategorii	52
Rysunek 5.4.14 Serwis dodawania kategorii	52
Rysunek 5.4.15 Rejestracja serwisu w kontekście	52
Rysunek 5.4.16 Kontroler do serwisu dodającego kategorię	53