AKADEMIA TARNOWSKA
Wydział Politechniczny

Kierunek: ***Informatyka***

Specjalność: Informatyka Stosowana

Specjalizacja: Inżynieria Oprogramowania

2023/2024

Bartłomiej Kotapka

PRACA INŻYNIERSKA

Projektowanie i implementacja aplikacji internetowej wspomagającej efektywne zarządzanie czasem i zadaniami

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, 2024

Spis Treści

1. Wstęp.....3

1.1. Motywacja.....3

1.2. Cel pracy.....3

1.3. Zakres pracy.....3

2. Wymagania.....5

2.1. Wymagania biznesowe.....5

2.2. Słownik pojęć.....5

2.3. Wymagania funkcjonalne użytkownika.....5

2.4. Wymagania niefunkcjonalne systemowe.....7

2.5. Reguły biznesowe.....7

3. Projekt aplikacji.....8

3.1. Diagram przypadków użycia.....8

3.2. Diagram ERD.....9

3.3. Diagramy aktywności.....10

3.4. Diagram sekwencji.....17

3.5. Projekt interfejsu użytkownika.....24

4. Opis technologii.....27

4.1. Aplikacja Internetowa.....27

4.2. Intellij IDEA.....27

4.3. Visual Studio Code.....28

4.4. Git.....29

4.5. Apache Maven.....29

4.6. Relacyjna baza danych PostgreSQL.....30

4.7. JPA Java Persistence API.....31

4.8. Spring boot framework.....31

4.9. React.....32

4.10. JavaScript.....33

4.11. Java.....33

5. Wybrane rozwiązania implementacyjne..... 35

6. Testy..... 40

7. Dokumentacja API..... 42

8. Interfejs użytkownika..... 43

9. Podsumowanie..... 51

Bibliografia..... 52

Spis rysunków..... 53

1. Wstęp

1.1. Motywacja

Czas jest najcenniejszą rzeczą jaką ma każdy z nas, jedni mają go więcej drudzy mniej, wpływa on na każdy aspekt naszego życia. Rozważne i przemyślane zarządzanie nim może znacząco poprawić naszą jakość życia, nawet jeśli mamy do dyspozycji ograniczone jego ilości. Jednym z efektywnych systemów zarządzania czasem osobistym jest planowanie i organizacja. Tylko dzięki konsekwentnemu i systematycznemu podejściu do kontrolowania go jesteśmy w stanie realizować długoterminowe cele, które są trudne do osiągnięcia przy braku świadomego zarządzania nim.

1.2. Cel pracy

Każdy z nas ma skończony czas na ziemi, i to jak go wykorzystujemy, ma ogromne znaczenie dla nas samych. Dlatego też ta aplikacja internetowa ma za zadanie pomóc zarządzać nim w odpowiedni sposób. Dzięki niej użytkownik będzie mógł zaplanować swój czas, co pozwoli mu na ustalenie sobie celów, dzięki którym stanie się bardziej produktywny. Ponadto aplikacja będzie gromadziła wszelkie statystyki, które pozwolą na analizę zarządzania czasem.

W osiągnięciu celu wykorzystane zostaną technologie programistyczne, takie jak Java i Spring, które pozwolą na uruchomienie serwera dla aplikacji klienckiej. Natomiast interfejs użytkownika zostanie zaimplementowany przy pomocy ReactJS. To połączenie technologii umożliwi zrealizowanie aplikacji, która pozwoli spełnić wyżej wymieniony cel.

1.3. Zakres pracy

Praca została zorganizowana w osiem rozdziałów, z których niektóre posiadają kilka podrozdziałów. Pierwszy rozdział stanowi wstęp, wprowadzając czytelnika w cel pracy oraz ukazując jej ogólną koncepcję. W drugim rozdziale analizowane są kluczowe wymagania projektu. Rozdział trzeci ukazuje projekt aplikacji, przedstawiając wszelkiego rodzaju diagramy UML a także projekt interfejsu użytkownika. W czwartym rozdziale przedstawiane są narzędzia, które stanowią fundament aplikacji. Rozdział piąty to ukazanie wybranych rozwiązań implementacyjnych. W szóstym rozdziale opisane są przeprowadzane testy.

Natomiast w siódmym rozdziale następuje moment prezentacji końcowego efektu projektu, gdzie widoczny jest wygląd gotowej aplikacji. W ostatnim rozdziale znajduje się podsumowanie pracy wraz z potencjalnymi ścieżkami, które mogą prowadzić do przyszłego rozwoju projektu.

2. Wymagania

W tym rozdziale przedstawione są wymagania aplikacji. Prezentowane są w następującej kolejności: Wymagania biznesowe, wymagania funkcjonalne użytkownika, wymagania niefunkcjonalne użytkownika, na samym końcu znajdują się reguły biznesowe.

2.1. Wymagania biznesowe

Aplikacja ma na celu pomóc w odpowiednim zarządzaniu czasem przy użyciu planera oraz generowanie raportów z zebranych statystyk. Zadaniem oprogramowania jest zaoferowanie użytkownikowi możliwości tworzenia zadań oraz generowanie statystyk. Interfejs jest wykonany w języku angielskim.

2.2. Słownik pojęć

W aplikacji definiujemy kilka kluczowych pojęć:

- Użytkownik - osoba korzystająca z aplikacji.
- Kategoria - określa ogólnie w jaki sposób spędzamy czas.
- Zadanie - określa precyzyjnie w jaki sposób spędzamy czas.
- Raport - są to zebrane statystyki z danego okresu.

2.3. Wymagania funkcjonalne użytkownika

WFU - Wymaganie Funkcjonalne Użytkownika

WFU1.
Użytkownik może zarejestrować się do aplikacji, gdzie będzie wymagane podanie loginu, hasła oraz imienia i nazwiska. Przy czym login musi być unikalny w całej aplikacji.

WFU2.
Użytkownik może zalogować się do aplikacji przy pomocy loginu oraz hasła.

WFU3.
Hasło użytkownika jest przetrzymywane w bazie danych w postaci MD5.

WFU4.

Każdy użytkownik może dodać nowe zadanie. Każde zadanie posiada:

- Kategorię,
- Nazwę zadania.

WFU5.

Każdy użytkownik może zmienić dane w zadaniu. Pola zadania podlegające zmianie to:

- Kategoria
- Nazwa zadania.

WFU6.

Każdy użytkownik systemu może przejść do danego miesiąca i przypisać w nim zadanie do pola poszczególnego dnia.

WFU7.

Każdy użytkownik systemu może edytować czas zadań z poszczególnych dni.

WFU8.

Każdy użytkownik może przeglądać i edytować poprzednie miesiące.

WFU9.

Każdy użytkownik może przejść do raportów, które można pobrać bądź wygenerować na stronie. Użytkownik będzie miał możliwość wyboru przedziału czasowego, z którego ma zostać wygenerowany raport. Raporty prezentują statystyki, które przedstawią czas jaki poświęciliśmy na poszczególne zadania oraz kategorie.

WFU10.

Każdy użytkownik będzie miał możliwość pobrania danego raportu do formatu CSV.

WFU11.

Każdy użytkownik będzie miał możliwość przeglądania i pobrania raportów z różnych terminów.

2.4. Wymagania niefunkcjonalne systemowe

WNS - Wymaganie niefunkcjonalne systemowe

WNS1.
Aplikacja powinna wykorzystywać darmowe rozwiązania dostępne w dziedzinie systemów informatycznych.

WNS2.
Aplikacja powinna być bezpieczna.

WNS3.
Aplikacja powinna być zgodna z przepisami dotyczącymi ochrony danych osobowych (RODO).

WNS4.
Aplikacja powinna być skalowalna, co oznacza, że można ją łatwo rozbudować i dostosować do zmieniających się potrzeb użytkowników bez konieczności znaczących zmian w infrastrukturze.

2.5. Reguły biznesowe

RB - Reguła biznesowa

RB1.
Usunięcie kategorii nie zmieni kategorii przypisanych już zadań.

RB2.
Usunięcie zadania nie usunie zadań, które zostały już użyte.

RB3.
Każde zadanie może mieć tylko jedną kategorię.

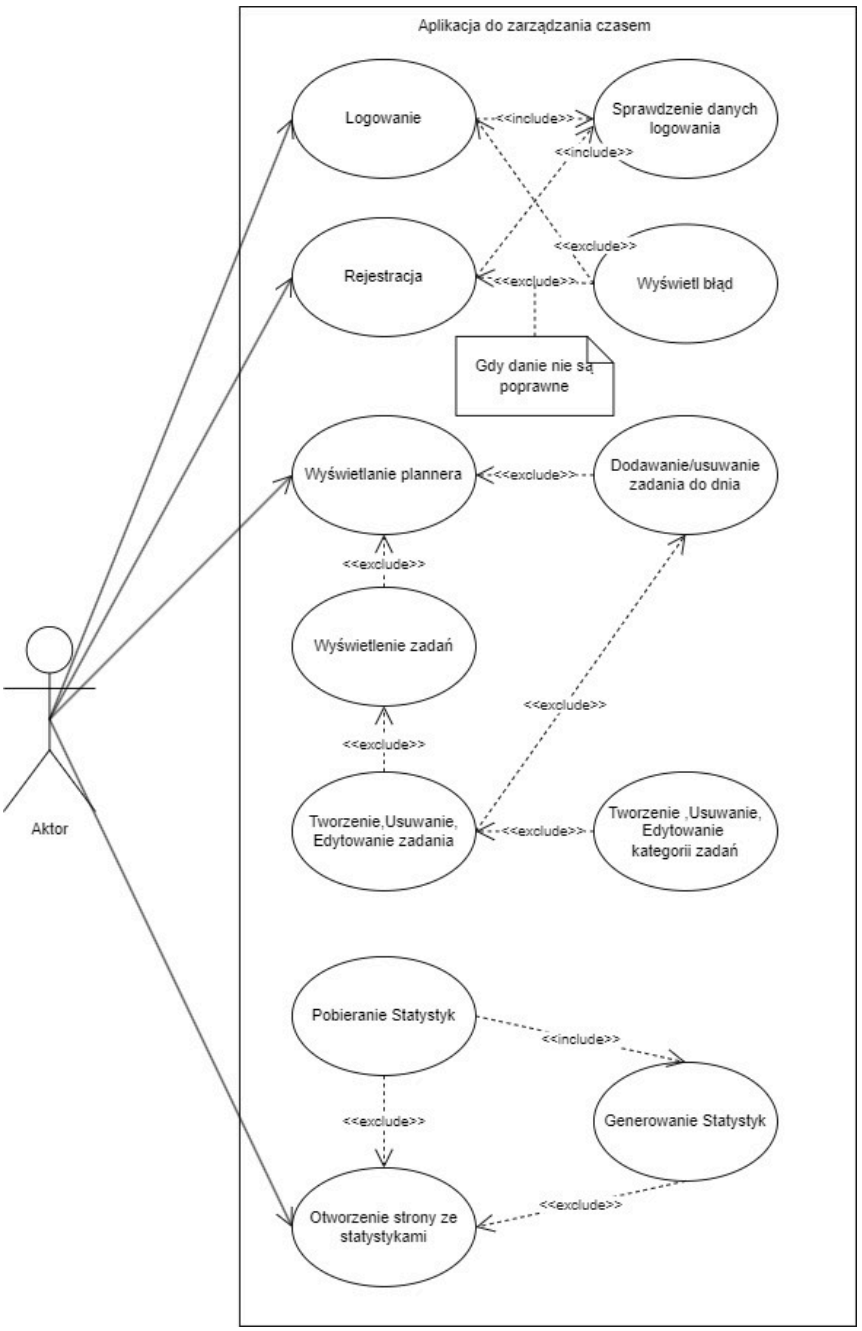
RB4.
Każde zadanie może zostać wykorzystane wielokrotnie.

3. Projekt aplikacji

Przed pisaniem kodu, najlepszym sposobem na zebranie pomysłów i przekształcenie ich w realną rzecz są wszelkiego rodzaju diagramy. Dzięki nim można w łatwy i przejrzysty sposób zaprojektować aplikację, diagramy te mogą przestawiać przypadki użycia przez użytkownika czy też strukturę bazy danych [4].

3.1. Diagram przypadków użycia

Na rysunku 3.1 przedstawiony jest diagram przypadków użycia aplikacji napisany w języku UML. Aktor, którym jest użytkownik posiada wiele interakcji z systemem takie jak rejestracja, logowanie, wyświetlanie planera oraz statystyk. W procesie logowania oraz rejestracji może zostać wyświetlony błąd, jeżeli dane podczas sprawdzania okażą się nieprawidłowe. Przy procesie wyświetlania plannera, aktor ma również możliwość dodawania, edytowania i usuwania zadań, a także przypisywania ich do dni. Natomiast otworzenie panelu statystyk jest rozszerzony o opcję generowania i pobierania ich.

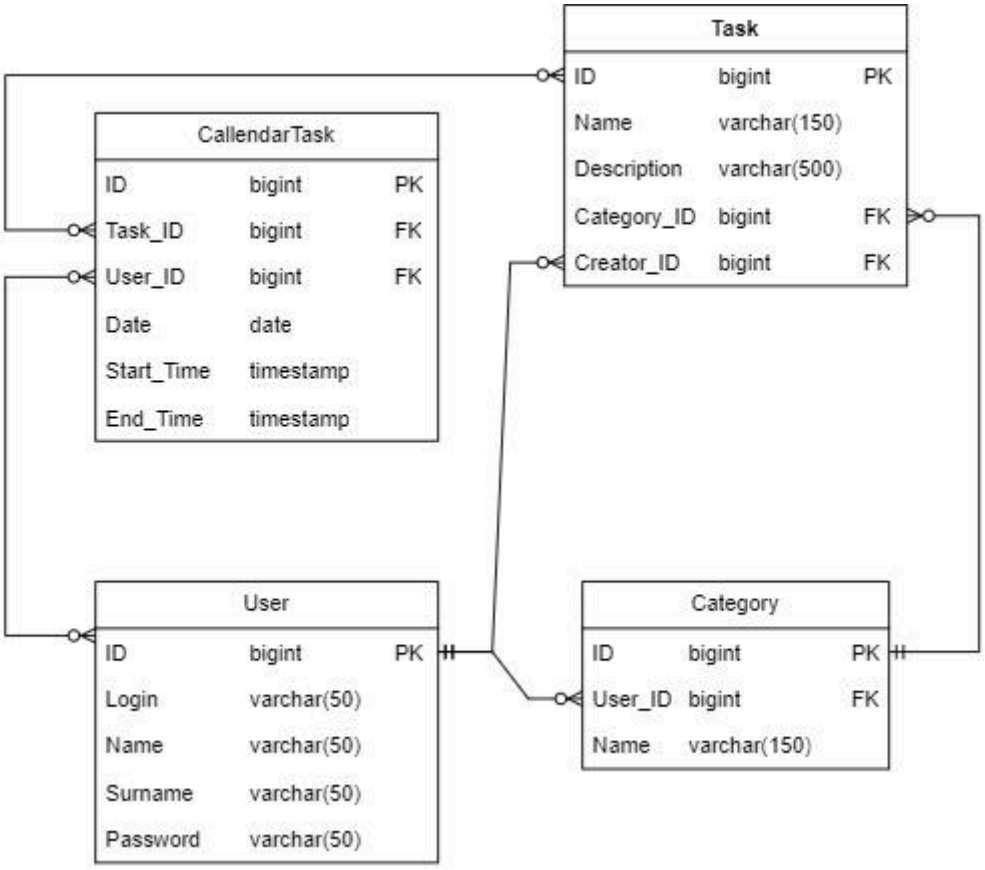


Rys. 3.1. Diagram Przypadków użycia [opracowanie własne]

3.2. Diagram ERD

Drugi diagram przedstawiony na rysunku 3.2 opisuje strukturę bazy danych poprzez przedstawienie obiektów a także relacji między nimi w taki sposób jak przechowywane są w bazie danych. W aplikacji baza posiada cztery encje, pierwsza z nich “User” przedstawiająca użytkownika, posiada on id, login, imię, nazwisko oraz hasło. Druga encja to “Category” przechowująca kategorie, posiada ona id, nazwę kategorii oraz klucz obcy użytkownika, aby

móc zidentyfikować, które kategorie należą do danego użytkownika. Tak samo jest w encji "Task", która oprócz posiadania klucza obcego użytkownika posiada również klucz obcy kategorii, gdyż zadanie nie może istnieć bez kategorii. Poza tym zadanie posiada id, nazwę oraz opis. Ostatnią najważniejszą encją jest "CalendarTask", która to jest odpowiedzialna za przechowywanie przypisanych tasków, ona również posiada klucz obcy użytkownika oraz zadania, ponadto ma id, datę dnia do którego przypisano zadanie oraz datę początkową i końcową zadania.

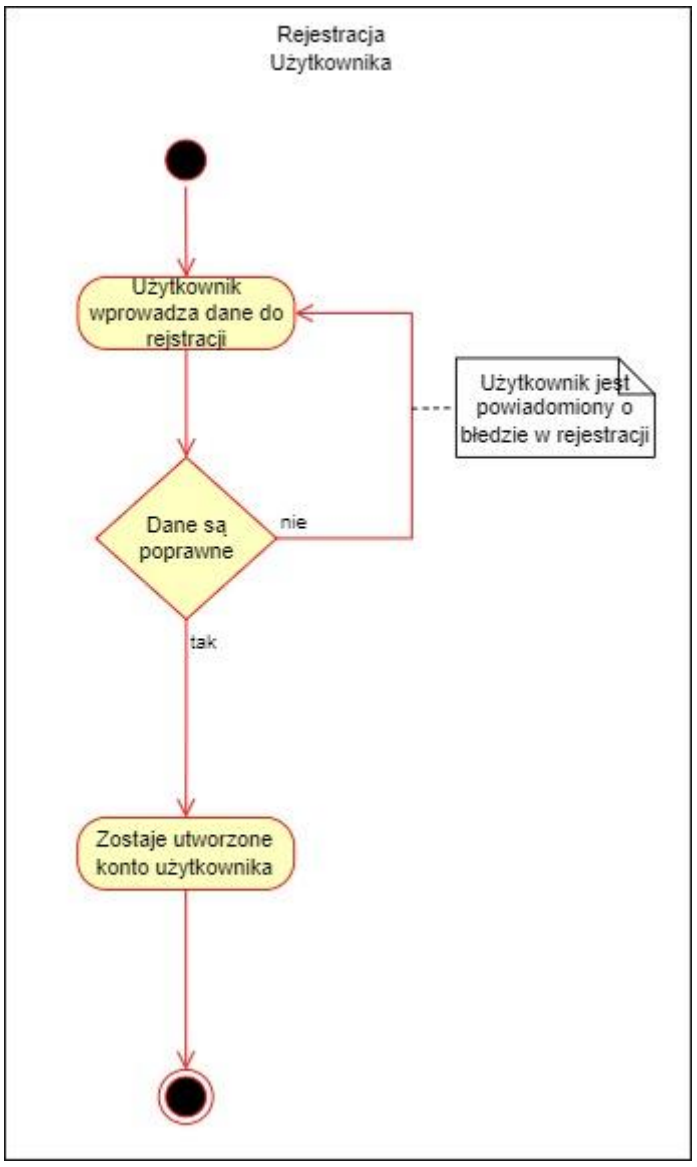


Rys. 3.2. Diagram ERD [opracowanie własne]

3.3. Diagramy aktywności

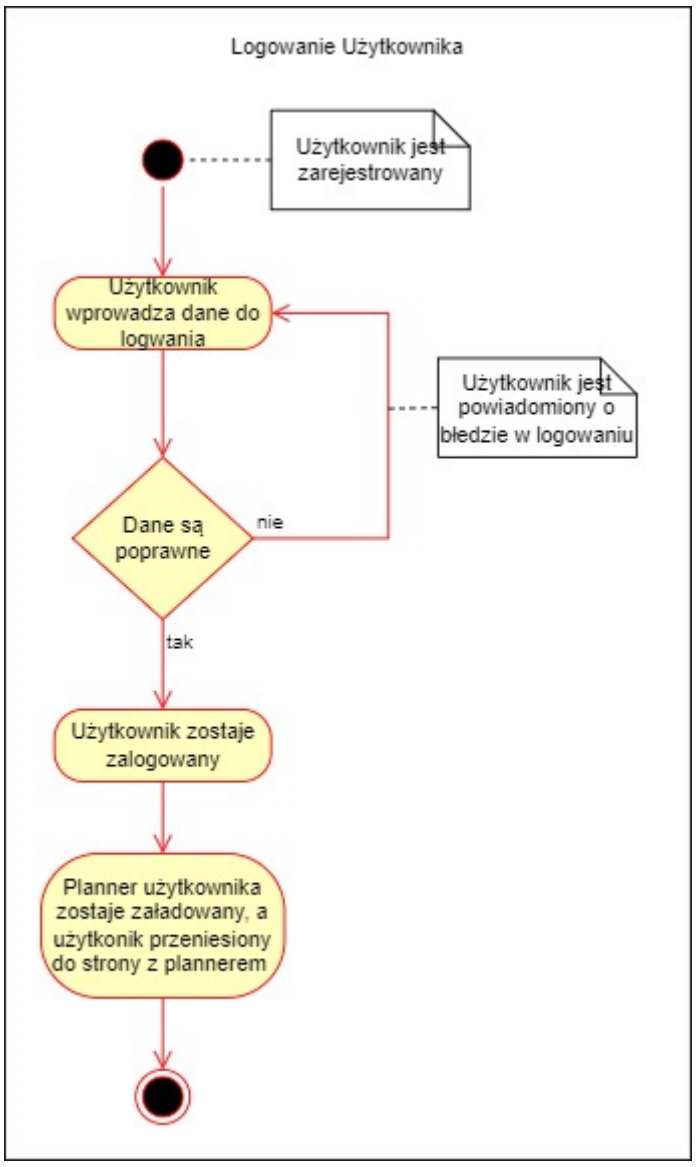
W diagramach UML, kolejnym istotnym typem są diagramy aktywności, które pozwalają na zobrazowanie sekwencji operacji i działań w systemie.

Rysunek 3.3 prezentuje proces rejestracji użytkownika. Użytkownik najpierw wprowadza dane potrzebne do rejestracji, następnie system sprawdza ich poprawność. Jeżeli dane są prawidłowe to konto zostaje utworzone.



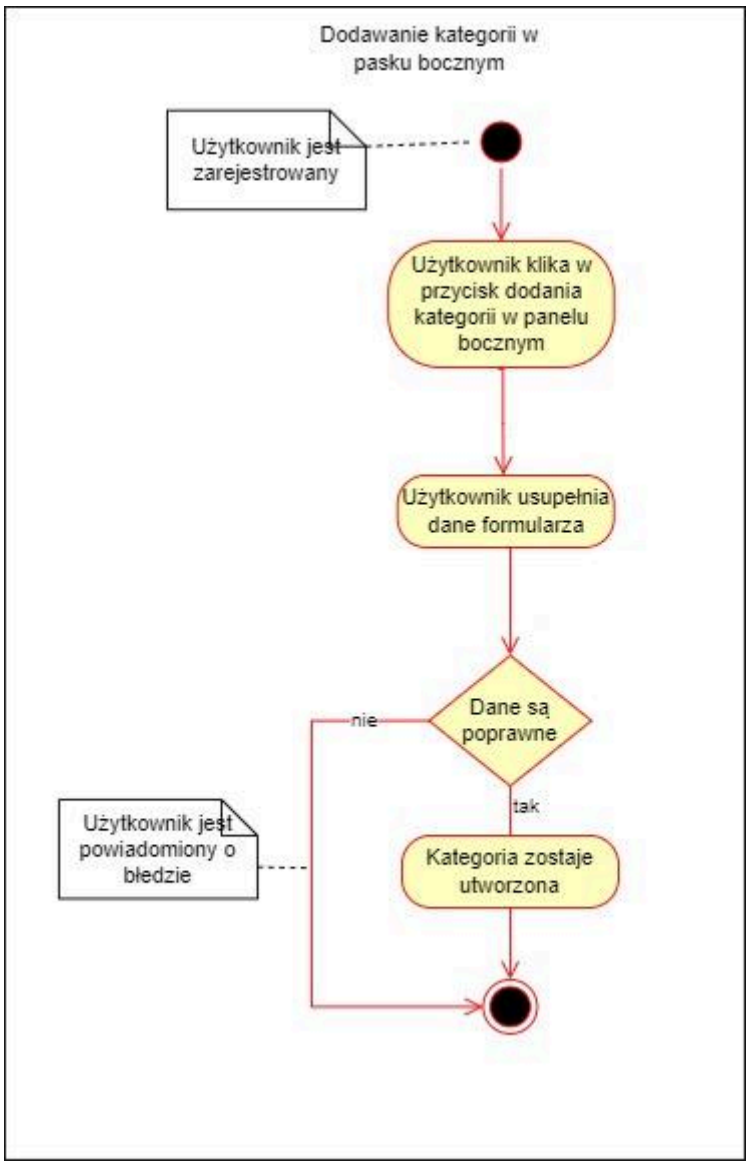
Rys. 3.3. Diagram aktywności dla rejestracji [opracowanie własne]

Podobnie do rejestracji, lecz z pewnymi modyfikacjami, przedstawia się diagram logowania na rysunku 3.4. Jedyna różnica jest taka, że w momencie gdy użytkownik poda poprawne dane to po zalogowaniu zostanie przeniesiony do strony z plannerem.



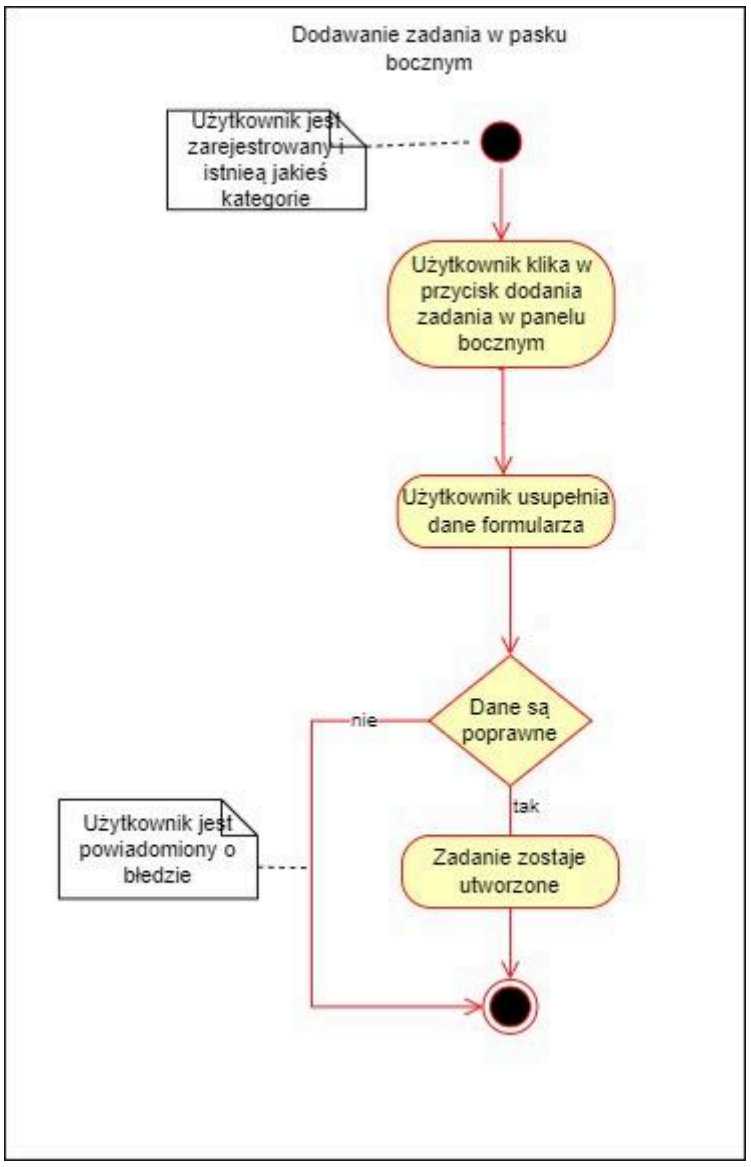
Rys. 3.4. Diagram aktywności dla logowania[opracowanie własne]

Rysunek 3.5 przedstawia tworzenie kategorii, które to później są wykorzystywane w zadaniach, aby je łatwiej pogrupować. Użytkownik musi być zarejestrowany, aby rozpocząć proces. Następnie klika w przycisk dodawania kategorii na panelu bocznym i uzupełnia formularz. Jeżeli dane są poprawne to kategoria zostanie utworzone, jeżeli nie to zostanie wyświetlony komunikat o błędzie.



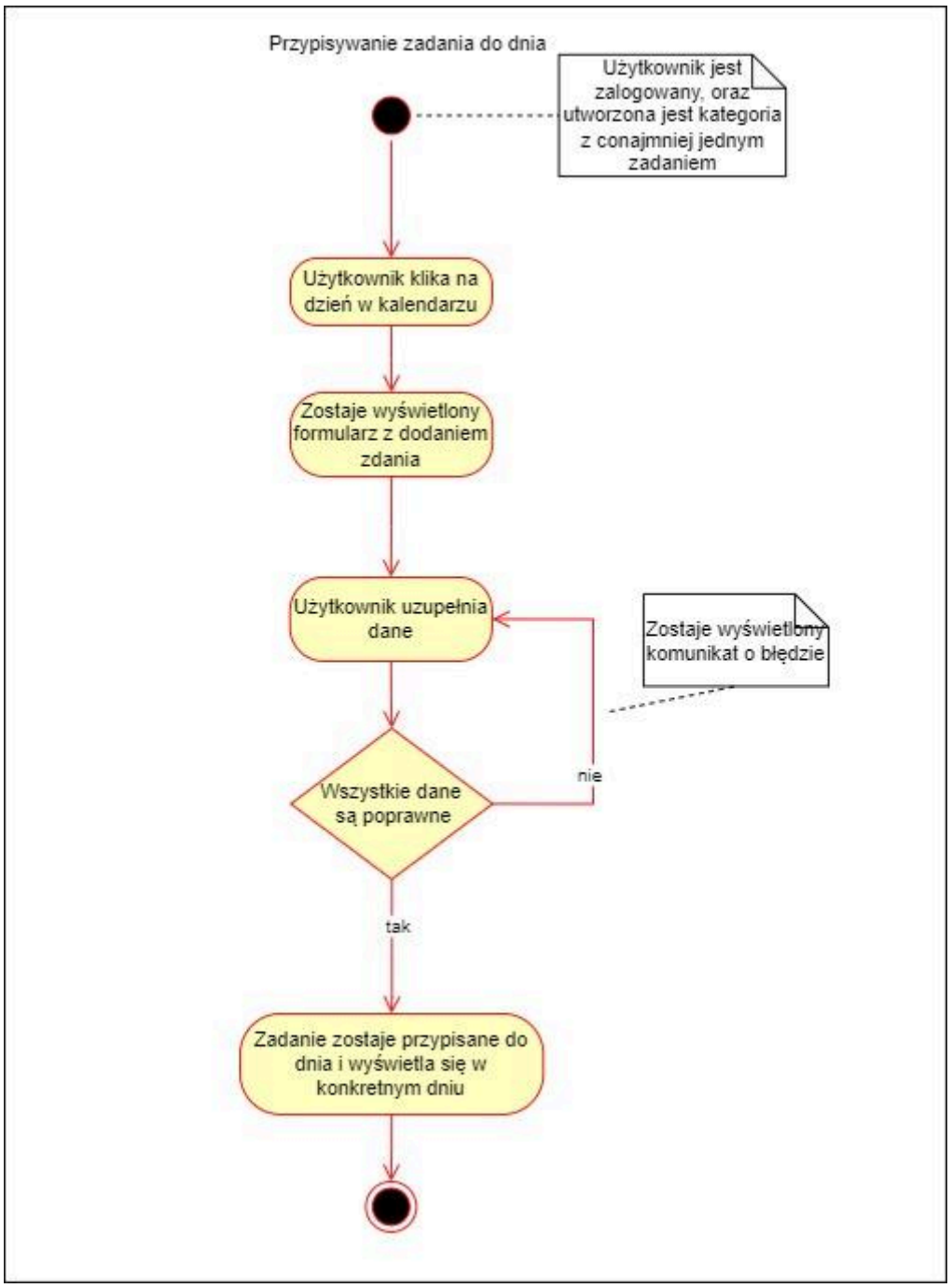
Rys. 3.5. Diagram aktywności dla dodawania kategorii [opracowanie własne]

Rysunek 3.6 skupia się na tworzeniu zadań. Przed utworzeniem zadania wymagane jest aby użytkownik był zalogowany i istniała co najmniej jedna kategoria, gdyż zadanie w systemie zawsze musi takową posiadać. Gdy te warunki są spełnione użytkownik może kliknąć w przycisk do tworzenia zadań. Następnie wyświetli się formularz, a po poprawnym jego uzupełnieniu zadanie zostanie utworzone. W przypadku gdy dane będą nieprawidłowe zostanie wyświetlony komunikat.



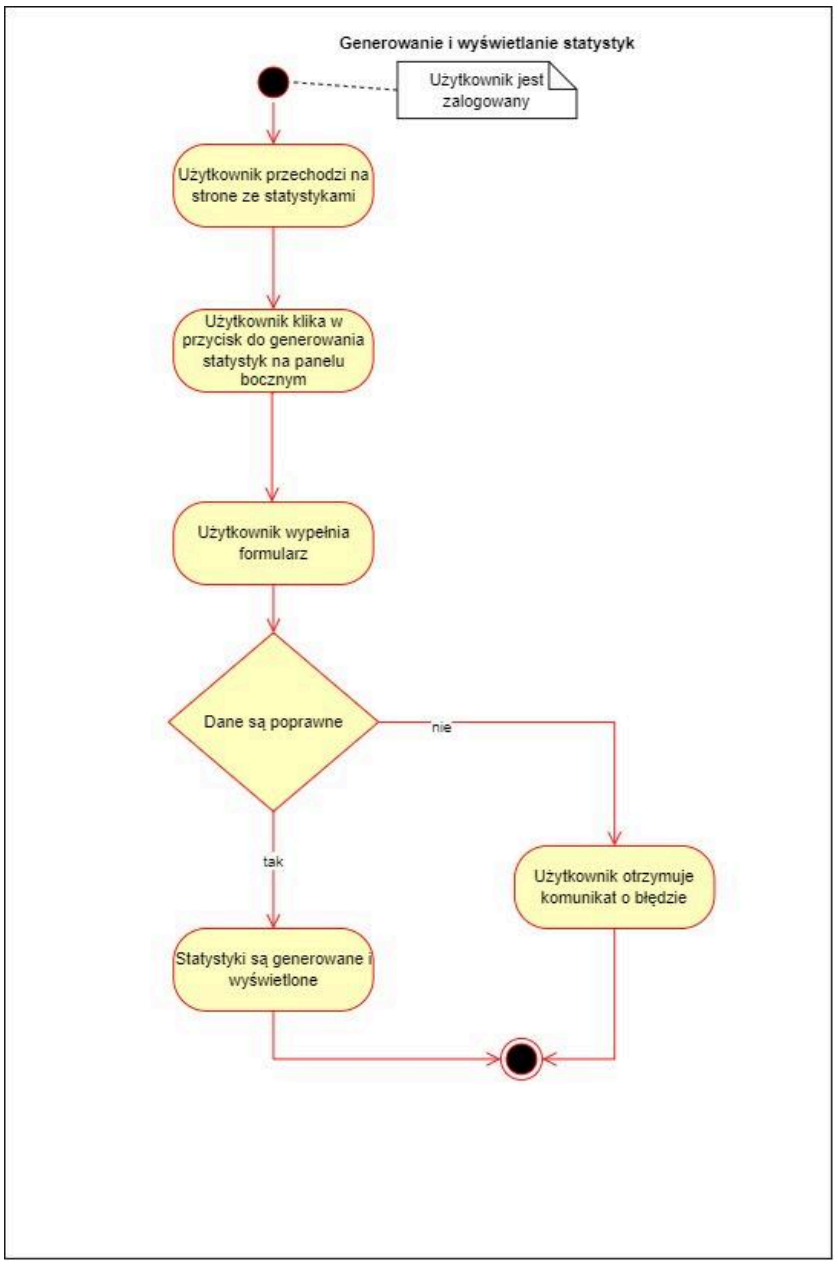
Rys. 3.6. Diagram aktywności dla dodawania zadania [opracowanie własne]

Rysunek 3.7 koncentruje się na jednym z głównych wymagań aplikacji, przedstawiając proces przypisywania zadania do konkretnego dnia. Aby przypisać zadanie do dnia wymagane jest, aby użytkownik był zalogowany i istniała co najmniej jedna kategoria z zadaniem. Następnie użytkownik wybiera dzień w kalendarzu i wypełnia formularz, jeżeli wypełni go poprawnie to zadanie zostanie przypisane do wybranego dnia z danymi z formularza, a jeżeli nie to zostanie wyświetlony komunikat o błędzie.



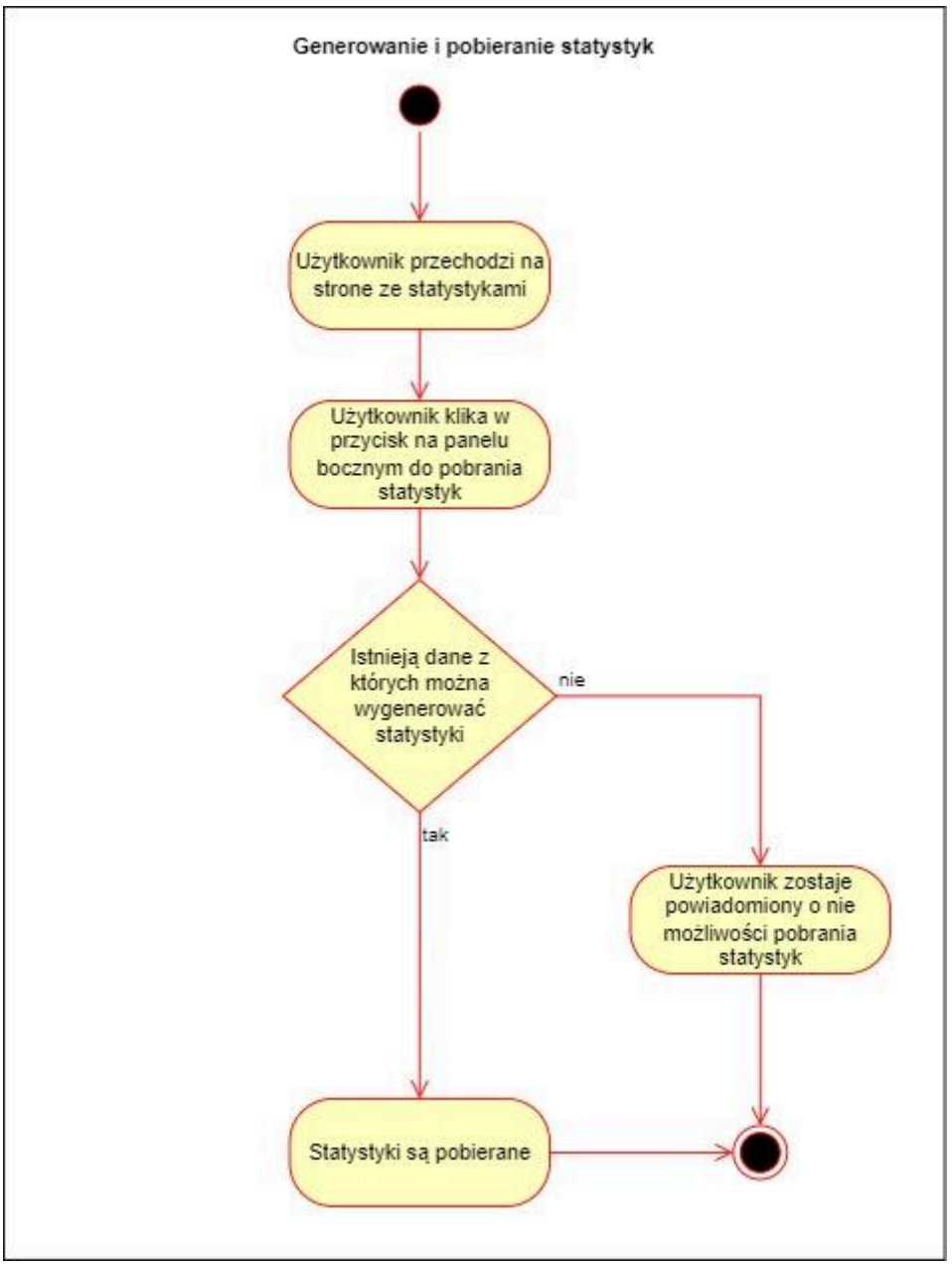
Rys. 3.7. Diagram aktywności dla przypisywania zadania do dnia[opracowanie własne]

Rysunek 3.8 koncentruje się na generowaniu statystyk w aplikacji. Aby móc zacząć proces użytkownik musi być zalogowany. Następnie po przejściu na stronę ze statystykami klika w przycisk do generowania ich, gdzie wypełnia formularz i po pozytywnej weryfikacji danych, statystyki zostaną wygenerowane i wyświetlone na stronie.



Rys. 3.8. Diagram aktywności dla generowania i wyświetlania statystyk [opracowanie własne]

Rysunek 3.9 również dotyczy statystyk, w tym wypadku pobierania ich. Żeby móc zacząć proces użytkownik musi być zalogowany. Następnie po przejściu na stronę ze statystykami klika w przycisk do pobrania ich, gdzie po sprawdzeniu danych, statystyki zostaną pobrane.

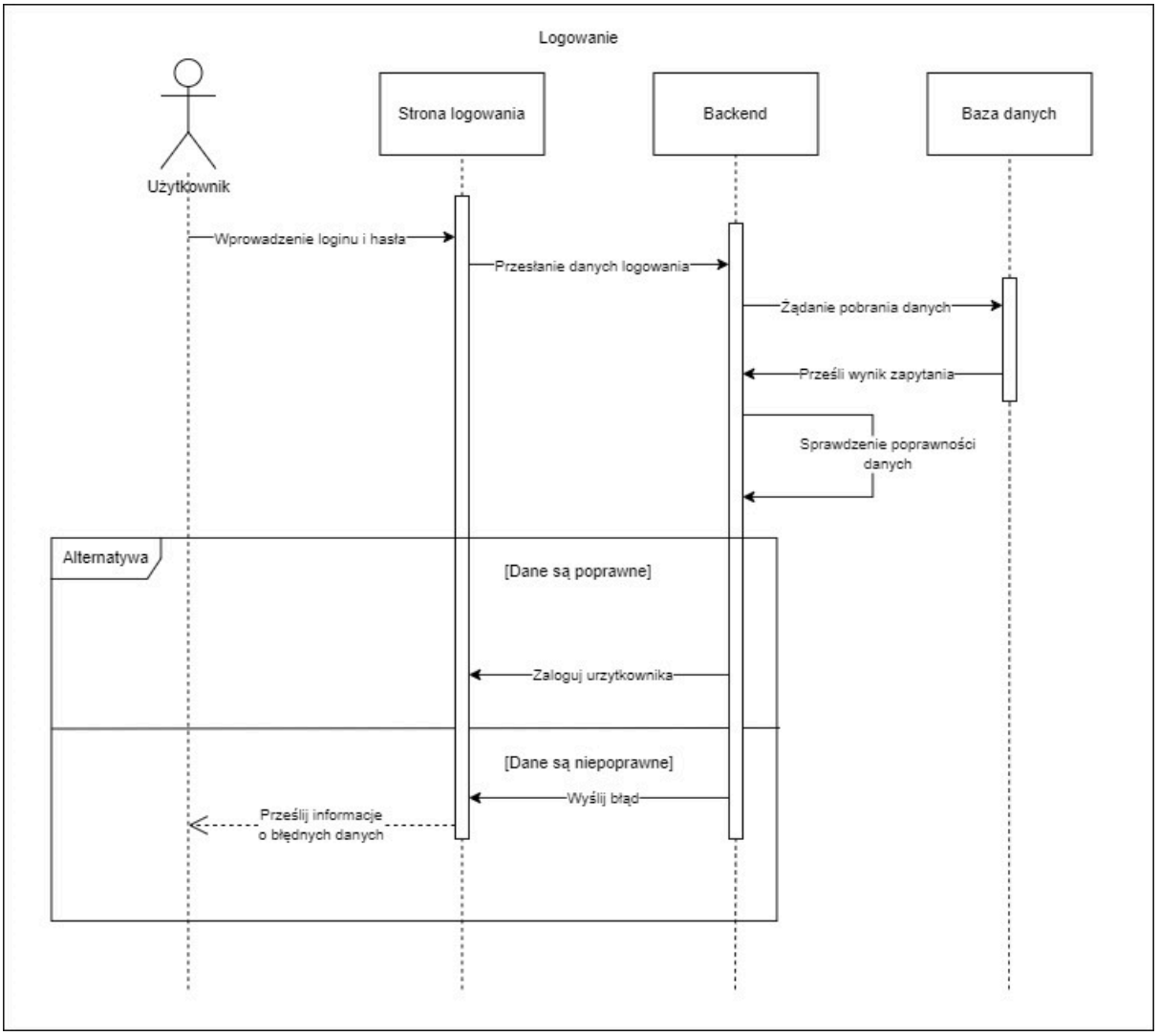


Rys. 3.9. Diagram aktywności dla generowania i pobierania statystyk [opracowanie własne]

3.4. Diagram sekwencji

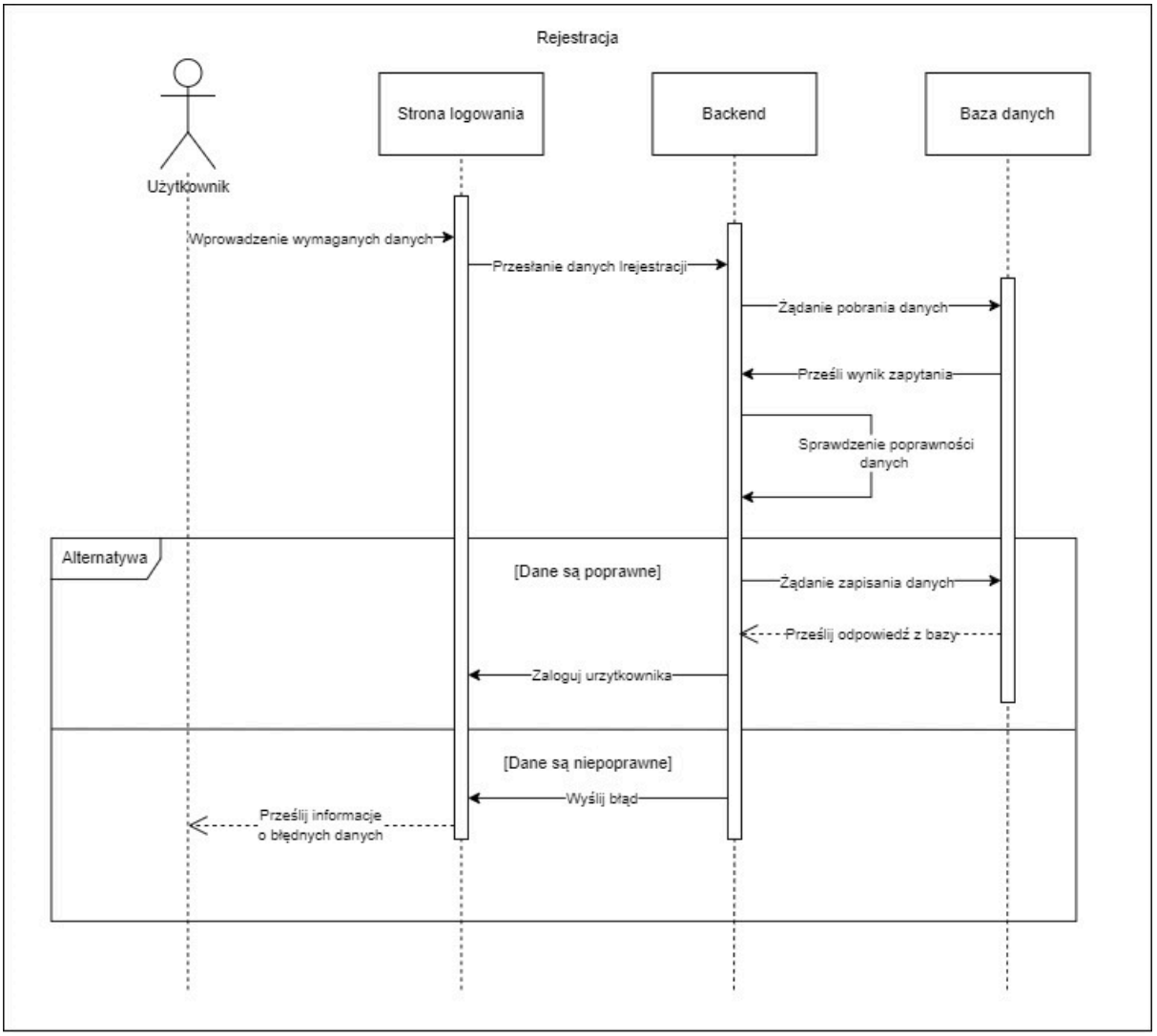
Diagramy sekwencji służą do przedstawiania interakcji i komunikacji pomiędzy różnymi elementami systemu. Dzięki nim możemy zobrazować chronologie wykonywania działań.

Rysunek 3.10 ukazuje diagram sekwencji dla procesu logowania istniejącego użytkownika. Użytkownik najpierw wprowadza swoje dane, które następnie są sprawdzane na serwerze. W pozytywnym przypadku użytkownik zostanie zalogowany, przy negatywnym sprawdzeniu zostaje powiadomiony o błędzie przy logowaniu.



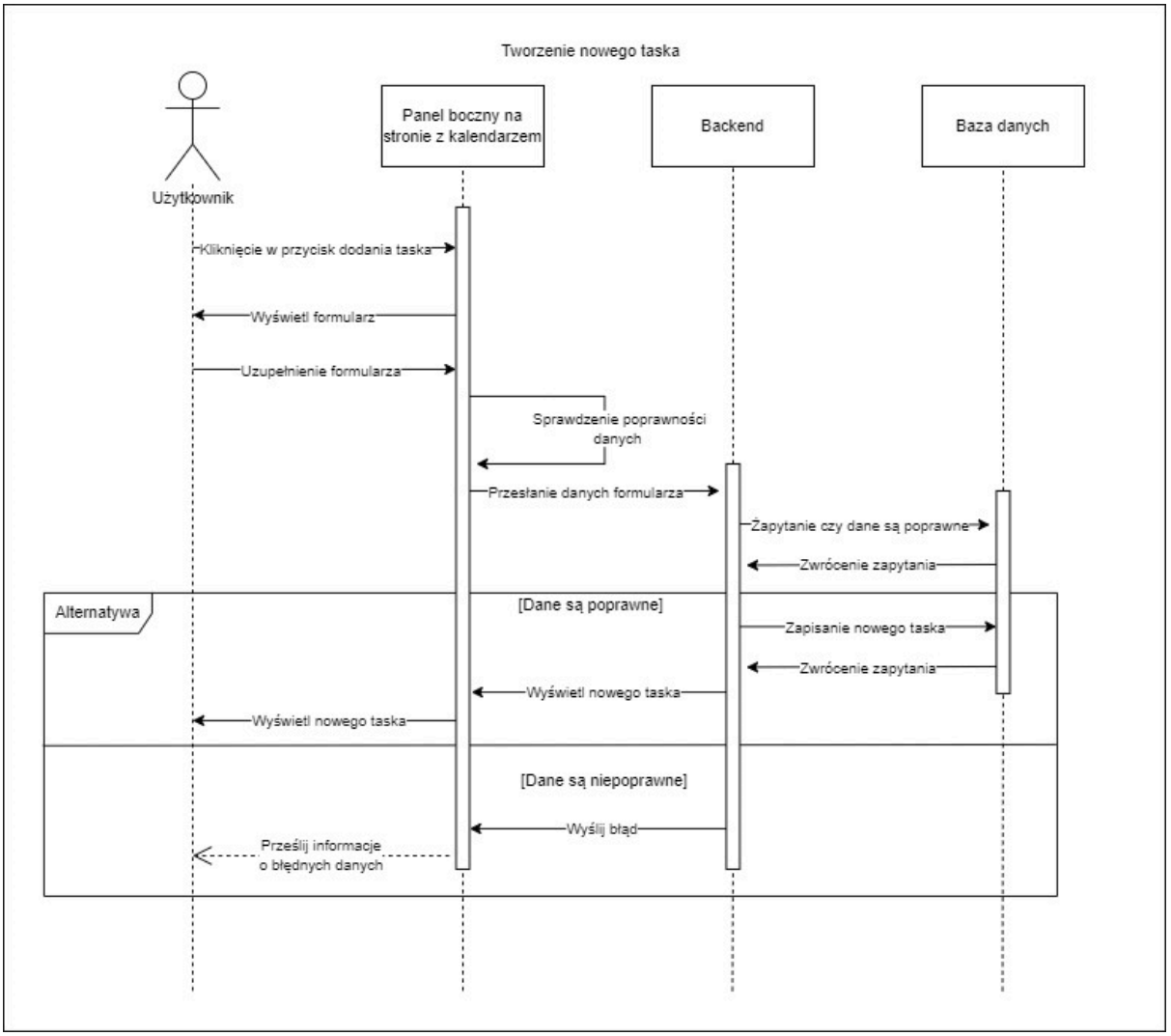
Rys. 3.10. Diagram sekwencji dla logowania [opracowanie własne]

Rysunek 3.11 jest podobny do poprzedniego (rysunek 3.10), lecz koncentruje się na sekwencji związanej z rejestracją nowego użytkownika. Tutaj również użytkownik wprowadza dane do formularza tym razem dla rejestracji, dane te następnie przesyłane są do backendu, gdzie sprawdzana jest możliwość utworzenia nowego konta. Jeżeli weryfikacja będzie prawidłowa to konto zostanie utworzone a użytkownik zalogowany. W przypadku nieprawidłowości zostanie przesłany komunikat.



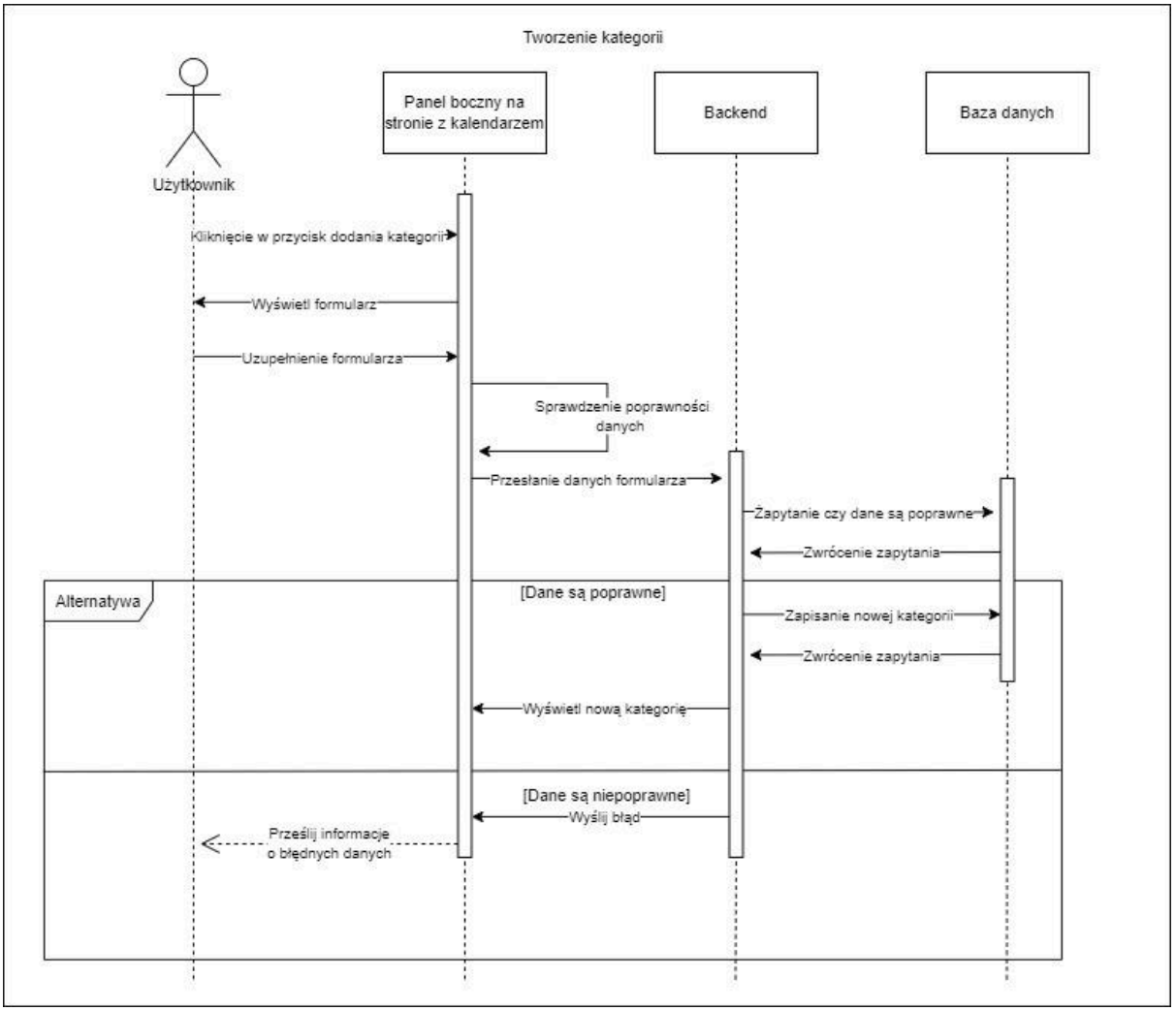
Rys. 3.11. Diagram sekwencji dla rejestracji [opracowanie własne]

Rysunek 3.12 przedstawia proces tworzenia nowego zadania w panelu bocznym. Użytkownik rozpoczyna proces od przyciśnięcia w przycisk dodania taska, a następnie wypełnia formularz. Dane są następnie przesyłane do backendu, gdzie następuje ich walidacja. Jeżeli dane są prawidłowe to zadanie zostanie zapisane w bazie danych i wyświetlone, w przeciwnym wypadku użytkownik otrzyma odpowiedni komunikat.



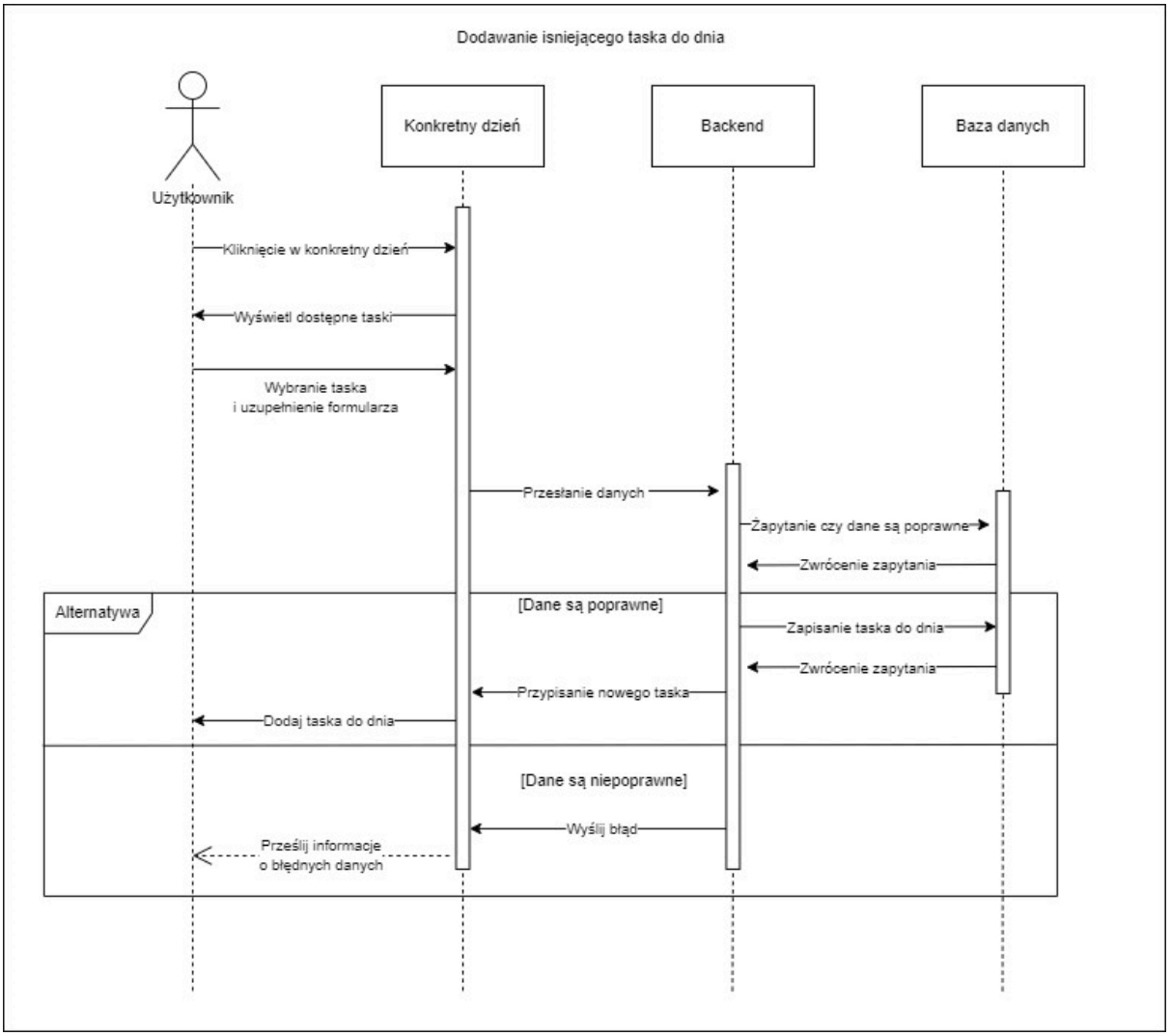
Rys. 3.12. Diagram sekwencji dla tworzenia nowego taska [opracowanie własne]

Rysunek 3.13 skupia się na tworzeniu nowej kategorii w panelu bocznym. Użytkownik musi przycisnąć przycisk dodania kategorii, a następnie uzupełnić formularz. Dane z formularza zostaną przesłane do backendu i przeanalizowane. W przypadku, gdy dane będą prawidłowe kategoria zostanie zapisana w bazie danych oraz wyświetlona użytkownikowi.



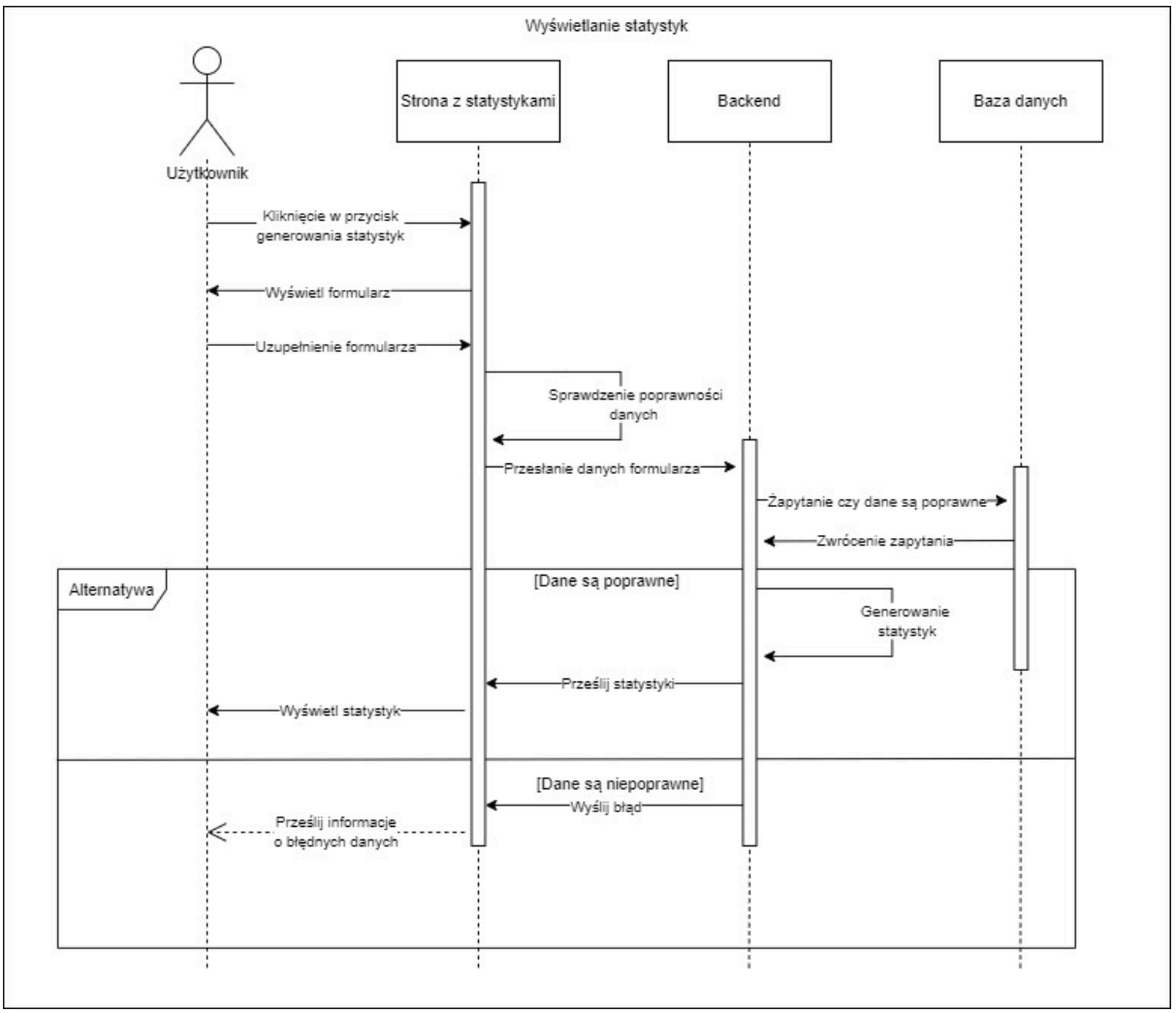
Rys. 3.13. Diagram sekwencji dla tworzenia nowej kategorii [opracowanie własne]

Rysunek 3.14 przedstawia dodawanie istniejącego zadania do konkretnego dnia w kalendarzu. Aby rozpocząć proces użytkownik musi kliknąć na konkretny dzień z kalendarza, następnie zostaną wyświetlone dostępne zadania i użytkownik będzie musiał wybrać jedno i uzupełnić formularz. Następnie dane te zostaną poddane walidacji na backendzie i jeżeli przejdzie ona pomyślnie to zadanie zostanie zapisane w bazie danych oraz będzie widoczne w kalendarzu użytkownika.



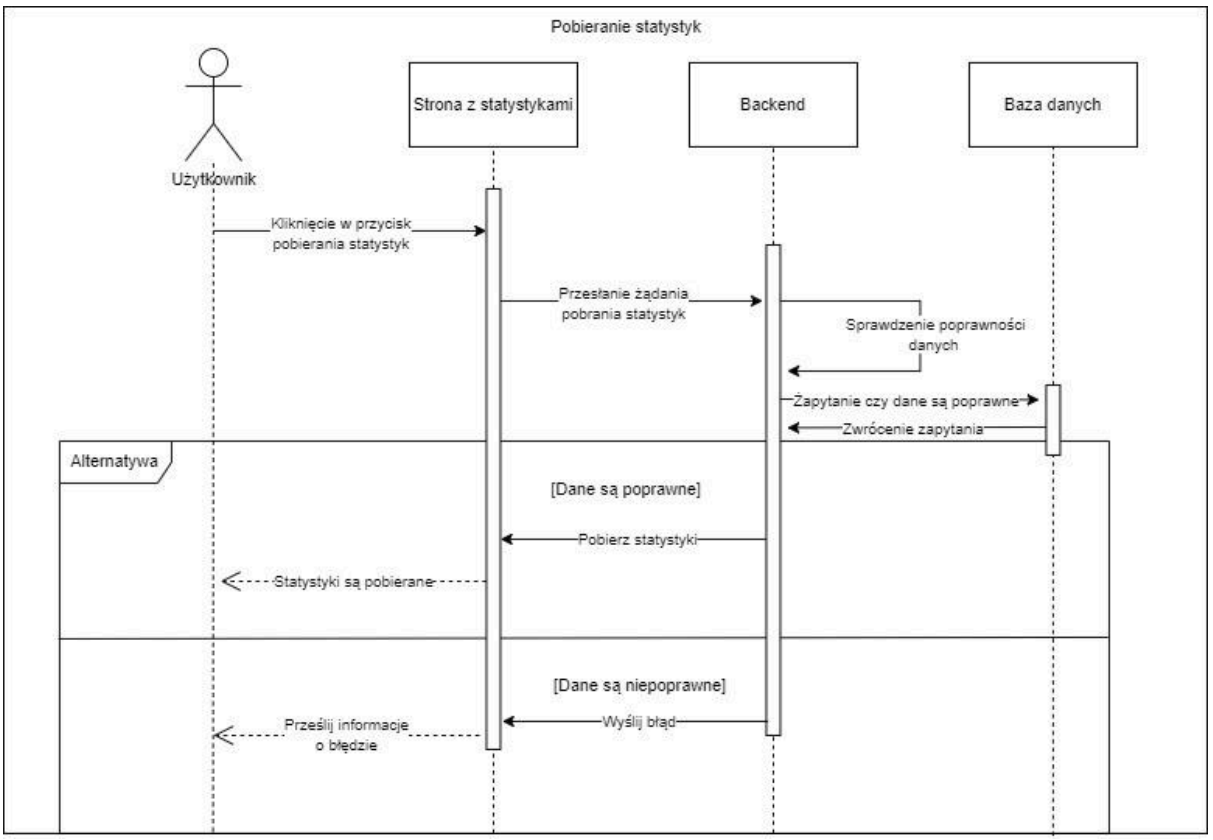
Rys. 3.14. Diagram sekwencji dla dodawania zadania do dnia [opracowanie własne]

Rysunek 3.15 koncentruje się na generowaniu i wyświetlaniu statystyk. Użytkownik klika na przycisk znajdujący się na stronie ze statystykami. Następnie wyświetlany jest formularz i po uzupełnieniu przesyłany jest do backendu, gdzie jest walidowany. Jeżeli dane są poprawne to statystyki zostaną wygenerowany, a następnie wyświetlone użytkownikowi, jeżeli nie to strona ze statystykami otrzymuje komunikat o błędzie.



Rys. 3.15. Diagram sekwencji dla generowania i wyświetlania statystyk [opracowanie własne]

Rysunek 3.16 również jest związany ze statystykami, ale w tym wypadku zamiast wyświetlać, pobiera je. Aby mogło do tego dojść, użytkownik musi przycisnąć przycisk pobrania statystyk. Gdy zostanie to wykonane, backend otrzyma żądanie pobrania statystyk i przeanalizuje czy jest to możliwe. W pozytywnym przypadku statystyki zostaną pobrane, a w negatywnym użytkownik otrzyma informację o błędzie.

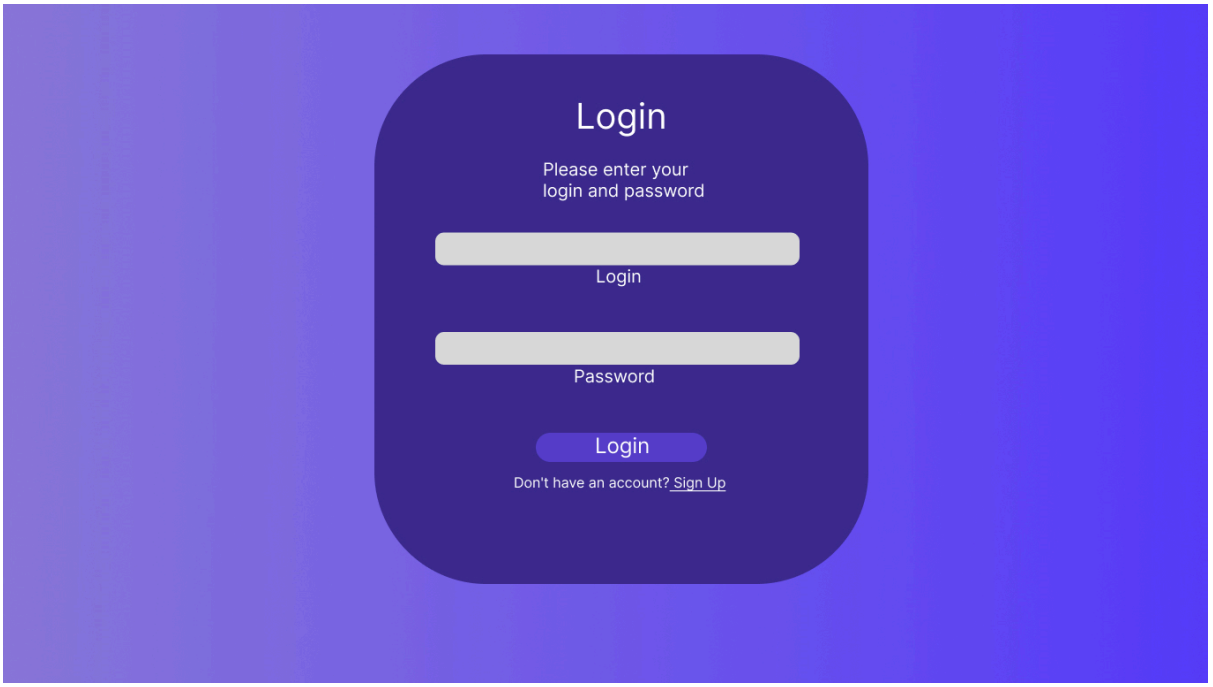


Rys. 3.16. Diagram sekwencji dla generowania i pobierania statystyk [opracowanie własne]

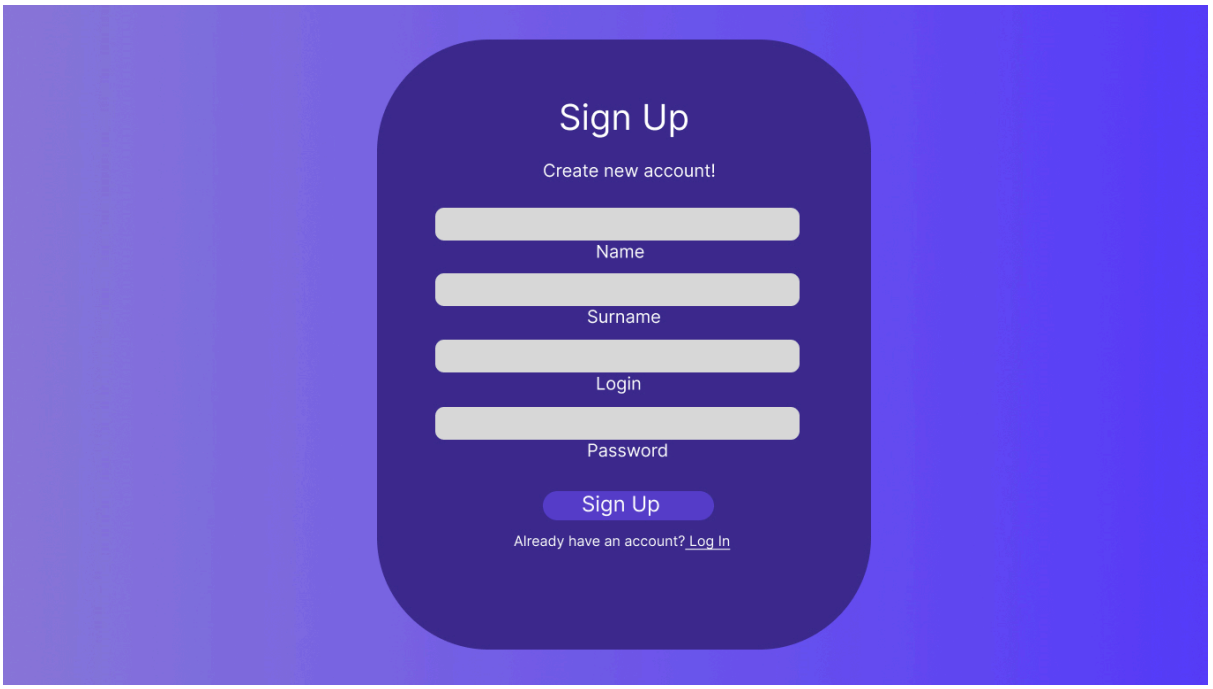
3.5. Projekt interfejsu użytkownika

Przed utworzeniem właściwego interfejsu użytkownika należy przygotować projekt. Znakomitym narzędziem do tego zadania jest aplikacja Figma. Można za jej pomocą tworzyć projekty interfejsów online, dodawać osoby do projektów, posiada również wersjonowanie, czyli możliwość zapisywania zmian.

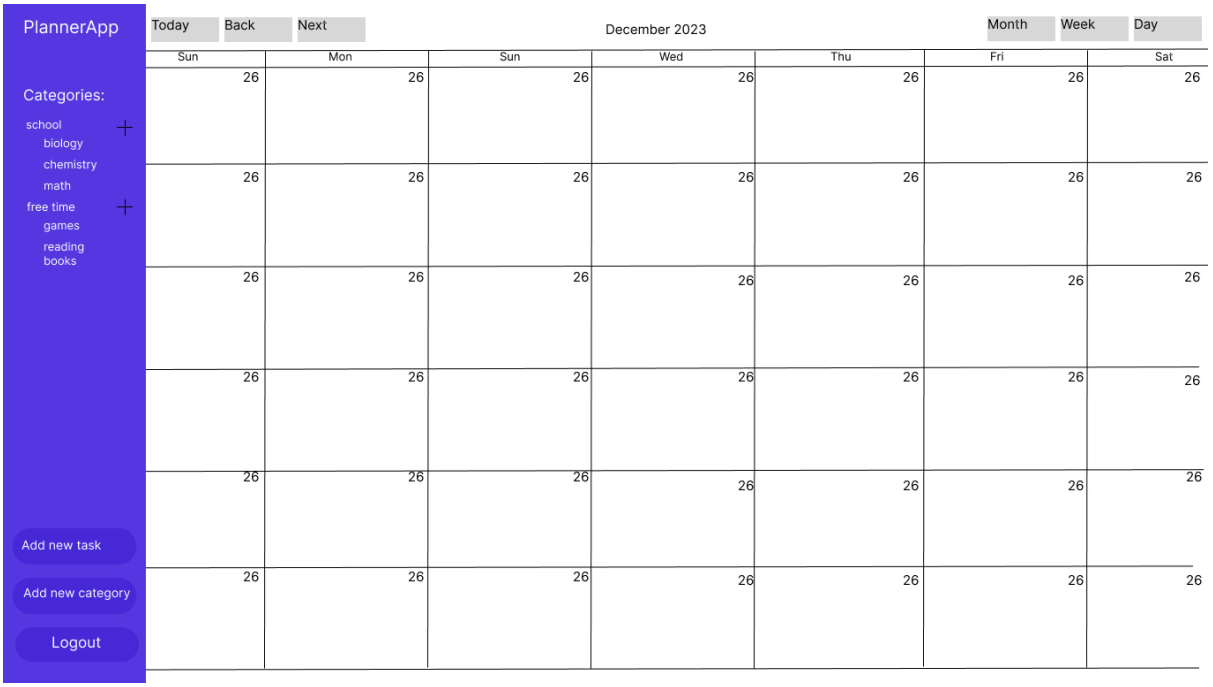
Rysunek 3.17, 3.18, 3.19, 3.20 przedstawiają projekty interfejsów użytkownika przygotowane w aplikacji Figma. Dzięki nim posiadamy wiedzę, jak powinien wyglądać finalny produkt. Kolorystyka została dobrana tak, aby korzystanie z aplikacji było wygodne zarówno w ciemnych jak i jasnych pomieszczeniach. Elementy zostały rozmieszczone tak, aby użytkownik z łatwością mógł poruszać się po aplikacji.



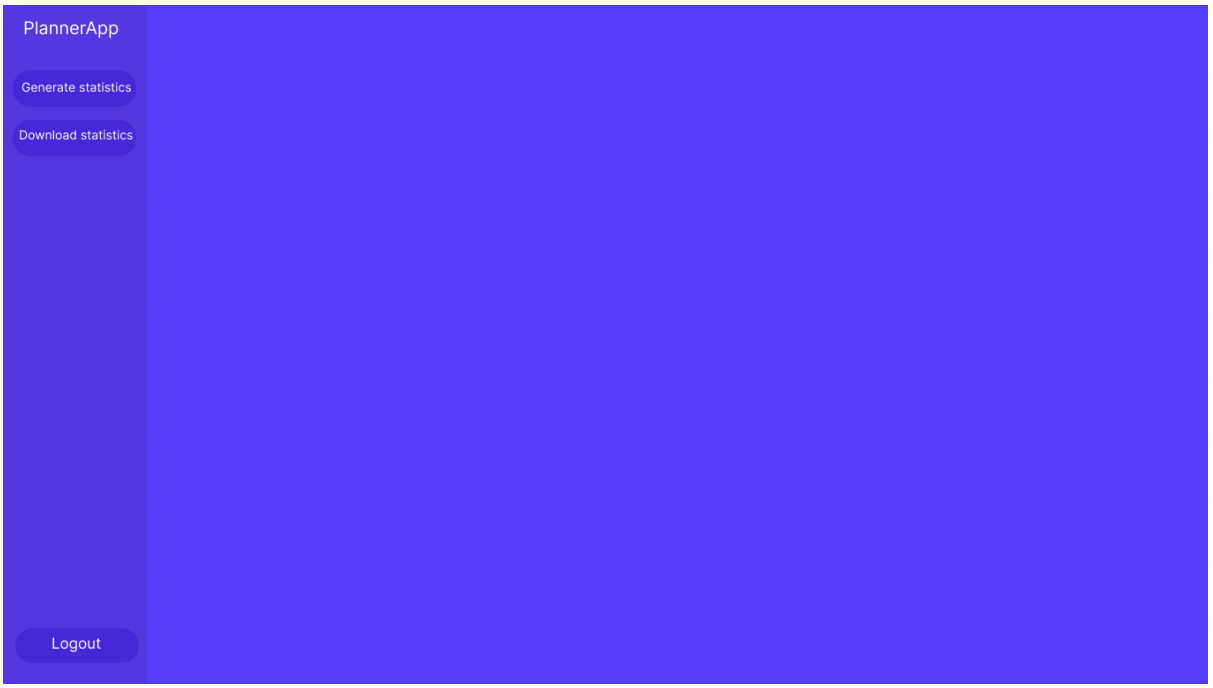
Rys. 3.17. Projekt Formularza logowania [opracowanie własne]



Rys. 3.18. Projekt formularza rejestracji [opracowanie własne]



Rys. 3.19. Projekt interfejsu użytkownika dla strony kalendarza[opracowanie własne]



Rys. 3.20. Prototyp interfejsu użytkownika dla strony kalendarza[opracowanie własne]

4. Opis technologii

4.1. Aplikacja Internetowa

Aplikacja internetowa jest to rodzaj programu komputerowego, który działa na zdalnym serwerze internetowym i umożliwia interakcję z użytkownikiem za pośrednictwem przeglądarki internetowej.

Jedną z kluczowych korzyści wynikających z zastosowania tego rozwiązania jest dynamiczna natura komunikacji z użytkownikiem. W przeciwieństwie do statycznych stron internetowych, aplikacje tego rodzaju są interaktywne, co pozwala na bardziej zaawansowane i elastyczne oddziaływanie z użytkownikiem, które jest dostosowane do ich potrzeb i zachowań.

Dodatkowo wielką zaletą jest elastyczność i wieloplatformowość. Dzięki temu rozwiązaniu, nie jesteśmy uzależnieni od konkretnego systemu operacyjnego, a możliwość uruchomienia aplikacji w dowolnej przeglądarce internetowej oznacza, że użytkownicy nie muszą martwić się o wystąpienie problemów ze zgodnością systemu, co ułatwia zarówno korzystanie, jak i aktualizacje oprogramowania.

Istotną kwestią jest również przechowywanie danych użytkowników na serwerach dostawcy. Dzięki temu zapewnione jest, że nawet w przypadku awarii urządzenia lub zmiany sprzętu, użytkownik nadal posiada dostęp do swoich danych, co sprawia że bezpieczeństwo i wygoda korzystania z aplikacji jest na najwyższym poziomie.

Warto zaznaczyć, że popularność aplikacji internetowych przekłada się na ich wszechstronność, ponieważ mogą być używane na różnych urządzeniach posiadających dostęp do sieci internetowej, takich jak smartfony, tablety i inne sprzęty elektroniczne, zwiększając dostępność i użyteczność dla użytkowników aplikacji [2].

4.2. Intellij IDEA

Główną motywacją do wyboru IntelliJ IDEA jako głównego środowiska pracy nad aplikacją była jej znakomita integracja z językiem Java, który był głównym językiem wykorzystywanym do napisania backendu, ponadto IntelliJ IDEA ułatwia wykorzystanie innych technologii, które również znalazły zastosowanie w projekcie. Kolejnym powodem był jego interfejs graficzny, po którym można poruszać się w łatwy i intuicyjny sposób. Na

dodatek środowisko pomaga w debugowaniu kodu poprzez sugerowanie wystąpienia błędów w danym miejscu, oprócz tego pokazuje ono często przyczyny wystąpienie takiego błędu, a także możliwości i propozycje rozwiązania go zgodne z ogólnymi praktykami.

Co więcej, w aplikacji wykorzystany został Spring Boot, którego zastosowanie stało się łatwiejsze dzięki temu środowisku. Spowodowane zostało to właśnie dzięki temu, że IntelliJ IDEA zapewnia szerokie wsparcie dla całego cyklu życia projektu opartego na Springu, zaczynając od generowania szablonu aplikacji, a kończąc na złożonych ustawieniach. Dzięki temu skonfigurowanie i rozwój aplikacji przebiegło w sprawny sposób.

Wybór IntelliJ IDEA był także motywowany szeroką gamą pluginów, które można w łatwy sposób wyszukać, pobrać i zainstalować. Dzięki elastyczności i możliwości uruchomienia wielu pluginów równocześnie można było przygotować miejsce pracy dostosowane do potrzeb programisty, co przełożyło się na zwiększenie wydajności w tworzeniu oprogramowania.

Kolejnym czynnikiem była integracja IntelliJ IDEA z systemem kontroli wersji Git. Środowisko daje nam możliwość kontrolowania zmian i monitorowanie postępu aplikacji poprzez wbudowaną funkcjonalność. Dzięki temu w łatwiejszy sposób można wykrywać błędy w kodzie oraz przemieszczać się między wersjami aplikacji.

Podsumowując, wybór IntelliJ IDEA jako głównego narzędzia do pracy nad aplikacją był spowodowany ogromną ilością benefitów, jakie jest w stanie zaoferować dla programisty. Jego intuicyjny interfejs, funkcjonalności oraz wsparcie dla najnowszych technologii, w znaczący sposób ułatwiły rozwój, testowanie i utrzymywanie wysokiej jakości kodu w projekcie. To narzędzie spełniło oczekiwania, a proces tworzenia oprogramowania stał się dzięki niemu przyjemny i efektywny.

4.3. Visual Studio Code

Do napisania części frontendowej wykorzystany został Visual Studio Code, który jest środowiskiem programistycznym zaprojektowanym przez firmę Microsoft. Narzędzie to okazało się niezwykle wszechstronne, doskonale sprawdzając się w procesie tworzenia frontendu aplikacji.

Visual Studio Code zostały wybrane ze względu na kilka czynników. Najważniejszym z nich była bezproblemowa integracja środowiska z technologiami, które były wykorzystywane. Dzięki temu nie tylko w frameworku React możliwe było

wykorzystywanie dedykowanych funkcji, lecz także korzystanie z narzędzi do debugowania, czy sugestii językowych do TypeScript.

Kolejnym z powodem było bogactwo darmowych rozszerzeń w zintegrowanej wyszukiwarce. Dzięki instalacji rozszerzeń można w łatwy sposób spersonalizować środowisko pracy do potrzeb projektu. Przekłada się to w znacznym stopniu na wygodę pracy, ale ma również wpływ na efektywność.

Korzystanie z Visual Studio Code umożliwia wygodne tworzenie oprogramowania. To środowisko daje możliwości przygotowania miejsca pracy pod dane potrzeby użytkownika. Dzięki jego możliwościom napisanie aplikacji przebiegło sprawnie, a także miało znaczny wpływ na tworzenie wyższej jakości kodu.

4.4. Git

Kolejnym niezwykle ważnym narzędziem był system kontroli wersji znany jako Git. Został wykorzystany jako kluczowe narzędzie do zarządzania kodem aplikacji. System ten pozwala na przechowywanie oraz śledzenie zmian w kodzie projektu poprzez połączenie lokalnego repozytorium, wraz ze zdalnymi repozytoriami, które mogą znajdować się na takich stronach jak GitHub czy BitBucket. Zapewniają one spójność i bezpieczeństwo kopii zapasowych aplikacji z różnych stadium jej rozwoju.

Korzystanie z Git'a może mieć istotny wpływ na proces tworzenia oprogramowania. Narzędzie to chociażby umożliwia zapisywanie kodu z danego etapu, dzięki czemu można w przystępny sposób analizować historię zmian. To z kolei pozwala na zdalne monitorowanie i debugowanie kodu, dzięki czemu jesteśmy w stanie zidentyfikować potencjalne błędy zanim trafią na kod produkcyjny.

Podsumowując, Git był narzędziem do tworzenia kopii kodu z kolejnych etapów jego produkcji, a także dawał możliwość zarządzania ewolucją projektu. Ponadto pozwolił na optymalizację i poprawę jakości kodu, dzięki szybkiemu reagowaniu na zmiany, czy błędy. To narzędzie zapewnia kontrolę nad procesem rozwoju projektu, co ma znaczenie, jeżeli chcemy tworzyć jakościowy kod.

4.5. Apache Maven

W aplikacji zostało wykorzystane narzędzie o nazwie Apache Maven, którego zadaniem jest zarządzanie zależnościami i artefaktami projektu. Najważniejszą cechą, która

przeważyła o wykorzystaniu tego narzędzia, jest jego zdolność do zarządzania cyklem życia projektu.

Kolejnym argumentem jest łatwość konfiguracji projektu, którą można wykonać za pomocą pliku “pom.xml”. W pliku “pom.xml” można konfigurować zależności, pluginy, konfiguracje kompilacji, testowanie oraz ustawienia związane z budową projektu. Umiejscowienie konfiguracji w jednym pliku znacząco usprawnia pracę nad projektem.

Dzięki Apache Maven udało się znacznie usprawnić procesy związane z budową, testowaniem i publikacją aplikacji. Narzędzie uprościło zarządzanie zewnętrznymi bibliotekami i zależnościami, minimalizując ryzyko konfliktów wynikających z różnych wersji oprogramowania. Dodatkowo Maven dostarcza narzędzia do raportowania, co umożliwia monitorowanie postępu projektu oraz szybkie reagowanie na ewentualne błędy, czy niezgodności w kodzie.

W rezultacie Apache Maven odegrał kluczową rolę w projekcie, zapewniając spójność, efektywność i kontrolę nad procesem rozwoju oprogramowania. Dzięki niemu można było skoncentrować się na tworzeniu wysokiej jakości aplikacji, mając pewność, że zarządzanie projektem od strony budowy i zależności jest rozwiązane [7].

4.6. Relacyjna baza danych PostgreSQL

Każda złożona aplikacja przechowuje dane, dlatego w projekcie wykorzystana została baza danych PostgreSQL. Została ona wybrana z kilku powodów.

Pierwszym z nich jest ogólne uznanie PostgreSQL jako jednego z najbardziej zaawansowanych systemów do zarządzania relacyjnymi bazami danych. Cieszy się on renomą w środowisku programistycznym, a reputację opiera na dwóch głównych cechach, jakimi są niezawodność i efektywność w przetwarzaniu danych.

Stabilność, która jest fundamentalna dla aplikacji internetowych, była głównym motywem wyboru PostgreSQL. Serwisy internetowe muszą być dostępne praktycznie przez cały czas, a niezawodność PostgreSQL gwarantuje właśnie taką ciągłość. Ta cecha eliminuje ryzyko wystąpienia awarii, co jest kluczowe w utrzymaniu spójności i nieprzerwanej pracy aplikacji.

Kolejnym, istotnym atutem PostgreSQL są jego możliwości przetwarzania dużych zbiorów danych. W przypadku aplikacji internetowych, które codziennie zbierają znaczną ilość informacji, ważne jest, aby baza danych była w stanie sprawnie je przetwarzać i

przechowywać. PostgreSQL doskonale sprostał tym wymaganiom, umożliwiając płynne zarządzanie i operacje na dużych zbiorach danych, co było kluczowe dla aplikacji.

Ogólnie rzecz biorąc, zastosowanie PostgreSQL w aplikacji zapewniło solidne, niezawodne i efektywne fundamenty, umożliwiając aplikacji nie tylko pracę w sposób ciągły, ale także skalowalność wraz z rosnącymi potrzebami zbierania i przetwarzania danych [1].

4.7. JPA Java Persistence API

W projekcie został wykorzystany mechanizm Java Persistence API (JPA), który pozwala na zarządzanie danymi w bazie. Dzięki niemu nie trzeba pisać zapytań SQL, ponieważ mechanizm ten jest warstwą abstrakcji, która umożliwia podstawowe operacje CRUD na danych przez wykorzystanie obiektów Java. Dzięki adnotacjom z JPA, można precyzyjnie zdefiniować relacje między różnymi encjami, co pozwala skoncentrować się na strukturze obiektów oraz ich wzajemnych powiązaniach, zamiast zagłębiać się w szczegóły techniczne samej bazy danych.

To podejście pozwala aplikacji być bardziej elastyczną i przenośną. Poprzez wykorzystanie tych samych mapowań i encji, można pracować niezależnie od konkretnej bazy danych. Oznacza to, że można łatwiej zmienić bazę danych, ponieważ struktura danych nie jest ściśle powiązana z konkretnym systemem bazodanowym. W przyszłości daje to możliwość łatwiejszej zmiany bazy danych, zgodną z potrzebami aplikacji.

Wykorzystanie Java Persistence API (JPA) nie tylko pozwala pracować wygodnie, ale także poprawia jakość pracy na danych. Skupienie się nad obiektami w języku Java znacznie ułatwiło operacje na danych, redukując ryzyko błędów oraz pisanie i utrzymanie kodu. To również zwiększyło elastyczność projektu, umożliwiając szybsze dostosowanie się do zmian w wymaganiach biznesowych, czy ewentualnej migracji na inną bazę danych.

W rezultacie, wykorzystanie Java Persistence API (JPA) przyczyniło się do efektywniejszego zarządzania danymi oraz wpłynęło na rozwój aplikacji, zapewniając większą przejrzystość, elastyczność i skalowalność projektu [9].

4.8. Spring boot framework

Fundamentem dla architektury projektu jest framework Spring Boot. Narzędzie to pozwala skupić się na wytwarzaniu kodu oraz rozwoju aplikacji, ponieważ eliminuje skomplikowane procesy konfiguracyjne.

Framework ten dostarcza gotowych modułów, co w znaczący sposób przyspiesza programowanie. W aplikacji zostały wykorzystanie gotowe rozwiązania do obsługi bezpieczeństwa, czy bazy danych. Pozwoliło to na zaoszczędzenie czasu, gdyż Spring Boot zapewnia te rozwiązania i nie trzeba tworzyć ich od zera.

Wykorzystanie Spring Boot zapewniło aplikacji także elastyczność i skalowalność, umożliwiając dokładne dostosowanie aplikacji pod potrzeby biznesowe. Również zdolność tego frameworku do łatwej integracji z innymi narzędziami, sprawiła że postawienie na Spring Boot było doskonałym wyborem.

Podsumowując, dzięki Spring Boot aplikacja mogła być szybko rozwijana, opierając się na solidnych fundamentach frameworka. To pozwoliło na skoncentrowanie się na kluczowych aspektach biznesowych, przez ułatwioną konfigurację. Bogata funkcjonalność sprawiła, że Spring Boot okazał się jednym z najwartościowszych elementów w procesie tworzenia projektu [5][6].

4.9. React

Do utworzenia frontendu aplikacji został wykorzystany React, który jest biblioteką JavaScript utworzoną przez firmę Facebook, która obecnie znana jest jako Meta. React pozwala skupić się na rozwoju frontendu, dzięki gotowym rozwiązaniom oraz eliminacji skomplikowanych procesów.

Jedną z głównych cech Reacta jest jego deklaratywność, dzięki niej nie trzeba ręcznie manipulować obiektem DOM, wystarczy jedynie opisać jak UI powinno reagować na zmiany, a React powinien automatycznie zaktualizować wygląd interfejsu aplikacji. Jest to cecha, która pozwala pozbyć się większości błędów wynikających z ręcznej manipulacji DOM'em, a także ułatwia utrzymywanie aplikacji i zwiększa czytelność kodu.

Kolejnym udogodnieniem jakie dostarcza React jest koncepcja komponentów, na które można patrzeć jak na klocki, z których tworzymy interfejs użytkownika. Dzięki nim można w łatwy sposób tworzyć i ponownie używać części już istniejącego interfejsu.

React również wpływa na wrażenia użytkownika, poprzez wspieranie tworzenia aplikacji jednostronnych, co z kolei pozwala na płynne i dynamiczne przejścia między widokami bez konieczności odświeżania strony.

React jest młodym frameworkiem, ale już cieszy się popularnością na całym świecie, co przekłada się na ciągły jego rozwój przez twórców.

Całościowo, wykorzystanie Reacta pozwala efektywnie budować nowoczesne interfejsy użytkownika, a pisanie kodu w modularny sposób zwiększa jego reużywalność i czytelność [8].

4.10. JavaScript

Kolejnym narzędziem wykorzystanym do napisania frontendu aplikacji był JavaScript. Jest on jednym z najczęściej wybieranych języków do tworzenia frontendu aplikacji internetowych.

JavaScript jest dynamicznym językiem, dzięki czemu w łatwy sposób możemy zarządzać obiektami. Jest on interpretowany co sprawia, że strony szybciej się ładują, gdyż kod jest wykonywany po stronie klienta.

Ważną jego zaletą jest jego bezproblemowa integracja z React, co pozwala na wykorzystanie obu technologii jednocześnie, a dzięki Node Package Manager w łatwy sposób można zarządzać zależnościami w projekcie.

4.11. Java

Java jest językiem programowania, na którym opiera się cały backend aplikacji. Pierwsza wersja tego języka została opracowana w 1995 roku, w aplikacji natomiast wykorzystywana jest Java w wersji 17.

Głównym powodem wyboru tego języka jest jego interpretowalność poprzez użycie JVM, czyli wirtualnej maszyny Java, która to pozwala uruchamiać oprogramowanie na każdym systemie, na którym ją zainstalujemy. Aplikacje napisane w języku Java mogą być nieznacznie wolniejsza w porównaniu do C czy C++, gdyż kod bajtowy musi zostać przetworzony przez JVM.

Kolejną równie ważną zaletą jest obiektowość, dzięki niej jesteśmy w stanie kreować kod, który można używać wielokrotnie. Inne zalety tego języka to mechanizmy, które można w nim wykorzystać takie jak mechanizm hermetyzacji. Polega on na świadomym ukrywaniu lub też udostępnianiu atrybutów, pól w obiektach czy metodach danej klasy, a to z kolei przekłada się na bardziej świadome i bezpieczne wykorzystanie ich w innych częściach kodu. Drugim z mechanizmów, który znacznie ułatwia pisanie przejrzystego kodu to abstrakcja. Pomaga ona nam ukryć mniej ważne informacje w projekcie poprzez przykrycie ich warstwą abstrakcji, która skupia się na ważniejszych elementach. Kolejnymi paradygmatami są

dziedziczenie i polimorfizm. Dziedziczenie pozwala na tworzenie klas zawierających chronione zmienne i funkcje klasy, po której dziedziczy, to ułatwia pisanie kodu przy większej ilości klas.

Java jest również bardzo bezpiecznym językiem, dzięki silnemu typowaniu, które nie pozwala przypisywać zmiennej jednego typu innych typów. Może to uchronić nasz program przed atakami związanymi z manipulacją pamięcią. Inną funkcjonalność związaną z pamięcią, a raczej jej czyszczeniem jest Garbage Collector, który to śledzi i usuwa z pamięci niewykorzystywane obiekty.

Nie można też zapomnieć o wielkiej społeczności jaka stoi za tym językiem, statystyki pokazują że około trzydzieści procent programistów na całym świecie korzysta z tego języka [10]. Jest to wielki plus, gdyż język ten jest stale wspierany, a wiele problemów jest poruszanych na forach internetowych takich jak Stack Overflow, dzięki czemu można znaleźć rozwiązanie naszego problemu.

Podsumowując, Java została wybrana ze względu na bezpieczeństwo, mechanizmy oraz jej ogromną społeczność, która udziela się w doskonaleniu tego języka [3].

5. Wybrane rozwiązania implementacyjne

W niniejszym rozdziale zostały przedstawione wybrane fragmenty kodu źródłowego aplikacji wraz z opisami.

W aplikacji znajdują się cztery kontrolery które odpowiedzialne są za komunikację HTTP. Na rysunku 5.1 znajduję się klasa kontrolera “TaskController”, wraz z trzema wybranymi metodami, które posiada. Pierwsza metoda “addTask” jest odpowiedzialna za dodawanie nowych zadań, druga to “saveAssignedTask”, jest jedną z najważniejszych metod, gdyż odpowiada za przypisywanie zadań do dnia. Ostatnią z nich jest “editAssignedTask” i odpowiada za edytowania przypisanych wcześniej zadań.

```java
@RestController
@RequestMapping(path = "/api")
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)

public class TaskController {

    8 usages
    TaskProviderFacade taskProviderFacade;

    no usages  Bartek Kotapka
    @Autowired
    public TaskController(TaskProviderFacade taskProviderFacade) {
        this.taskProviderFacade = taskProviderFacade;
    }

    no usages  Bartek Kotapka
    @PostMapping(value = "/addTask", produces = "application/json")
    @Operation(summary = "add new task")
    public ResponseEntity<TaskDto> addTask(@RequestBody TaskDto taskDto) {
        TaskDto task = taskProviderFacade.saveTask(taskDto);
        return ResponseEntity.ok(task);
    }

    no usages  new *
    @PostMapping(value = "/saveAssignedTask", consumes = "application/json", produces = "application/json")
    @Operation(summary = "save assigned task")
    public ResponseEntity<AssignedTaskDto> saveAssignedTask(@RequestBody AssignedTaskDto task) {
        AssignedTaskDto assignedTask = taskProviderFacade.saveAssignedTask(task);
        return ResponseEntity.ok(assignedTask);
    }

    no usages  new *
    @PostMapping(value = "/editAssignedTask", consumes = "application/json", produces = "application/json")
    @Operation(summary = "delete assigned tasks")
    public ResponseEntity<EditedTaskDto> editAssignedTask(@RequestBody EditedTaskDto assignedTaskDto) {
        EditedTaskDto assignedTask = taskProviderFacade.editAssignedTask(assignedTaskDto);
        return ResponseEntity.ok(assignedTask);
    }
```

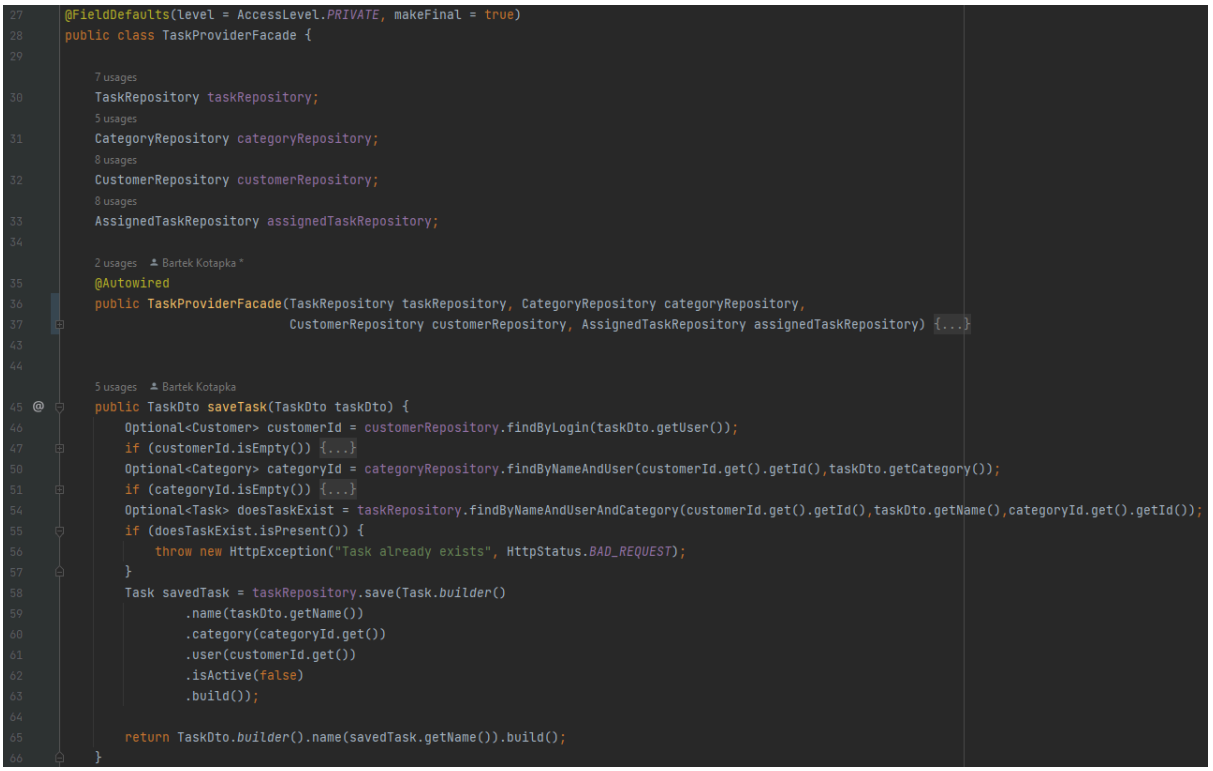
Rys. 5.1. Wygląd kontrolera zadań[opracowanie własne]

Aby zachować bezpieczeństwo danych użytkowników aplikacji, zastosowany został algorytm MD5 przekształcający hasło użytkownika na zaszyfrowany ciąg znakowy.
Rysunek 5.2 przedstawia funkcję która przyjmuje ciąg znaków i przy pomocy algorytmu przekształca hasło, dzięki czemu do bazy danych trafia już zaszyfrowane.

```java
public class MD5 {
    2 usages  Bartek Kotapka
    public static String getMd5(String input)
    {
        try {
            MessageDigest md = MessageDigest.getInstance( algorithm: "MD5");
            byte[] messageDigest = md.digest(input.getBytes());
            BigInteger no = new BigInteger( signum: 1, messageDigest);
            String hashtext = no.toString( radix: 16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        }
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
}
```

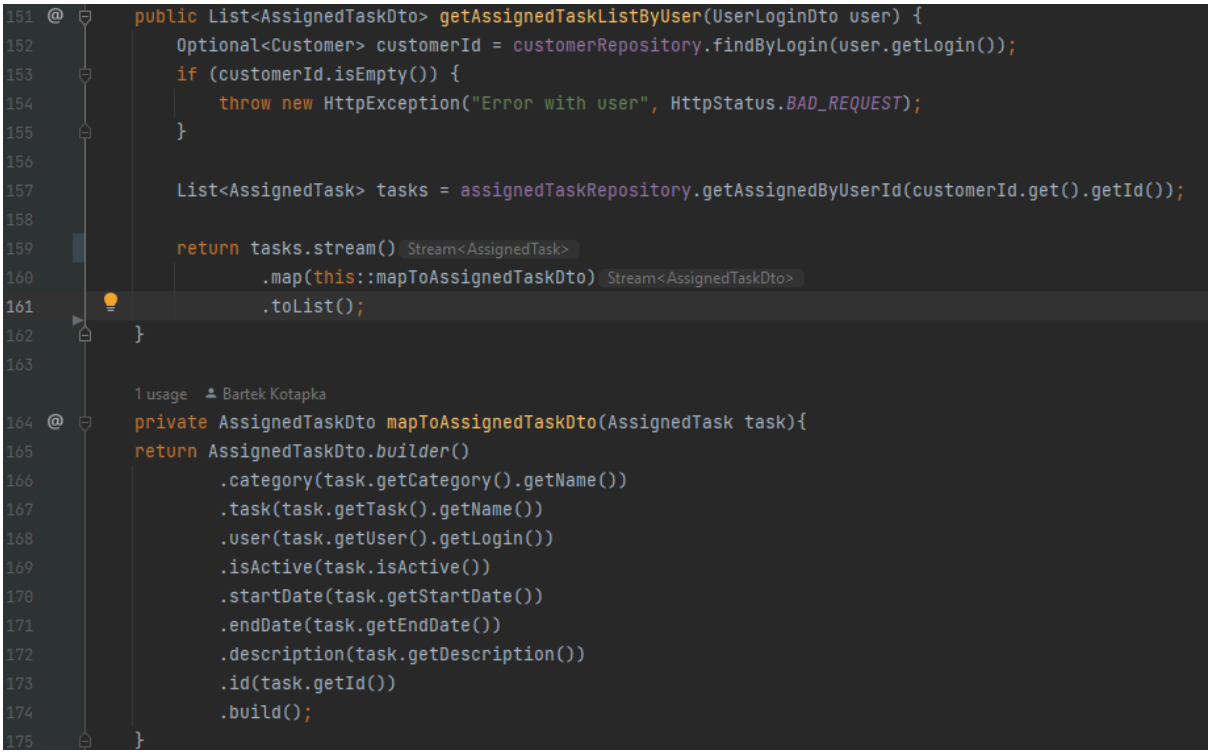
Rys. 5.2. Klasa z mechanizmem szyfrującym[opracowanie własne]

Aby ułatwić testowanie oraz żeby aplikacja była przejrzysta i łatwa w rozwoju wykorzystany został wzorzec fasady. Jedno z przykładowych implementacji fasady zostało zaprezentowane na rysunkach 5.3 oraz 5.4, w fasadzie użyte są dwa repozytoria. W tej klasie znajduje się prosty interfejs, który umożliwia korzystanie z zaimplementowanych metod.
Na rysunku 5.3 widoczna jest metoda “saveTask”, która odpowiada za zapisywanie zadań. Aby takie zadanie zostało zapisane muszą zostać spełnione warunki znajdujące się w metodzie. Zapewniają, że użytkownik, do którego przypisane jest zadanie, istnieje oraz sprawdzają, czy nie istnieje już zadanie o tej samej nazwie. Rysunek 5.4 przedstawia metodę, która przygotowuje listę przypisanych zadań dla danego użytkownika. Dzięki temu widoczne one są zaraz po załadowaniu planera.



```java
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
public class TaskProviderFacade {

    7 usages
    TaskRepository taskRepository;
    5 usages
    CategoryRepository categoryRepository;
    8 usages
    CustomerRepository customerRepository;
    8 usages
    AssignedTaskRepository assignedTaskRepository;

    2 usages  Bartek Kotapka *
    @Autowired
    public TaskProviderFacade(TaskRepository taskRepository, CategoryRepository categoryRepository,
                              CustomerRepository customerRepository, AssignedTaskRepository assignedTaskRepository) {...}


    5 usages  Bartek Kotapka
    public TaskDto saveTask(TaskDto taskDto) {
        Optional<Customer> customerId = customerRepository.findByLogin(taskDto.getUser());
        if (customerId.isEmpty()) {...}
        Optional<Category> categoryId = categoryRepository.findByNameAndUser(customerId.get().getId(),taskDto.getCategory());
        if (categoryId.isEmpty()) {...}
        Optional<Task> doesTaskExist = taskRepository.findByNameAndUserAndCategory(customerId.get().getId(),taskDto.getName(),categoryId.get().getId());
        if (doesTaskExist.isPresent()) {
            throw new HttpException("Task already exists", HttpStatus.BAD_REQUEST);
        }
        Task savedTask = taskRepository.save(Task.builder()
                .name(taskDto.getName())
                .category(categoryId.get())
                .user(customerId.get())
                .isActive(false)
                .build());

        return TaskDto.builder().name(savedTask.getName()).build();
    }
```

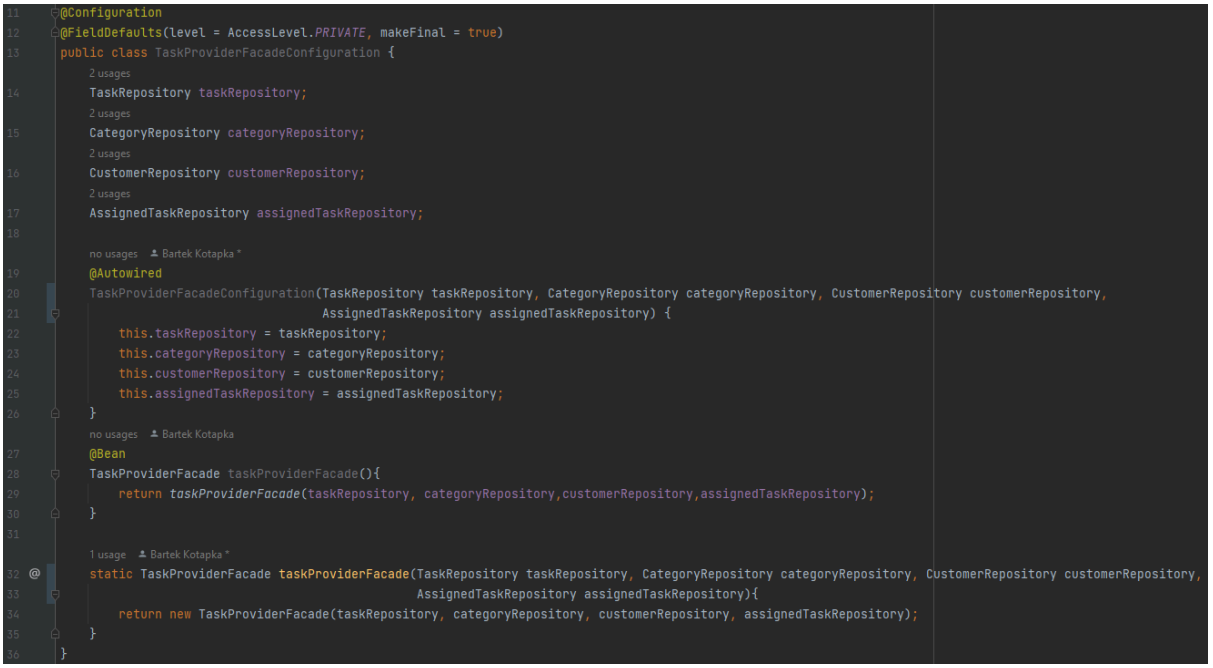
Rys. 5.3. Klasa fasady dla zadań(część 1)[opracowanie własne]



```java
    public List<AssignedTaskDto> getAssignedTaskListByUser(UserLoginDto user) {
        Optional<Customer> customerId = customerRepository.findByLogin(user.getLogin());
        if (customerId.isEmpty()) {
            throw new HttpException("Error with user", HttpStatus.BAD_REQUEST);
        }

        List<AssignedTask> tasks = assignedTaskRepository.getAssignedByUserId(customerId.get().getId());

        return tasks.stream()  Stream<AssignedTask>
                .map(this::mapToAssignedTaskDto)  Stream<AssignedTaskDto>
                .toList();
    }

    1 usage  Bartek Kotapka
    private AssignedTaskDto mapToAssignedTaskDto(AssignedTask task){
    return AssignedTaskDto.builder()
            .category(task.getCategory().getName())
            .task(task.getTask().getName())
            .user(task.getUser().getLogin())
            .isActive(task.isActive())
            .startDate(task.getStartDate())
            .endDate(task.getEndDate())
            .description(task.getDescription())
            .id(task.getId())
            .build();
    }
```

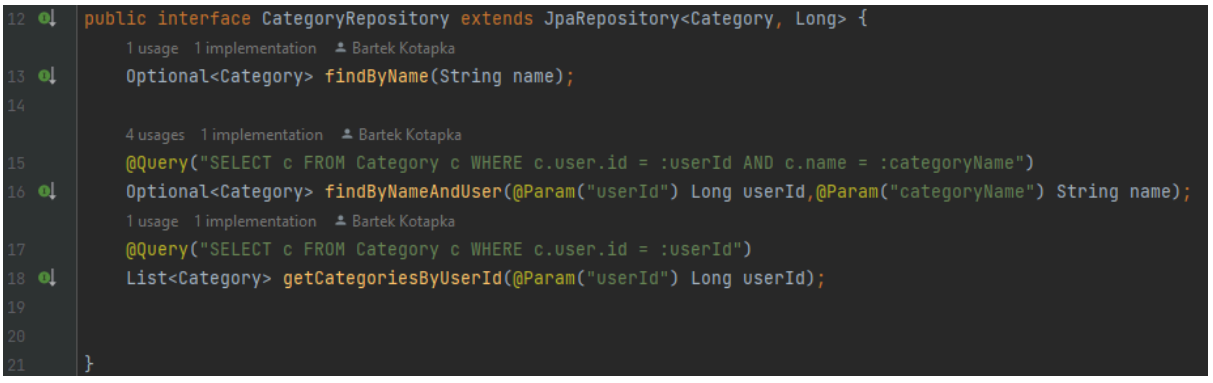
Rys. 5.4. Klasa fasady dla zadań(część 2)[opracowanie własne]

Zaimplementowane zostały również klasy konfiguracyjne do fasad (rysunek 5.5). Dzięki tej klasie można oddzielić konfiguracje od fasady, co sprawia że konfiguracja systemu staje się bardziej elastyczna.



Rys. 5.5. Klasa konfiguracyjna fasady dla zadań[opracowanie własne]

Do przeprowadzania operacji na danych został wykorzystany mechanizm JPA. Umożliwia on podstawowe operacje CRUD na bazie danych (rysunek 5.6). Pierwsza operacja “findByNameAndUser” pozwala pobrać z bazy danych kategorię dla użytkownika o konkretnej nazwie kategorii. Drugie zapytanie “getCategoriesByUserId” zwraca listę kategorii dla użytkownika.



Rys. 5.6. Interfejs JPA dla kategorii[opracowanie własne]

Klasa która znajduje się na rysunku 5.7 przedstawia tabelę, która odwzorowuje encję w bazie danych.

```java
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Data
@Entity(name = "Category")
@Table
public class Category {
    1 usage
    @Id
    @SequenceGenerator(
            name = "category_sequence",
            sequenceName = "category_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "category_sequence"
    )
    private Long id;
    3 usages
    private String name;
    3 usages
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private Customer user;
}
```

Rys. 5.7.Klasa reprezentująca encje kategorii[opracowanie własne]

6. Testy

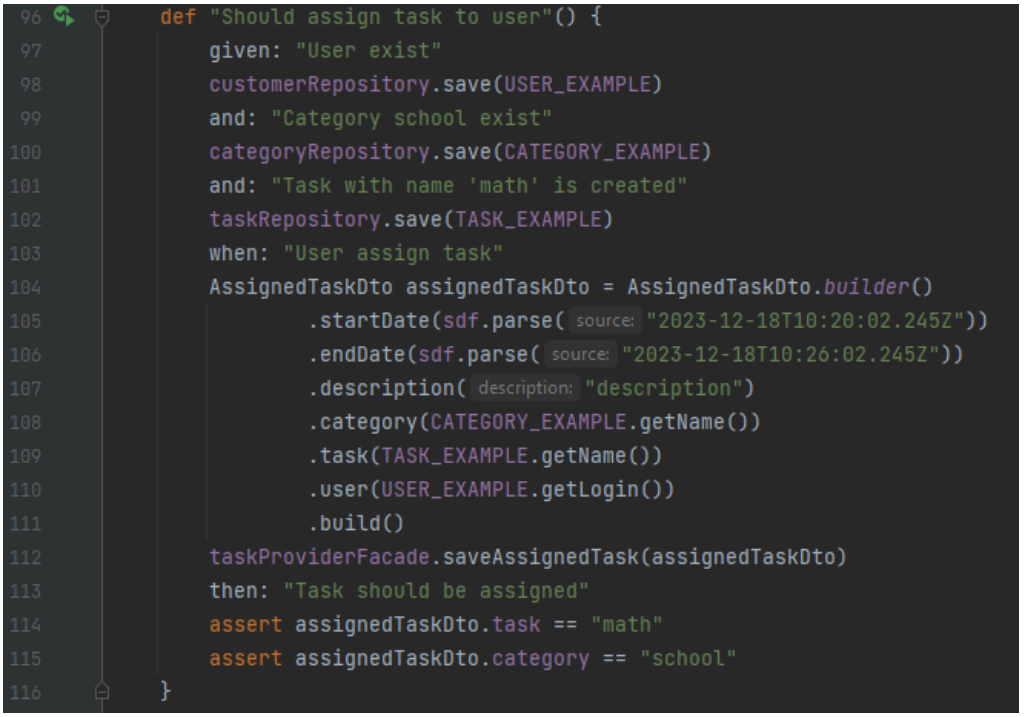
Aby zadbać o poprawne działanie aplikacji trzeba przeprowadzić testy. W projekcie zostały wykonane dwa rodzaje testów, pierwszy z nich to testy jednostkowe, a drugie to manualne. Manualne testy były prowadzone przez cały etap produkcji oprogramowania, a także na zakończeniu.

Na rysunkach 6.1, 6.2 oraz 6.3 znajdują się przykładowe testy jednostkowe z aplikacji. Do napisania ich użyty został framework spock z wykorzystaniem języka groovy.

Na rysunku 6.1 znajdują się testy sprawdzające, czy zadanie zostanie utworzone, jeżeli kategoria zostanie wybrana lub nie. Testy jednostkowe na rysunku 6.2 oraz 6.3 sprawdzają jedną z głównych funkcji aplikacji, czyli poprawne przypisywanie zadań do dnia.

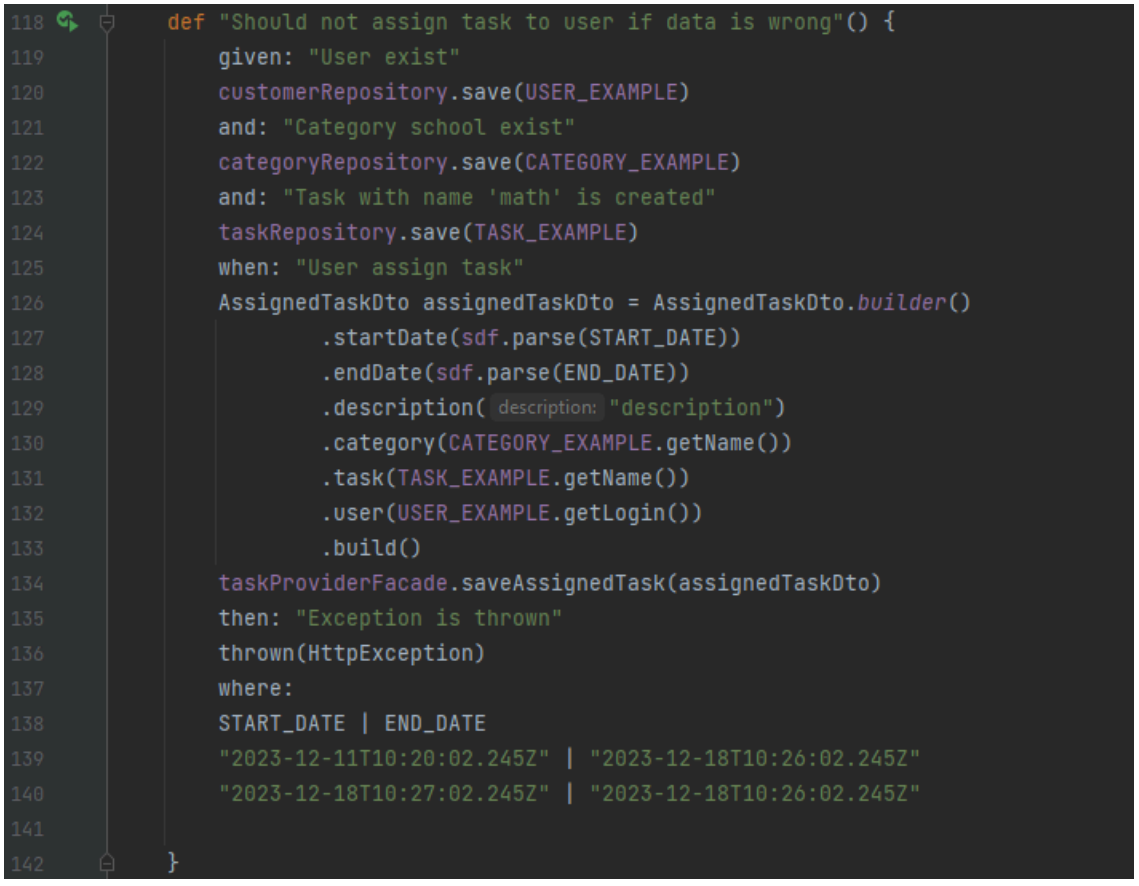
```
58    def "Should create task with category"() {
59        given: "User exist"
60        customerRepository.save(USER_EXAMPLE)
61        and: "Category school exist"
62        categoryOperationFacade.saveCategory(CATEGORY_DTO_EXAMPLE)
63        when: "Task with name 'math' is created"
64        TaskDto math = TaskDto.builder().user( user: "JAN").category( category: "school").name( name: "math").build()
65        taskProviderFacade.saveTask(math)
66        then: "Task is saved"
67        assert math.name == "math"
68        assert math.category == "school"
69    }
70
71    def "Should not create task without category"() {
72        given: "User exist"
73        customerRepository.save(USER_EXAMPLE)
74        and: "Category school exist"
75        categoryOperationFacade.saveCategory(CATEGORY_DTO_EXAMPLE)
76        when: "Task with name 'math' is created"
77        TaskDto math = TaskDto.builder().user( user: "JAN").category( category: "").name( name: "math").build()
78        taskProviderFacade.saveTask(math)
79        then: "Exception is thrown"
80        thrown(HttpException)
81    }
```

Rys. 6.1. Testy jednostkowe dla tworzenia zadań[opracowanie własne]



```
96      def "Should assign task to user"() {
97          given: "User exist"
98          customerRepository.save(USER_EXAMPLE)
99          and: "Category school exist"
100         categoryRepository.save(CATEGORY_EXAMPLE)
101         and: "Task with name 'math' is created"
102         taskRepository.save(TASK_EXAMPLE)
103         when: "User assign task"
104         AssignedTaskDto assignedTaskDto = AssignedTaskDto.builder()
105                 .startDate(sdf.parse( source: "2023-12-18T10:20:02.245Z"))
106                 .endDate(sdf.parse( source: "2023-12-18T10:26:02.245Z"))
107                 .description( description: "description")
108                 .category(CATEGORY_EXAMPLE.getName())
109                 .task(TASK_EXAMPLE.getName())
110                 .user(USER_EXAMPLE.getLogin())
111                 .build()
112         taskProviderFacade.saveAssignedTask(assignedTaskDto)
113         then: "Task should be assigned"
114         assert assignedTaskDto.task == "math"
115         assert assignedTaskDto.category == "school"
116     }
```

Rys. 6.2. Testy jednostkowe dla przypisywania zadań(część 1)[opracowanie własne]

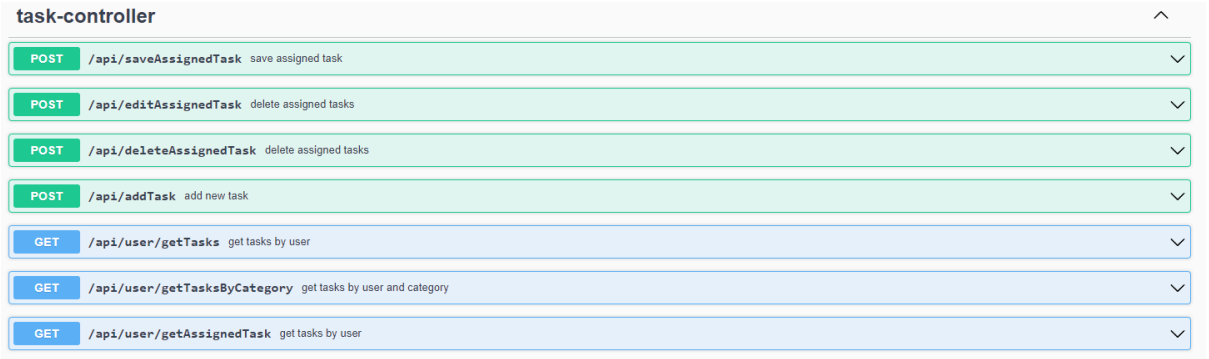


```
118     def "Should not assign task to user if data is wrong"() {
119         given: "User exist"
120         customerRepository.save(USER_EXAMPLE)
121         and: "Category school exist"
122         categoryRepository.save(CATEGORY_EXAMPLE)
123         and: "Task with name 'math' is created"
124         taskRepository.save(TASK_EXAMPLE)
125         when: "User assign task"
126         AssignedTaskDto assignedTaskDto = AssignedTaskDto.builder()
127                 .startDate(sdf.parse(START_DATE))
128                 .endDate(sdf.parse(END_DATE))
129                 .description( description: "description")
130                 .category(CATEGORY_EXAMPLE.getName())
131                 .task(TASK_EXAMPLE.getName())
132                 .user(USER_EXAMPLE.getLogin())
133                 .build()
134         taskProviderFacade.saveAssignedTask(assignedTaskDto)
135         then: "Exception is thrown"
136         thrown(HttpException)
137         where:
138         START_DATE | END_DATE
139         "2023-12-11T10:20:02.245Z" | "2023-12-18T10:26:02.245Z"
140         "2023-12-18T10:27:02.245Z" | "2023-12-18T10:26:02.245Z"
141
142     }
```

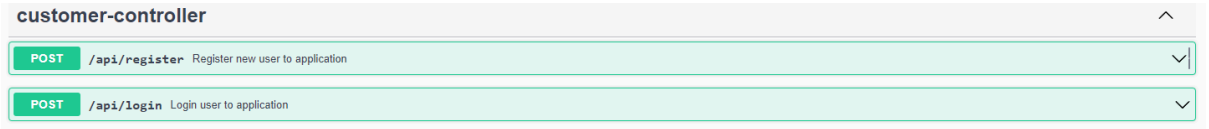
Rys. 6.3. Testy jednostkowe dla przypisywania zadań(część 2)[opracowanie własne]

7. Dokumentacja API

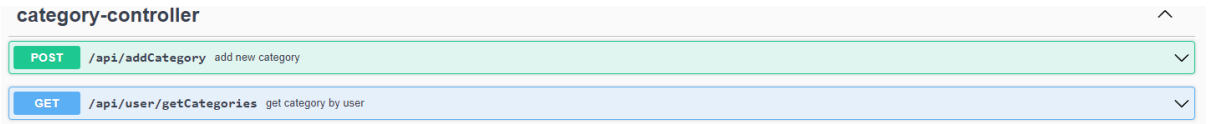
Przed utworzeniem interfejsu użytkownika, aby móc testować serwer aplikacji wykorzystane zostało narzędzie Swagger, które jest dokumentacją przedstawiającą endpointy i umożliwiającą operowanie na nich. Na rysunkach 7.1, 7.2, 7.3, 7.4 przedstawione są endpointy z kontrolerów znajdujących się na serwerze. Endpointy zostały pogrupowane automatycznie przez narzędzie Swagger według ich miejsca znajdowania się w poszczególnych kontrolerach na serwerze.



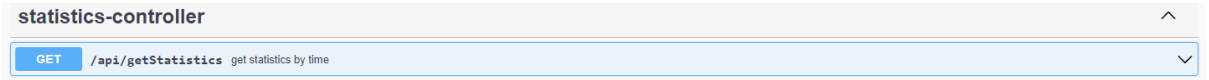
Rys. 7.1 Kontroler zadań [opracowanie własne]



Rys. 7.2 Kontroler użytkownika [opracowanie własne]



Rys. 7.3 Kontroler kategorii [opracowanie własne]

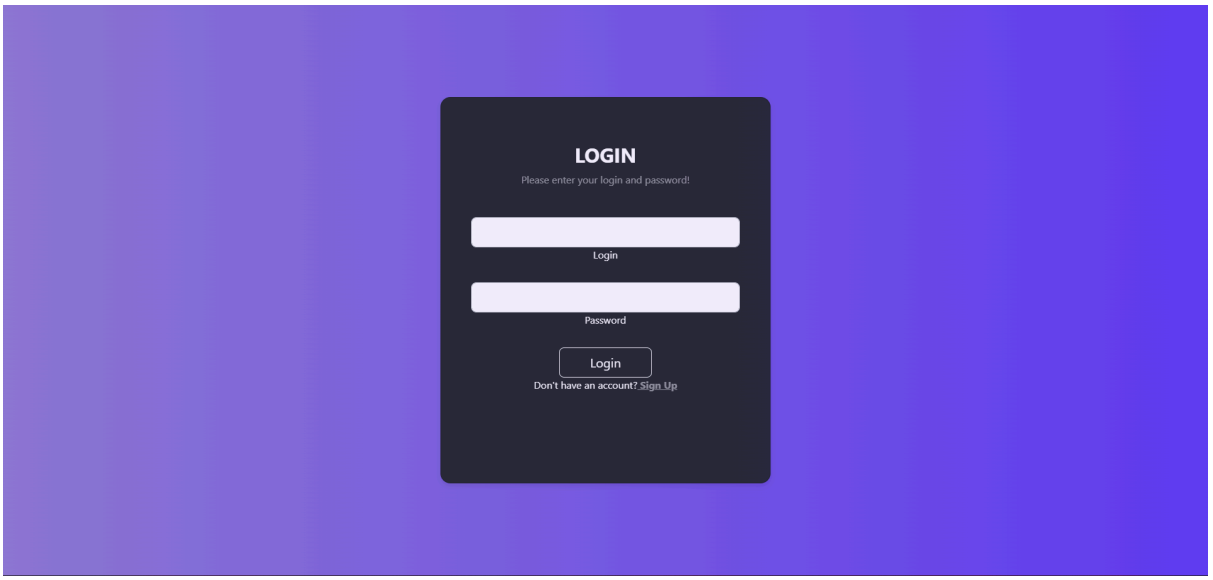


Rys. 7.4 Kontroler statystyk[opracowanie własne]

8. Interfejs użytkownika

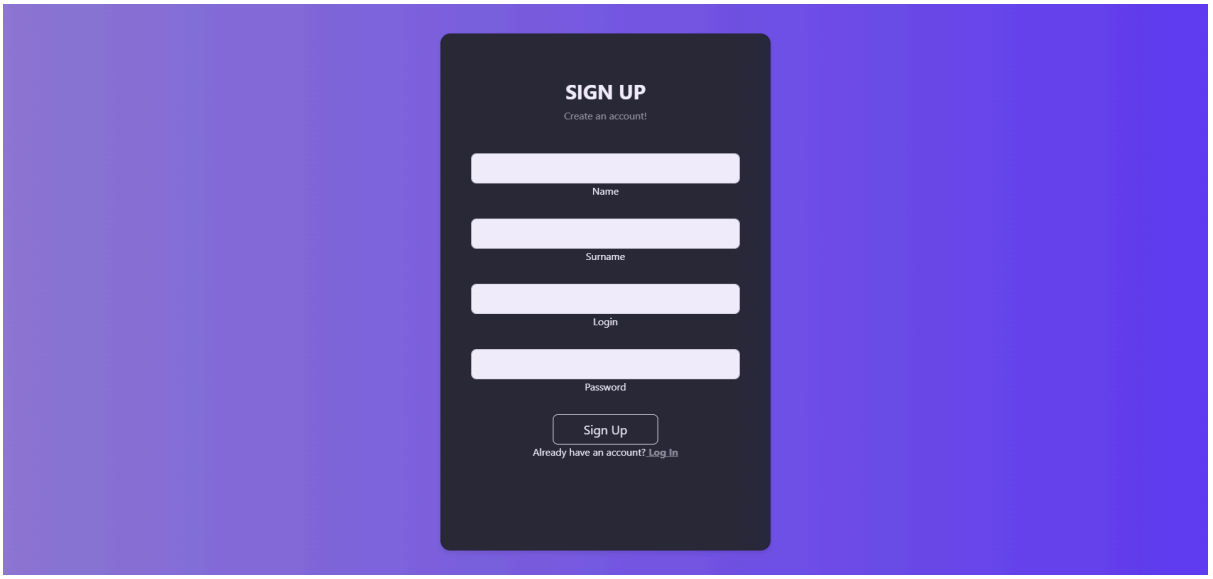
W niniejszym rozdziale zostanie przedstawiony finalny interfejs użytkownika, który znajduję się w aplikacji.

Na rysunku 8.1 przedstawiony jest wygląd panelu, który widzimy po przejściu na stronę aplikacji. Osoba posiadająca konto może się zalogować wpisując poprawne dane. Po pomyślnym zalogowaniu zostanie przeniesiona do głównej strony aplikacji.



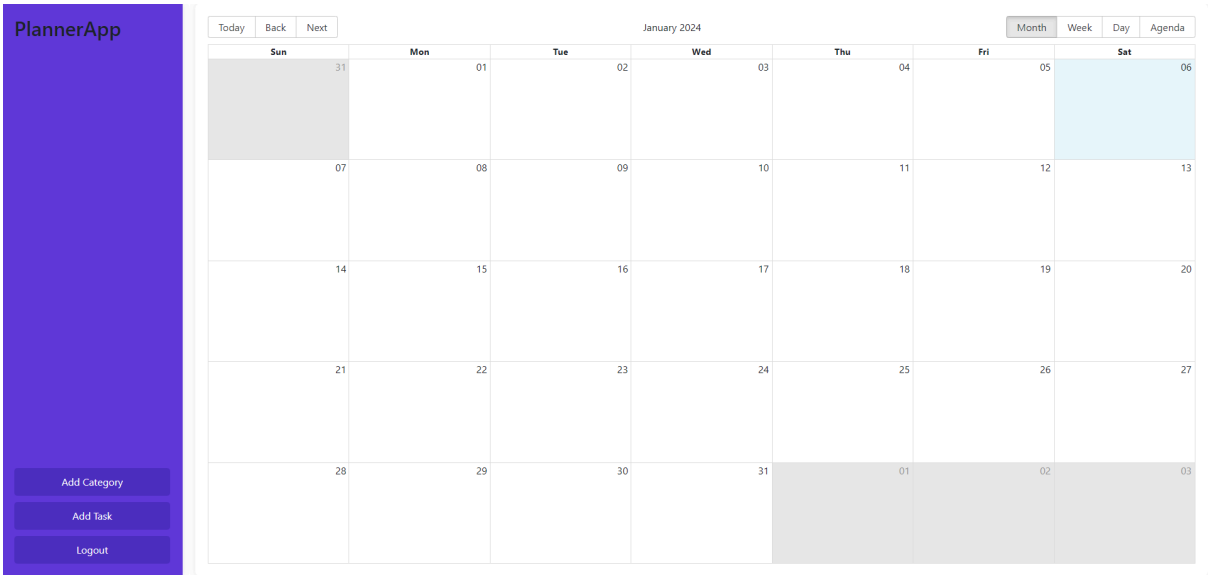
Rys. 8.1. Gotowy interfejsu użytkownika dla panelu logowania[opracowanie własne]

Dla osób nieposiadających jeszcze konta, można zarejestrować nowe, przechodząc do odpowiedniego formularza. Wygląd tego formularza znajduje się na rysunku 8.2. Użytkownik może utworzyć konto po uzupełnieniu danych. Jeżeli użytkownik będzie chciał zarejestrować nowe konto z istniejącym już loginem, nie uda mu się to, gdyż login jest identyfikatorem użytkownika i musi być unikalny. Jeżeli rejestracja przebiegnie pomyślnie to użytkownik zostanie zalogowany i przeniesiony do głównej strony aplikacji.



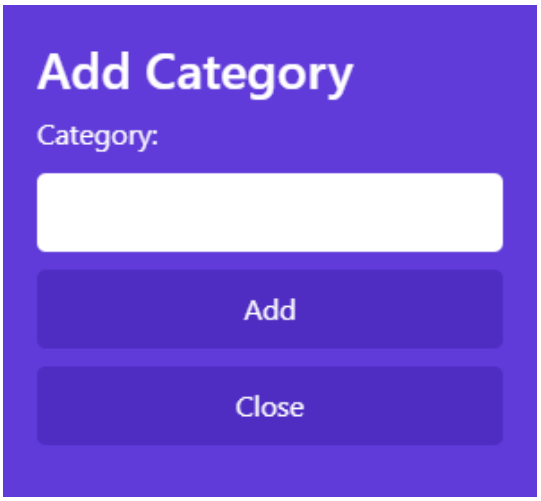
Rys. 8.2. Gotowy interfejsu użytkownika dla rejestracji[opracowanie własne]

Główna strona aplikacji to kalendarz z panelem bocznym. Na panelu bocznym znajdują się przyciski do dodawania kategorii i zadań oraz do poruszania się po aplikacji. Użytkownik ma do dyspozycji kalendarz, po lewej górnej stronie można poruszać się w przód i tył w kalendarzu. W prawym górnym rogu znajdują się przyciski do zmiany wyglądu kalendarza. Dostępne opcje to widok miesiąca, tygodnia, dnia oraz zadań.



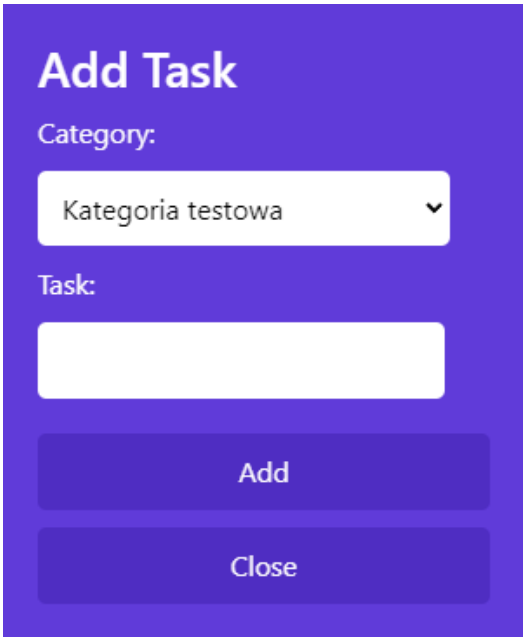
Rys. 8.3. Gotowy interfejsu użytkownika dla rejestracji[opracowanie własne]

Rysunek 8.4 przedstawia okienko popup, które pojawia się po przyciśnięciu przycisku dodania kategorii w panelu bocznym. Kategorie muszą mieć unikatowe nazwy dla użytkownika.



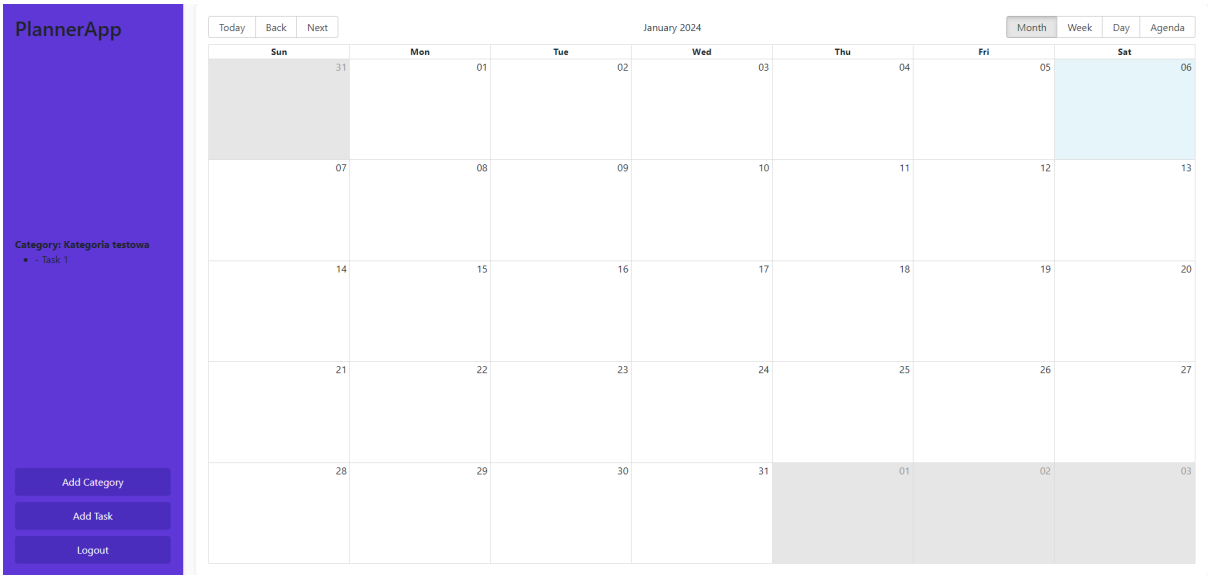
Rys. 8.4. Gotowy interfejsu użytkownika dodawania kategorii[opracowanie własne]

Rysunek 8.5 przedstawia okienko popup dodawania zadań. Aby dodać zadanie musi być utworzona co najmniej jedna kategoria, którą wybierzemy z pola wyboru. W jednej kategorii nie można dodać dwóch zadań o takiej samej nazwie.



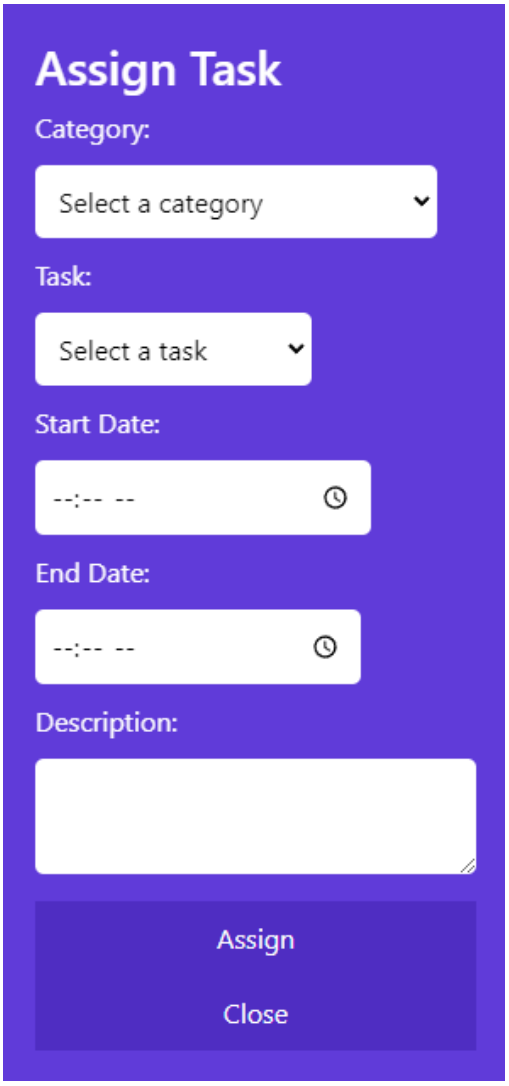
Rys. 8.5. Gotowy interfejsu użytkownika dodawania zadań[opracowanie własne]

Po dodaniu zadania pojawi się ono na pasku bocznym przy swojej kategori, widok ten zaprezentowany jest na rysunku 8.6.



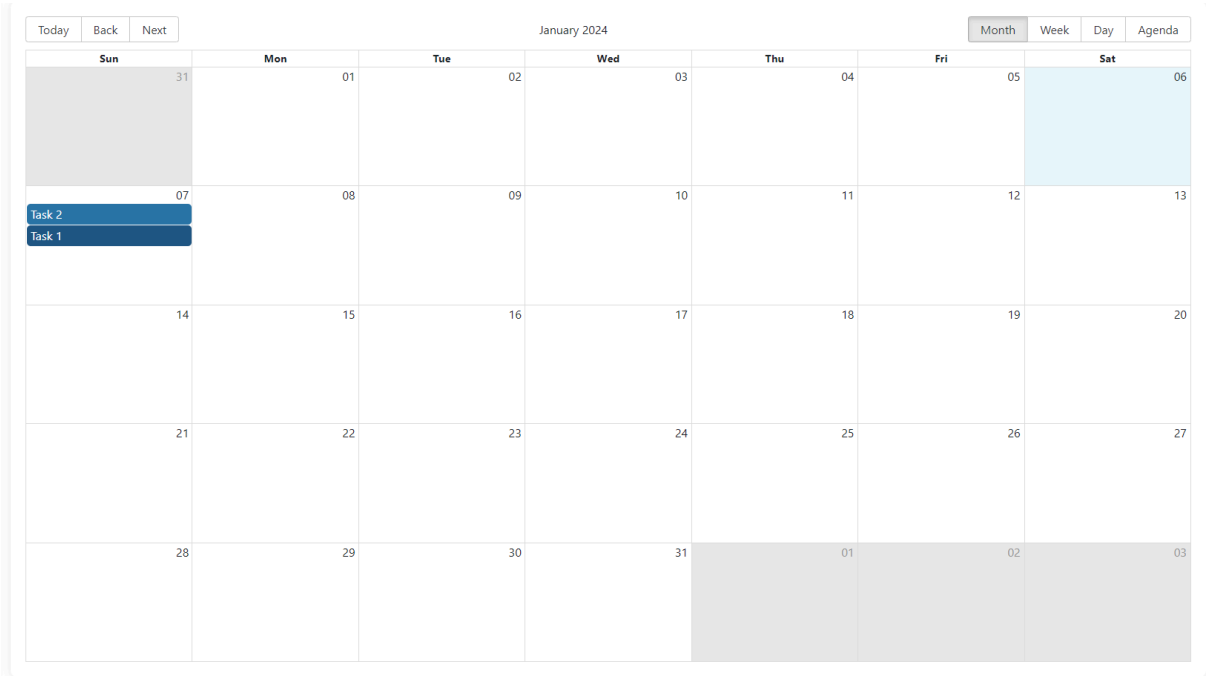
Rys. 8.6. Wygląd dodanego zadania[opracowanie własne]

Do każdego dnia można przypisać utworzone wcześniej zadania. Po kliknięciu na dany dzień pokaże się formularz przypisania zadania (rysunek 8.7). Można w nim wybrać dane zadanie, dodać godziny i opis.

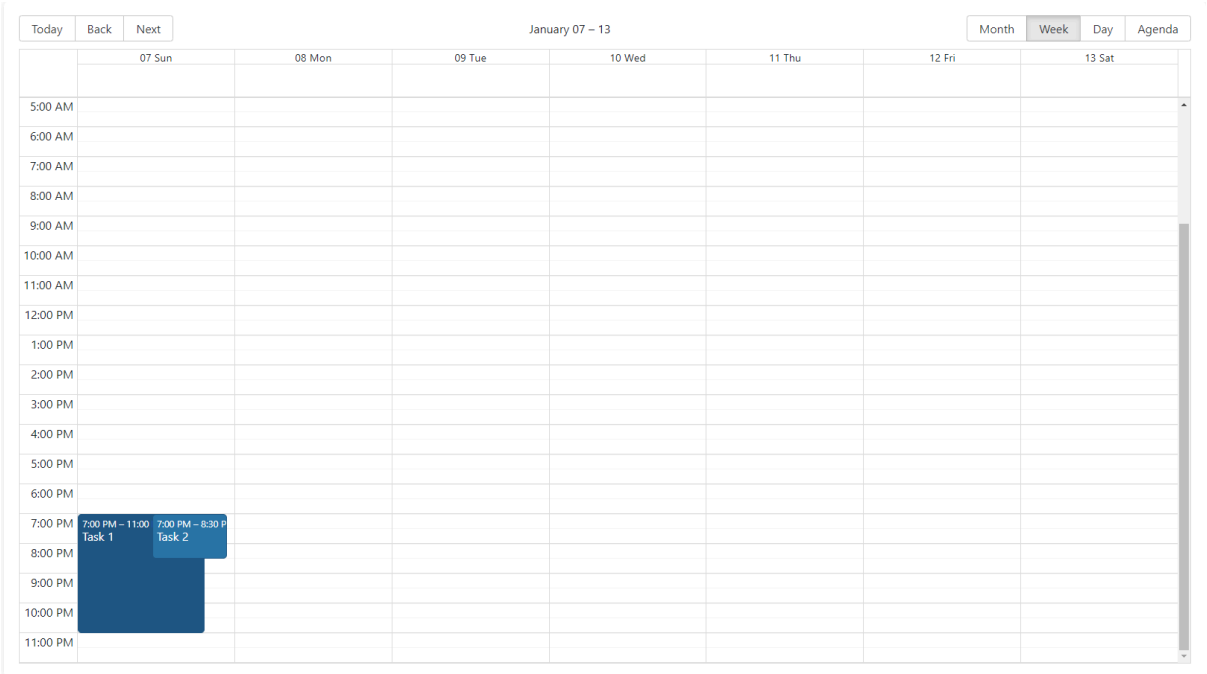


Rys. 8.7. Wygląd przypisywania zadania do dnia[opracowanie własne]

Po poprawnym uzupełnieniu formularza, dane zadanie zostanie dodane do dnia i będzie widoczne dla poszczególnych widoków, tak jak na rysunkach 8.8, 8.9, 8.10, 8.11.

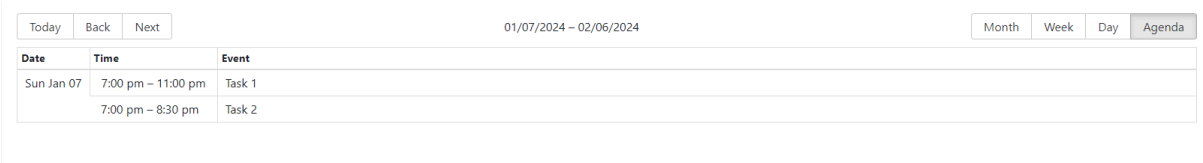


Rys. 8.8. Wygląd przypisanego zadania(widok miesiąca)[opracowanie własne]



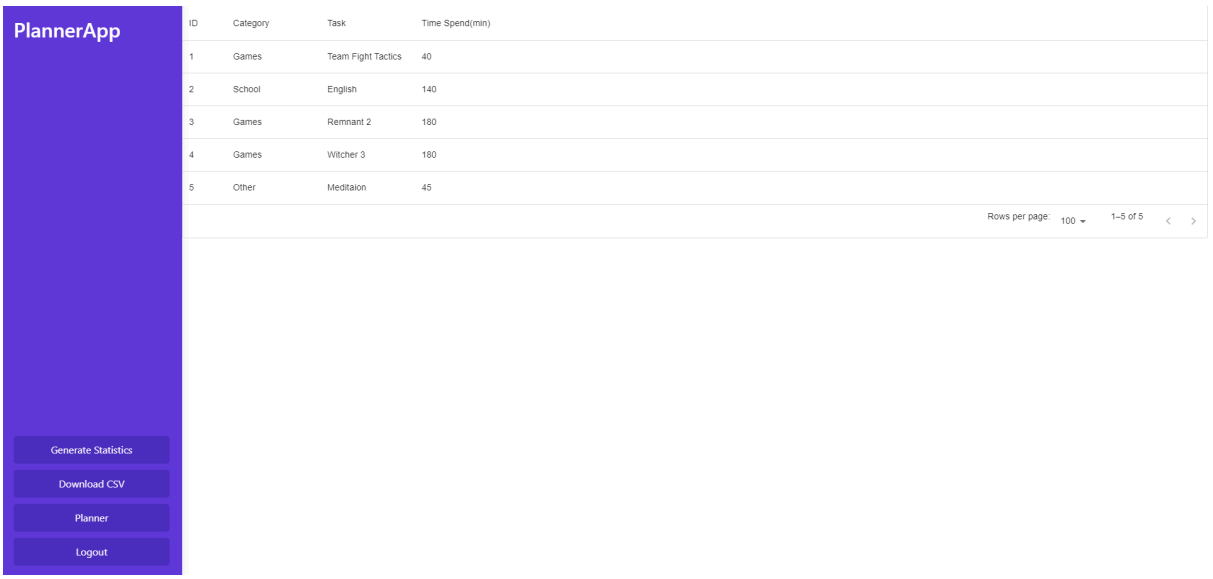
Rys. 8.9. Wygląd przypisanego zadania(widok tygodnia)[opracowanie własne]

Rys. 8.10. Wygląd przypisanego zadania(widok dnia)[opracowanie własne]

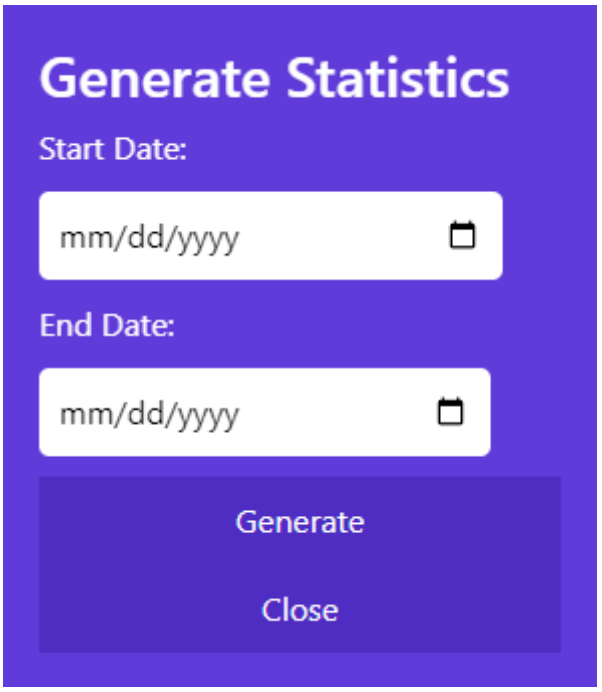


Rys. 8.11. Wygląd przypisanego zadania(widok agenda)[opracowanie własne]

Na rysunku 8.12 widać panel statystyk. Aby wygenerować statystyki należy użyć przycisku na panelu bocznym, gdy go użyjemy ukaże nam się formularz taki jak na rysunku 8.13 w którym należy wybrać daty, na podstawie których możemy wygenerować statystyki. Następnie ukażą nam się statystyki, które posiadają kategorie, zadanie i ilość minut na nie poświęcone.



Rys. 8.12. Wygląd panelu statystyk[opracowanie własne]



Rys. 8.13. Wygląd formularza wyboru dat do generowania statystyk[opracowanie własne]

9. Podsumowanie

Celem pracy dyplomowej było zaprojektowanie i zaimplementowanie aplikacji internetowej, której zadaniem było dostarczenie użytkownikowi możliwości sprawnego zarządzania czasem i zadaniami. Dzięki wykorzystaniu opisanych technologii prace przebiegły pomyślnie i udało się zrealizować podstawowe wymagania oraz cele, które zostały wcześniej założone przy projektowaniu aplikacji.

Dzięki trzymaniu się założeń i dobremu zaprojektowaniu aplikacji, jest ona gotowa do dalszego rozwoju. Ciekawym pomysłem może być zaimplementowanie narzędzia do analizy zbieranych statystyk. Mogłyby one zostać przeanalizowane i przedstawione w ciekawy sposób na przykład poprzez tabele czy wykresy. Innym pomysłem na rozwój jest zaimplementowanie aplikacji mobilnej, gdyż posiadanie smartfona jest powszechne, a łatwiej jest korzystać z aplikacji mobilnej niż z przeglądarki internetowej. Aplikacja mogłaby też przynosić korzyści finansowe, dzięki wykorzystaniu jednego z modeli, pierwszy z nich to ASP, który mógłby działać w ten sposób, że klient raz dokonuje zakupu aplikacji i staje się ona jego własnością, drugi model to SaaS, który obecnie staje się bardzo popularny i polega na tym, że klient płaci miesięczny abonament za produkt, a dostawca jest odpowiedzialny za jego utrzymanie i rozwój. Innym pomysłem jest również udostępnienie aplikacji za darmo z reklamami, które byłyby wyświetlane podczas korzystania z aplikacji.

Niniejsza praca pozwoliła utrwalić wiedzę zdobytą podczas toku studiów, głównie w zakresie projektowania i implementacji aplikacji internetowych z wykorzystaniem takich języków jak Java, JavaScript wraz z frameworkiem Spring Boot i biblioteką React.

Bibliografia

[1] Drake, J. D., & Worsley, J. C. (2002). *Practical PostgreSQL*. " O'Reilly Media, Inc.".

[2] Jazayeri, Mehdi. "Some trends in web application development." Future of Software Engineering (FOSE'07). IEEE, 2007.

[3] Martin, Robert C. Clean code: a handbook of agile software craftsmanship. Pearson Education, 2009.

[4] Rosenberg, Doug, and Kendall Scott. Use case driven object modeling with UML. Reading: Addison-Wesley Professional, 1999.

[5] Walls, C. (2015). Spring Boot in action. Simon and Schuster.

[6](2023) Spring boot documentation.

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[7] (2023) https://maven.apache.org/

[8] Dokumentacja React https://legacy.reactjs.org/docs/getting-started.html

[9] Dokumentacja Spring Data JPA, https://spring.io/projects/spring-data-jpa

[10](2023)https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

Spis rysunków

Rys. 3.1. Diagram Przypadków użycia [opracowanie własne]
Rys. 3.2. Diagram ERD [opracowanie własne]
Rys. 3.3. Diagram aktywności dla rejestracji [opracowanie własne]
Rys. 3.4. Diagram aktywności dla logowania[opracowanie własne]
Rys. 3.5. Diagram aktywności dla dodawania kategorii [opracowanie własne]
Rys. 3.6. Diagram aktywności dla dodawania zadania [opracowanie własne]
Rys. 3.7. Diagram aktywności dla przypisywania zadania do dnia[opracowanie własne]
Rys. 3.8. Diagram aktywności dla generowania i wyświetlania statystyk [opracowanie własne]
Rys. 3.9. Diagram aktywności dla generowania i pobierania statystyk [opracowanie własne]
Rys. 3.10. Diagram sekwencji dla logowania [opracowanie własne]
Rys. 3.11. Diagram sekwencji dla rejestracji [opracowanie własne]
Rys. 3.12. Diagram sekwencji dla tworzenia nowego taska [opracowanie własne]
Rys. 3.13. Diagram sekwencji dla tworzenia nowej kategorii [opracowanie własne]
Rys. 3.14. Diagram sekwencji dla dodawania zadania do dnia [opracowanie własne]
Rys. 3.15. Diagram sekwencji dla generowania i wyświetlania statystyk [opracowanie własne]
Rys. 3.16. Diagram sekwencji dla generowania i pobierania statystyk [opracowanie własne]
Rys. 3.17. Projekt Formularza logowania [opracowanie własne]
Rys. 3.18. Projekt formularza rejestracji [opracowanie własne]
Rys. 3.19. Projekt interfejsu użytkownika dla strony kalendarza[opracowanie własne]
Rys. 3.20. Prototyp interfejsu użytkownika dla strony kalendarza[opracowanie własne]
Rys. 5.1. Wygląd kontrolera zadań[opracowanie własne]
Rys. 5.2. Klasa z mechanizmem szyfrującym[opracowanie własne]
Rys. 5.3. Klasa fasady dla zadań(część 1)[opracowanie własne]
Rys. 5.4. Klasa fasady dla zadań(część 2)[opracowanie własne]
Rys. 5.5. Klasa konfiguracyjna fasady dla zadań[opracowanie własne]
Rys. 5.6. Interfejs JPA dla kategorii[opracowanie własne]
Rys. 5.7.Klasa reprezentująca encje kategorii[opracowanie własne]
Rys. 6.1. Testy jednostkowe dla tworzenia tworzenia zadań[opracowanie własne]
Rys. 6.2. Testy jednostkowe dla przypisywania zadań(część 1)[opracowanie własne]
Rys. 6.3. Testy jednostkowe dla przypisywania zadań(część 2)[opracowanie własne]

Rys. 7.1 Kontroler zadań [opracowanie własne]
Rys. 7.2 Kontroler użytkownika [opracowanie własne]
Rys. 7.3 Kontroler kategorii [opracowanie własne]
Rys. 7.4 Kontroler statystyk[opracowanie własne]
Rys. 8.1. Gotowy interfejsu użytkownika dla logowania[opracowanie własne]
Rys. 8.2. Gotowy interfejsu użytkownika dla rejestracji[opracowanie własne]
Rys. 8.3. Gotowy interfejsu użytkownika dla rejestracji[opracowanie własne]
Rys. 8.4. Gotowy interfejsu użytkownika dla dodawania kategorii[opracowanie własne]
Rys. 8.5. Gotowy interfejsu użytkownika dla dodawania zadań[opracowanie własne]
Rys. 8.6. Wygląd dodanego zadania[opracowanie własne]
Rys. 8.7. Wygląd przypisywania zadania do dnia[opracowanie własne]
Rys. 8.8. Wygląd przypisanego zadania(widok miesiąca)[opracowanie własne]
Rys. 8.9. Wygląd przypisanego zadania(widok tygodnia)[opracowanie własne]
Rys. 8.10. Wygląd przypisanego zadania(widok dnia)[opracowanie własne]
Rys. 8.11. Wygląd przypisanego zadania(widok agenda)[opracowanie własne]
Rys. 8.12. Wygląd panelu statystyk[opracowanie własne]
Rys. 8.13. Wygląd formularza wyboru dat do generowania statystyk[opracowanie własne]