

WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI  
POLITECHNIKA WROCŁAWSKA

# APLIKACJA WSPOMAGAJĄCA ZARZĄDZANIE ZADANIAMI

DANIEL DRAPAŁA  
NR INDEKSU: 244939

Praca inżynierska napisana  
pod kierunkiem  
dr Marcin Kik



Politechnika  
Wrocławska

WROCŁAW 2021



# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>1</b>
1.1	Wprowadzenie . . . . .	1
1.2	Zakres pracy . . . . .	1
1.3	Podział pracy . . . . .	1
<b>2</b>	<b>Analiza problemu</b>	<b>3</b>
2.1	Problem zarządzania zadaniami . . . . .	3
2.2	Podejście Kanban . . . . .	3
2.3	Aplikacja internetowa . . . . .	4
2.4	Porównanie istniejących aplikacji . . . . .	5
<b>3</b>	<b>Projekt</b>	<b>7</b>
3.1	Opis systemu . . . . .	7
3.2	Wzorzec architektoniczny Model-View-Controller . . . . .	7
3.3	Wymagania funkcjonalne i нефункционалне . . . . .	8
3.4	Komunikacja klient - serwer . . . . .	9
3.5	Diagram pakietów . . . . .	10
3.6	Baza danych . . . . .	11
3.6.1	Projekt . . . . .	11
3.6.2	Normalizacja . . . . .	14
3.7	Przypadki użycia . . . . .	15
<b>4</b>	<b>Interfejs użytkownika</b>	<b>23</b>
4.1	Widok strony głównej . . . . .	23
4.2	Widok nagłówka strony . . . . .	24
4.3	Widoki rejestracji i logowania . . . . .	25
4.3.1	Widok zmiany hasła . . . . .	26
4.4	Widok listy zadań i użytkowników . . . . .	27
4.5	Widok Statystyk . . . . .	28
4.6	Widoki związane z tablicą . . . . .	29
4.7	Widok Szczegółów Zadania . . . . .	31
<b>5</b>	<b>Implementacja systemu</b>	<b>33</b>
5.1	Implementacja wzorca MVC . . . . .	33
5.2	Użyte technologie . . . . .	33
5.3	Narzędzia pomocnicze i środowisko programistyczne . . . . .	34
5.4	Opis ważniejszych implementacji . . . . .	34
5.4.1	Bezpieczeństwo aplikacji . . . . .	34
5.4.2	Tablica przeciągnij i upuść . . . . .	36
5.4.3	Rysowanie wykresów . . . . .	37
<b>6</b>	<b>Instalacja i wdrożenie</b>	<b>39</b>
<b>7</b>	<b>Podsumowanie</b>	<b>41</b>



7.1	podsumowanie pracy . . . . .	41
7.2	Plany rozwoju aplikacji . . . . .	41
<b>Bibliografia</b>		<b>43</b>
<b>Spis rysunków</b>		<b>45</b>
<b>Spis kodów źródłowych</b>		<b>47</b>
<b>A Zawartość płyty CD</b>		<b>49</b>

# Wstęp

## 1.1 Wprowadzenie

Optymalizacja procesów, kosztów czy czasu pracy to kluczowe zagadnienia, z którymi borykają się zarówno duże, jak i małe przedsiębiorstwa. Problem dotyczy również pojedynczych osób, które chcą wykorzystać produktywnie otrzymaną dobę, nie zaniedbując żadnej sfery życia. Przedsiębiorcy zarządzają kosztami, jakością produktu oraz czasem pracowników, tak aby uzyskać zadowolenie klienta przy jednoczesnym zysku dla firmy. Konstruktorzy przy zachowaniu norm i przepisów starają się znaleźć optymalne proporcje pomiędzy zachowaniem odpowiednich parametrów konstrukcyjnych, a kosztami budowy. Tak więc optymalizacja może być rozumiana jako poszukiwanie najlepszego rozwiązania na podstawie wybranego kryterium. Kierując się chęcią optymalizacji i wzrostu produktywności pracy przy zastosowaniu prostych technik, powstał pomysł na aplikację wspomagającą zarządzanie zadaniami. Obecnie znajomość podejścia takiego jak Kanban, czy Agile w dziedzinach związanych z informatyką jest oczekiwana przez pracodawców. Wielkie firmy, w których pracuje rzesze pracowników korzystają z podanych podejść na co dzień. Za pomocą wielopoziomowych aplikacji, które posiadają skomplikowane funkcje, zostały zaimplementowane podstawowe podejścia Kanban lub Agile. Celem niniejszej pracy inżynierskiej jest próba spopularyzowania podejścia Kanban wśród mniejszych firmy, które nie są związane w żadnym stopniu z dziedziną informatyki.

## 1.2 Zakres pracy

Zakres pracy obejmuje opisanie i stworzenie aplikacji wspierającej zarządzanie zadaniami w zespole pracującym nad dowolnym projektem przy pomocy technologii Java ze szkieletem architektonicznym Spring oraz Angular. Tematem niniejszej pracy jest aplikacja internetowa, która służy do komunikacji, archiwizowania i raportowania wykonywanych zadań wraz z ich przejrzystym wyświetlaniem. Przemysłowa organizacja czasu i zarządzanie zadaniami zwiększa efektywność i pozwala na realizację długookresowych celów wymagających znacznych nakładów pracy lub kosztów. Zastosowanie aplikacji w przedsiębiorstwach pozwoli na zwiększenie obrotów, a tym samym możliwość na większe zyski.

## 1.3 Podział pracy

Niniejsza praca inżynierska składa się z siedmiu rozdziałów. Rozdział pierwszy stanowi wprowadzenie do tematu, w którym wyjaśniono cel, zakres i podział pracy. W drugim rozdziale omówiono problematykę zarządzania zadaniami, proponowane rozwiązanie - podejście Kanban, porównano również istniejące rozwiązania na rynku aplikacji biznesowych. Zostały przeanalizowane dwie aplikacje Trello i LeanKit. Na zakończenie rozdziału zdefiniowano grupę potencjalnych użytkowników, chętnych do korzystania z aplikacji. W rozdziale trzecim przedstawiono projekt systemu, w postaci wymagań funkcjonalnych i нефункциональных oraz przypadków użycia. Cała aplikacja została zobrazowana za pomocą diagramu pakietów. Ostatecznie został zaprezentowany projekt bazy danych. W rozdziale czwartym został opisany wizualny aspekt aplikacji, czyli interfejs użytkownika. Przedstawione zostały najważniejsze panele aplikacji. W piątym rozdziale przedstawiono sposób implementacji, wykorzystywaną technologię, problemy implementacyjne oraz opisy kodów źródłowych ważniejszych elementów aplikacji. W rozdziale szóstym przedstawiono sposób instalacji i wdrożenia systemu w środowisku docelowym. Ostatni rozdział jest podsumowaniem pracy nad aplikacją, jak i przedstawieniem przyszłościowych planów rozwoju aplikacji.



# Analiza problemu

W tym rozdziale omówiony został problem zarządzania zadaniami w zespole oraz proponowany sposób rozwiązania za pomocą podejścia *Kanban* i analiza istniejących aplikacji.

## 2.1 Problem zarządzania zadaniami

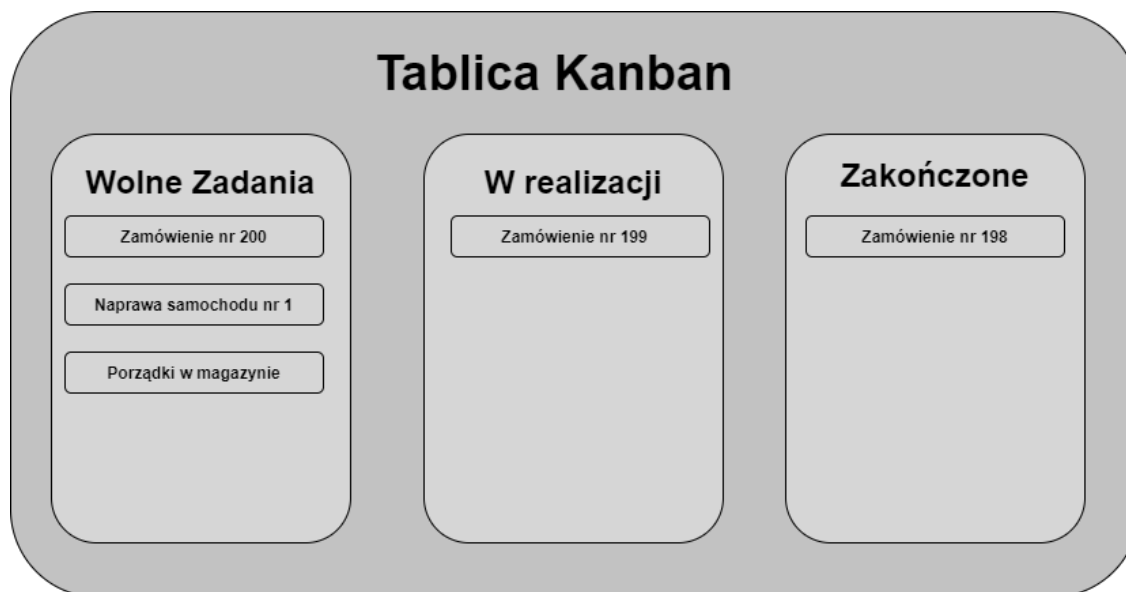
Projekt jest to przygotowany zbiór aktywności zależnych w dany sposób od siebie, zmierzają do wykonania pewnego celu. Może to być na przykład duże wydarzenie, aplikacja, ulepszenie istniejących systemów czy też sama praca inżynierska. Zazwyczaj taki plan obejmuje obszerny zakres pracy, rozciągnięty w planowanym okresie czasu przez określoną liczbę osób nazywanych zespołem. Grupa ludzi dostając informacje na temat stanu końcowego, w większości przypadków nie będzie w stanie wyobrazić sobie ukończenia projektu, dlatego potrzebujemy odpowiedniego zarządzania zadaniami. Złożony problem w tym przypadku dzielimy na różnego rodzaju polecenia, od skomplikowanych, trwających tygodnie, po krótkie, obejmujące przykładowo tylko konsultację w celu potwierdzenia informacji od innego członka drużyny. Jeśli zespół nie posiada wspólnej przestrzeni, wspólne wykonywanie kolejnych zadań staje się coraz bardziej problematyczne. Niektóre zadania wymagają ukończenia poprzednich, znowu niekiedy zdarzy się też tak, że któregoś z poleceń wykonać się nie da. Następuje w takim wypadku wielki problem organizacyjny. Gdyż pracownicy zabierają się za zadania, dobierając je z długiej listy do zrobienia. Początkowo może się wydawać, że wszystko działa sprawnie. Niestety, w tego typu pracy potrzebna jest wzajemna komunikacja, więc zespół organizuje codziennie dwugodzinne narady, na których pracownicy odpowiadają, co udało im się zrobić. Jest to swego rodzaju rozwiązanie, ale ma sporo wad. Cały zespół traci dwie godziny, wszyscy uczestniczą i słuchają o postępach innych osób, których zadania niekiedy nie są zależne od siebie. Musi to być również w pewien sposób udokumentowane, co zostało zrobione, co jest do zrobienia, czego udało się dowiedzieć. Osoba odpowiedzialna za zarządzanie zadaniami, potrzebuje odpowiednich narzędzi do:

- komunikacji w zespole
- dodawania, usuwania, aktualizacji zadań w projekcie
- przechowywania informacji na temat aktualnych poleceń, wykonanych, zablokowanych, czy też archiwalnych
- każdy członek zespołu powinien mieć do takich narzędzi dostęp oraz mieć możliwość dodawania swoich opinii

## 2.2 Podejście Kanban

Podejście *Kanban* to jedna z metodyk wspomagająca zarządzanie zadaniami. Polega ona na zebraniu wszystkich zadań potrzebnych do osiągnięcia pewnego celu i zobrazowanie ich na tablicy, która podzielona jest na różne działy. Jest to jeden z popularniejszych sposobów zarządzania zadaniami, wiele zespołów posiada tablicę *Kanban* w biurze, zbudowaną z białej tablicy i kolorowych karteczek samoprzylepnych. Tablica jest podzielona na dwie części “do zrobienia” i “zrobione”.

Pierwotnym celem systemu *Kanban* było zarządzanie produkcją i redukcja jej kosztów za pomocą wizualnego sterowania. Zgodnie z systemem najpierw należy określić materiał i jego ilość potrzebną do procesu A. Informacja wraz z niezbędnymi surowcami zostaje wysłana do procesu B, tak aby produkcja mogła wystartować. Zostaje wyprodukowana konkretna ilość towaru, a po zakończeniu procesu narzędzia, pojemniki wraz z informacją wraca do pierwotnego procesu A. Rozpoczyna się kolejny cykl pracy. W konsekwencji produkcja jest w stanie dostosować się do zapotrzebowania klientów oraz zminimalizować ewentualną nadprodukcję, a tym samym koszty. System pozwala na koordynację pomiędzy zapotrzebowaniem a wielkością produkcji.

Rysunek 2.1: Prosta tablica *Kanban*

W ten sposób narodziło się podejście *Kanban*, które znalazło zastosowanie poza procesami produkcyjnymi i wykorzystano je w innych dziedzinach przy zarządzaniu zadaniami.

Metoda *Kanban* pochodzi z Japonii. Słowo *kanban* w języku japońskim oznacza szyld, tablicę informacyjną, kartę lub znak. Kluczowym elementem metody jest karta *Kanban*. Przekazuje ona informację dotyczącą przeniesienia materiału wewnątrz zakładu lub od zewnętrznego dostawcy, w analizowanym wypadku zostaje przeniesione określenie zadanie do wykonania. Istnieje przekonanie, że systemy kierowane popytem prowadzą do mniejszej ilości zapasów oraz dynamicznych zmian w produkcji, dzięki czemu wspomagają konkurencyjność firmy. Obecnie informacje mogą być wysyłane drogą elektroniczną, co zmniejsza wykorzystanie kart. Elektroniczne podejście umożliwia eliminację błędów ręcznych czy przypadkowe zagubienie. Co ciekawe system e-kanban można zintegrować z innymi systemami, dzięki czemu otrzymujemy szersze pole danych do optymalizacji produkcji.

Opisana metoda *Kanban* została uproszczona i wykorzystana przy zarządzaniu małymi zespołami. Wykorzystano wizualizację procesu w formie tablicy oraz ograniczono liczbę zadań. Dzięki temu, na pierwszy rzut oka można określić nad czym się pracuje i co będzie robione w następnej kolejności. Dzięki ograniczeniu zadań, które aktualnie są wykonywane, zwiększa się wydajność zespołu. Poprzez stworzenie aplikacji i zastąpienie fizycznej tablicy, możemy dokumentować historię zadań, tworzyć statystyki, eliminować oraz śledzić błędy.

Porównując do innych metod zarządzania, metoda *Kanban* jest prosta w implementacji oraz jej efekty są szybciej zauważalne. Do głównych zalet tablic *Kanban* należy zwiększenie wydajności i produktywności pracowników, usprawnienie komunikacji w zespole oraz doskonalenie i optymalizacja procesów. Metoda *Kanban* może być stosowana w różnych dziedzinach, np. sprzedaż, zarządzanie codziennymi obowiązkami lub programowanie.

## 2.3 Aplikacja internetowa

Aplikacja internetowa do zarządzania zadaniami jest w stanie zapewnić każdy z wypełnionych wymogów, dostarczając niezbędnych informacji na temat postępów i pozostając przyjazna dla użytkownika. Zamysł aplikacji opiera się na stworzeniu wspólnej tablicy, której celem jest przedstawić w przejrzysty sposób zadania i ich statusy. Aktualnie polecenia lub z nadanymi statusami, będą wyświetlać się jako bloki, które można przesuwać po tablicy. Jest to proste zobrazowanie postępu prac nad daną grupą zadań. Osoba zarządzająca może zobaczyć podgląd rozwoju projektu. Natomiast pracownik, który zrobił postęp w wybranym zadaniu,



z łatwością jest w stanie udokumentować i wyświetlić go dla całego zespołu.

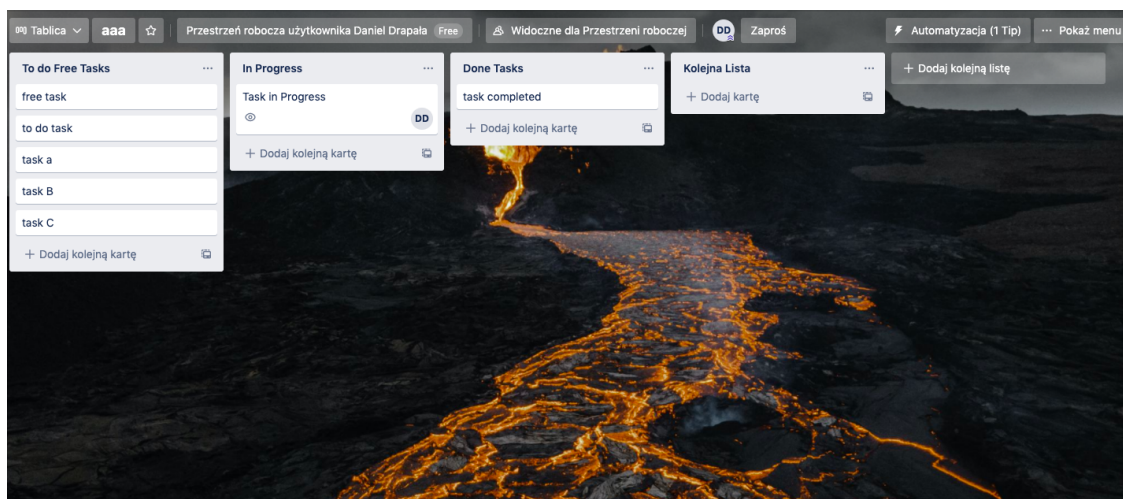
## 2.4 Porównanie istniejących aplikacji

Istnieje szereg aplikacji o zbliżonej funkcjonalności. Opisane zostaną dwie aplikacje: *Trello* i *LeanKit*, ponieważ odzwierciedlają one podobieństwa i różnice pomiędzy aplikacją zaproponowaną w niniejszej pracy inżynierskiej.

Zaczynając od *Trello*, jest to aplikacja bazująca głównie na swej prostocie. Tablicę zadań udostępnioną za pomocą wiadomości e-mail z zaproszeniem, czy też współdzielonego hiperłącza utworzyć można zaledwie w pięć minut. Zadania na tablicy wyglądają na proste, składające się jedynie z opisu, lecz każde zadanie ma swoją stronę, gdzie opis staje się bardziej zaawansowany, można dodać między innymi:

- pliki,
- listę podzadań,
- termin ostatecznego zakończenia zadania,
- etykiety kategoryzujące poszczególne grupy zadań,
- osoby wykonujące zadanie,
- komentarze,

Użytkownicy mogą posiadać dwie role, administratora i zwykłego użytkownika. Różnią się one jedynie tym, że administrator może usuwać i dodawać użytkowników oraz zmieniać ustawienia główne tablicy. Wiele funkcji takich jak przedstawianie zadań na kalendarzu, osie czasu, dodanie lokalizacji do zadań istnieją w wersji płatnej rozszerzonej. Wiele aplikacji, służących do tworzenia tablicy *Kanban* wzoruje się na tej aplikacji i jest ona jak dotąd jedną z najlepszych rozwiązań. Omawiana aplikacja mocno spopularyzowała metodykę zarządzania zadaniami poprzez tablicę *Kanban*.

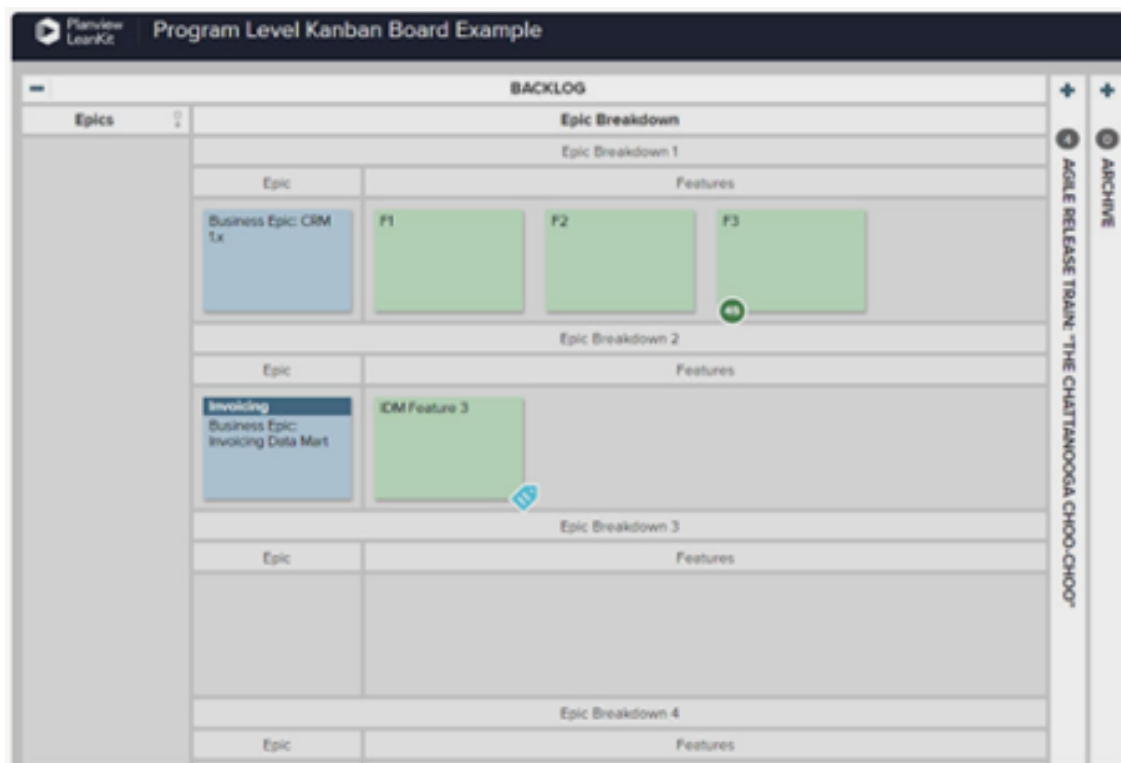


Rysunek 2.2: Przykładowa tablica *Kanban* w aplikacji *Trello*

Kolejną aplikacją jest *LeanKit*, która została wybrana jako druga aplikacja do porównania, ponieważ tej tablicy nie charakteryzuje prostota. Jej atutem są złożone funkcje tworzenia tablicy. Najbardziej charakterystyczną opcją jest skomplikowane tworzenie karty na tablicy. Poprzednia aplikacja pozwalała na tablicy utworzyć karty przetrzymujące zadania, które można przenosić z jednej karty na drugą. Aplikacja oferuje tworzenie karty, podkarty, podkarty podkarty, i tak dalej, przez co bardziej skomplikowane do skategoryzowania zadania użytkownik jest w stanie opisać w jednym miejscu. Zadania posiadają podobne atrybuty do



*Trello*, oprócz tego posiadają jeszcze dziedziczenie zadań, to znaczy, że każde zadanie ma listę zadań rodziców i listę zadań dzieci. Są to listy przetrzymujące odnośniki do zadań, z wyniku których powstało dane zadanie (rodzice), lub zadania, które zostały utworzone przez dane zadanie (dzieci). Poza skomplikowaną reprezentacją tablicy *Kanban*, *LeanKit* posiada wiele funkcji po wizualizację zadań w kalendarzu, możliwość wertykalnej i horyzontalnej tablicy (jak i wertykalno-horyzontalnej), raporty w postach czystych danych lub wykresów czy też tworzenia zależności pomiędzy wieloma innymi projektami i tablicami. Jest to aplikacja posiadająca kompleksowe konfiguracje, użytkownik korzystający z programu po raz pierwszy najprawdopodobniej potrzebowałby specjalistycznego szkolenia w celu wykorzystania wszystkich możliwości.



Rysunek 2.3: Przykładowa tablica *Kanban* w aplikacji *LeanKit*

Porównanie istniejących aplikacji uwidoczniło cechy, które zostały wdrożone do TaskManager. Program również charakteryzuje się prostotą. Proces inicjalizacji tablicy, projektów jest dłuższy, aby potem aplikacja mogła służyć uczestnikom projektu. Pierwszą różnicą jest podział na dwie role, ADMIN, USER. Rola administratora projektu (Admin) polega na posiadaniu pełnej kontroli nad dodawaniem i konfiguracją użytkowników, zadań oraz tablic. Drugą cechą charakterystyczną jest zakładka statystyk, gdzie zobaczyć można ogólne informacje na temat całego projektu, wyników poszczególnych użytkowników, jak i zadań w przeciągu ostatnich sześciu miesięcy. Kluczową cechą, która może przekonać klienta do korzystania z TaskManager jest to produkcyjny wewnętrzny system typu klient serwer niezależny od innych aplikacji. Wraz ze wzrastającymi oczekiwaniami klient może zlecić rozszerzenie programu na swój sposób, czy to przez podłączenie do kanału komunikacyjnego lub kalendarza z grafiką pracowniczym.

# Projekt

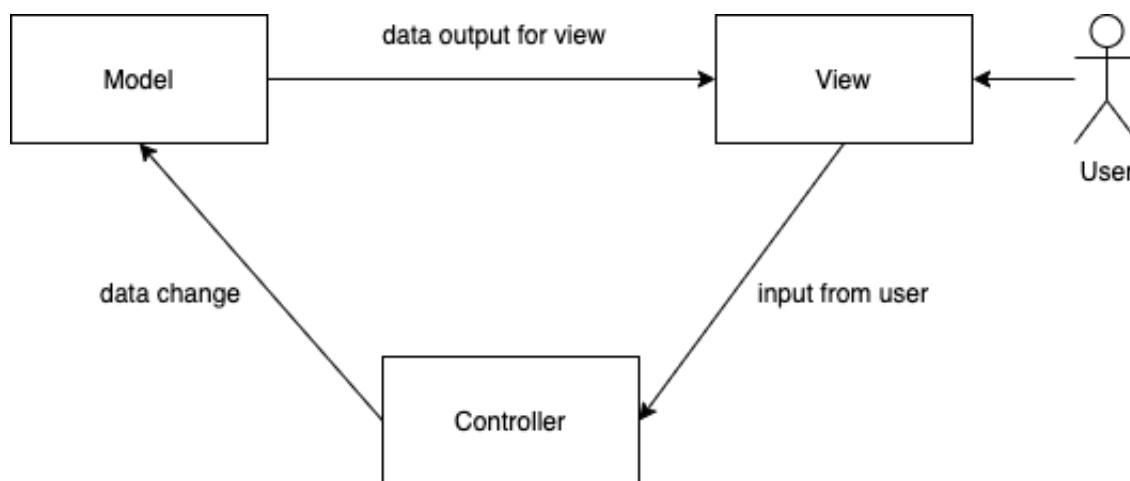
W danym rozdziale opisana została aplikacja pod względem technicznym.

## 3.1 Opis systemu

Tworzenie aplikacji internetowej to skomplikowany proces, który może przysporzyć wiele problemów w przyszłości. Pierwszym problemem jest utrzymanie istniejącego kodu, to znaczy kontrolowanie stanu aplikacji i naprawianie pojawiających się błędów, związanych z pierwotną implementacją. Drugim problemem jest możliwość rozwoju aplikacji. Kiedy aplikacja przynosi zyski i sprawdza się w kontakcie z użytkownikiem, oczywistym podejściem jest rozbudowa aplikacji. W tym celu podczas planowania implementacji trzeba zastanowić się nad odpowiednim podejściem. W tym celu powstały wzorce projektowe, które pomagają programistom uporać się z różnymi problemami w sposób zorganizowany i zoptymalizowany, ułatwiają rozwiązywanie błędów podczas implementacji oraz zwiększają czytelność kodu.

## 3.2 Wzorzec architektoniczny Model-View-Controller

System zbudowany został w oparciu o model MVC (*Model-View-Controller*). Jest to podejście architektoniczne, które dzieli aplikację na trzy główne komponenty. *Model*, odpowiedzialny za dostęp i przechowywanie danych. *View*, czyli widok odpowiada za interfejs użytkownika, jak dane są przedstawiane i odbierane. *Controller* zajmuje się logiką biznesową, komunikując się z modelem i widokiem. Widok dostarcza informacje od klienta, które obsługiwane są przez *Controller* i odpowiednie zmiany wprowadzane są do modelu. [13]



Rysunek 3.1: Komunikacja pomiędzy komponentami we wzorcu MVC

Zalety korzystania ze wzorca *MVC* to lepsza przejrzystość kodu, skalowalność i utrzymanie. W momencie wystąpienia błędu, może zostać obciążona odpowiedzialnością jedna z trzech części systemu, więc optymalizuje to znacząco proces szukania miejsca, w którym dany błąd występuje.



### 3.3 Wymagania funkcjonalne i нефункционалне

Wymaganiami funkcjonalnymi nazywamy zbiór szczegółów, pomagających określić to co system powinien wykonywać w zamyśle architektury systemu. Wymagania нефункционалне to zazwyczaj techniczne reguły, które system powinien spełniać, takie jak na przykład strona internetowa prawidłowo wyświetlana przez pięć lub więcej wiodących przeglądarek.

#### Wymagania funkcjonalne

1. Przegląd listy użytkowników
2. Przegląd listy zadań
3. Tworzenie, edytowanie, usuwanie zadań
4. Korzystanie z Tablicy Kanban w celu raportowania postępów w zadaniach
5. Wyświetlanie Statystyk projektu w postaci wykresów
6. Podstawowa autentykacja użytkownika
7. Wyświetlanie szczegółów związanych z danym zadaniem. Szczegółowe parametry zadania to:
  - (a) nazwa
  - (b) data do zakończenia zadania
  - (c) użytkownik przypisany do zadania
  - (d) kolumna, w której aktualnie znajduje się dane zadanie
  - (e) opis zadania
  - (f) sekcja komentarzy
  - (g) nazwa użytkownika tworzącego zadanie

#### Wymagania funkcjonalne dodatkowe dla administratora

1. Tworzenie, aktywacja kont użytkowników, dostęp do większej ilości danych w porównaniu do użytkownika
2. Ustawienie czasu po jakim czyszczona jest historia czynności związanych z zadaniami
3. Konfiguracja tablicy:
  - (a) określenie nazwy tablicy i kolumn
  - (b) określenie liczby kolumn
  - (c) wybranie domyślnej kolumny (na którą wprowadzane będą nowe zadania)
  - (d) wybranie końcowej kolumny (automatycznie zmienia status zadania na zakończony) – opcjonalne

#### Wymagania нефункционалне

1. prostota w użytkowaniu
2. aplikacja responsywna na tyle, aby można było skorzystać z głównych funkcji systemu za pomocą urządzenia mobilnego
- 3.

### 3.4 Komunikacja klient - serwer

Zbiór założeń, który definiuje komunikację pomiędzy systemami nazywamy API (z angielskiego Application Programming Interface, czyli interfejs programowania aplikacji). Komunikacja odbywa się za pomocą protokołów HTTP (*Hypertext Transfer Protocol*). Protokół HTTP jest to sposób komunikacji z określonym zbiorem reguł. Wiadomości dzielą się na żądania i odpowiedzi. Klient wysyła zapytania, oczekuje na odpowiedź. Serwer oczekuje na zapytania, odsyłając na nie odpowiednie odpowiedzi [11].

Protokół ten składa się z:

- Metoda protokołu i wersja
- Nagłówek (z ang. *header*), który posiada wszystkie potrzebne informacje na temat połączenia
- Treści wiadomości (z ang. *body*)
- wiadomość zwrotna zawsze posiada status odpowiedzi w postaci kodu i wiadomości (na przykład 200 OK)

Metody protokołu pozwalają rozróżnić rodzaj wiadomości. W aplikacji używana są metody:

- POST - wiadomość przychodząca od klienta, zazwyczaj używany do stworzenia obiektu na bazie danych
- GET - zapytanie o pobór danych za pomocą wskazanej ścieżki może to być lista obiektów z bazy danych czy też jeden specyficzny obiekt, którego identyfikator zawarty jest w ścieżce ( /api/task/1)
- PUT - jest to pokrewna metoda do POST, lecz używana do aktualizowania istniejących danych zamiast wprowadzania nowych
- DELETE - metoda służy do usuwania zasobów z bazy danych

Serwisy służące do komunikacji tworzone były ze standardem *Representational State Transfer (REST)*. Jest to podejście architektoniczne, które spełniając standard korzystania z komunikacji protokołem HTTP oraz korzystaniem z URI. Oprócz tego przestrzegać musi również zasad takich jak:

- zasoby są zidentyfikowane poprzez ścieżkę URI
- zasoby mogą mieć wiele reprezentacji
- zasoby mogą być dodawane, aktualizowane, usuwane oraz pobierane za pomocą zwykłych protokołów HTTP
- serwer nie posiada żadnej informacji na temat stanu, wszystkie szczegóły muszą trafiać do serwera w żądaniu od klienta

Przestrzegając powyższych zasad usprawniamy czytelność, niezawodność i skalowalność procesu komunikacji. Żądanie posiada wszystkie szczegóły potrzebne do otrzymania odpowiedzi, więc jest czytelniejsze, zwalnia serwer z niepotrzebnych odpowiedzialności oraz dodawanie nowej ścieżki nie dodaje żadnej informacji do przechowywania dla systemu. [12]

Klient do komunikacji używa klas zaimplementowanych w pakiecie *services*. Składają się one z prostych metod, które potrzebują jedynie odpowiednią ścieżkę, odpowiednią metodę HTTP i oczekiwany typ danych. Przykładowa klasa z pakietu *rest* 3.1 udostępnia metodę na ścieżkę *"/task"*. Metoda wykonuje walidację i próbę zapisania przysłanego obiektu w bazie danych, jeśli proces powiedzie się odsyłana jest wiadomość OK ze statusem 200 i stworzonym obiektem.



## 3.5 Diagram pakietów

Pakiety podzielone są na elementy serwera i aplikacji webowej, przedstawione zostały na diagramie 3.2.

### Serwer

**repository** posiada charakterystyczne dla wzorca architektonicznego Spring Data klasy z adnotacją `@Repository`. Są to klasy, które udostępniają gotowe proste zapytania do bazy danych.

**domain** definiuje model danych. Są to tabele i ich wszystkie atrybuty przepisane do klas języka Java.

**config** jak sama nazwa wskazuje posiada wszystkie klasy związane z konfiguracją serwera poczynając od sposobu formatowania dat, wyboru domyślnego języka strony, czy też konfigurację dostępu użytkowników do potrzebnych interfejsów.

**service** to grupa klas z adnotacją `@Service` odpowiedzialnych za wykonywanie logiki związanej z modyfikacją i odpowiednim zapisywaniem danych. Korzysta z pakietu repository, a sam wykorzystywany jest w pakiecie rest.

**rest** jest odpowiedzialny za schemat API serwera. Udostępnia metody na podane ścieżki zwane URL, które umożliwiają komunikowanie się z serwerem. Przykładowo aplikacja webowa wysyła na serwer zapytanie HTTP POST na URL *www.taskManager.com/task*. Serwer odczytuje zapytanie i przygotowuje odpowiedź. W tym celu odpowiednia klasa z pakietu rest mapuje podane zapytanie do odpowiedniej metody, która wykonuje akurat w tym przykładzie zapisanie zadania do bazy danych i zwraca zrotną wiadomość 3.1.

**security** zawiera klasy odpowiedzialne za autentykację połączenia, zanim jakiegokolwiek zapytanie trafi do metod z pakietu rest, najpierw serwer oczekuje na odpowiedź ze strony klienta z informacją o sukcesie autoryzacji i pozwoleni jakie zalogowany użytkownik posiada.

```
@PostMapping("/task")
public ResponseEntity<Task> createTask(@Valid @RequestBody TaskDTO taskDTO){
    if (taskDTO.getId() != null) {
        (...)
    }
    return ResponseEntity
        .headers(HeaderUtil.createAlert(applicationName))
        .body(task);
}
```

Kod źródłowy 3.1: Uproszczona metoda kontrolera z pakietu rest

### aplikacja webowa

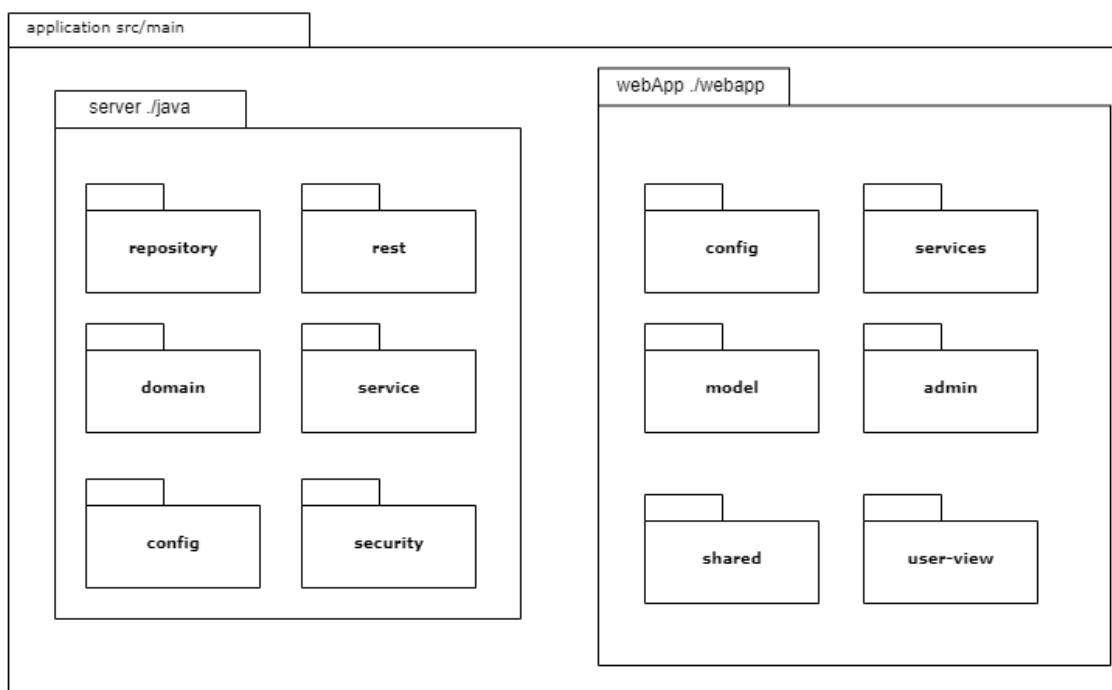
**config, model i service** to pakiety o tych samych właściwościach co w serwerze. Warto zauważyć, że model zapisany po stronie klienta może być ograniczony do atrybutów potrzebnych jedynie do odpowiedniego wyświetlenia interfejsu dla użytkownika.

**user-view** posiada wszystkie komponenty widoków dostępne dla użytkownika i administratora, są to główne widoki aplikacji, takie jak tablica, statystyki, lista zadań i tym podobne.

**admin** posiada komponenty dostępne tylko dla administratora i wszelką logikę związaną z konfiguracją nowych użytkowników.

**shared** jest to pakiet, na który składają się wszystkie wspólne dla wszystkich innych klas komponenty, czyli bloki wyświetlające błędy, responsywna lista, nagłówek strony, stopka strony.

Poniżej znajduje się diagram pakietów projektu wygenerowany przez narzędzie do rysowania diagramów diagrams.net [3]



Rysunek 3.2: Diagram pakietów z których złożona jest cała aplikacja

## 3.6 Baza danych

### 3.6.1 Projekt

Sekcja ta w pełni skupiona jest na budowie bazy danych. Jako baza danych wykorzystana została baza dystrybucji MySQL jako że jest to jedna z relacyjnych baz danych ogólnodostępnych. MySQL cechuje się skalowalnością i przede wszystkim ochroną danych. Wraz z możliwościami rozwoju danej aplikacji, postawiono na uniwersalną bazę danych, którą bez problemu można rozbudować i skomplikować. W celu kontroli wersji projektu bazy danych do serwera zaimplementowała zostało narzędzie zwane *Liquibase*, które przechowuje wszystkie skrypty związane z tworzeniem bazy danych, modyfikacją tabel lub danych w plikach o formacie XML. *Liquibase* pomaga wprowadzać zmiany na istniejącej bazie danych przy zachowaniu poprzedniego stanu, zostawiając po udanej modyfikacji historię zmian.

Narzędzie *Liquibase* generuje dwie tabele 3.3 w bazie danych. Tabela zapisuje nazwę pliku i jej sumę kontrolną, więc zapobiega to zmianom na plikach inicjalizujących bazę danych. Jest to bardzo pomocna rzecz dla aplikacji produkcyjnych, ponieważ bazy danych dużych systemów wykorzystywanych w czasie rzeczywistym są bardzo ważne i niebędna jest szczególna kontrola zmian na szkieletie bazy danych.



databasechangellog	databasechangelloglock
ID VARCHAR(255)	ID INT
AUTHOR VARCHAR(255)	LOCKED BIT(1)
FILENAME VARCHAR(255)	LOCKGRANTED DATETIME
DATEEXECUTED DATETIME	LOCKEDBY VARCHAR(255)
ORDEREXECUTED INT	
EXECTYPE VARCHAR(10)	
MD5SUM VARCHAR(35)	
DESCRIPTION VARCHAR(255)	
COMMENTS VARCHAR(255)	
TAG VARCHAR(255)	
LIQUIBASE VARCHAR(20)	
CONTEXTS VARCHAR(255)	
LABELS VARCHAR(255)	
DEPLOYMENT_ID VARCHAR(10)	

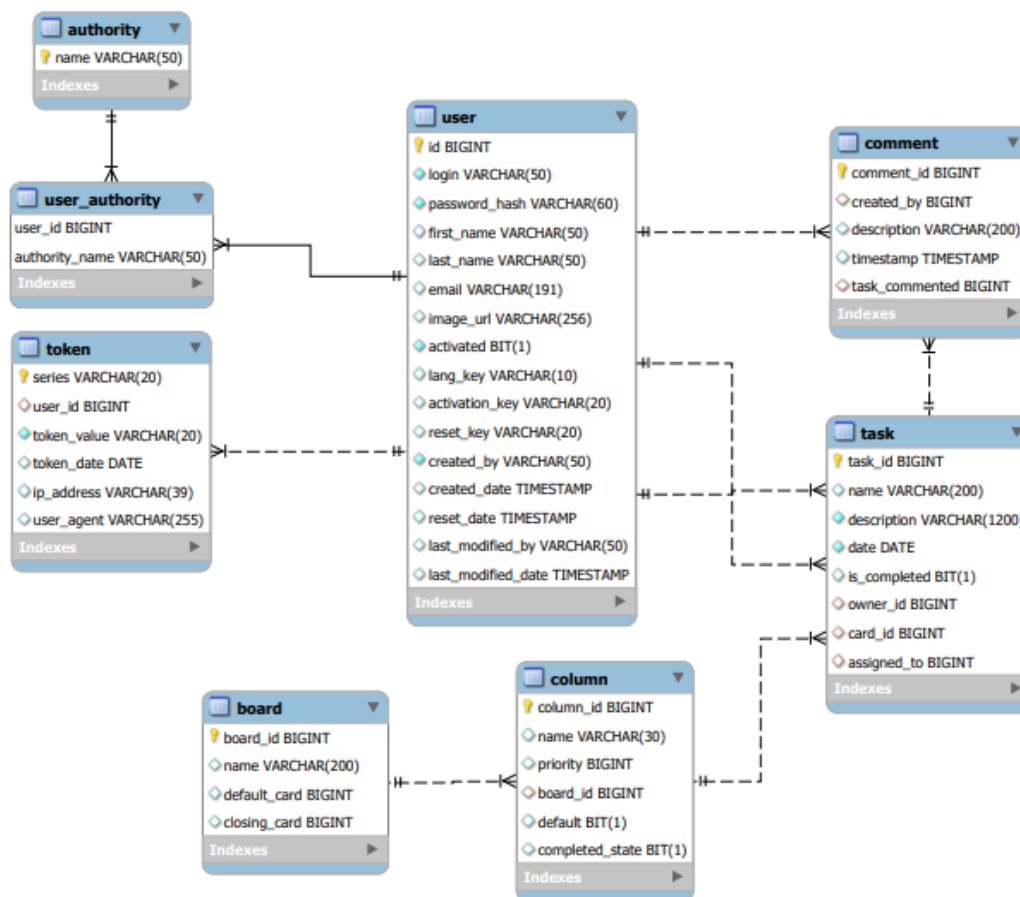
Rysunek 3.3: Tabele kontroli wersji

Główny szkielet bazy danych 3.4 składa się z tabel:

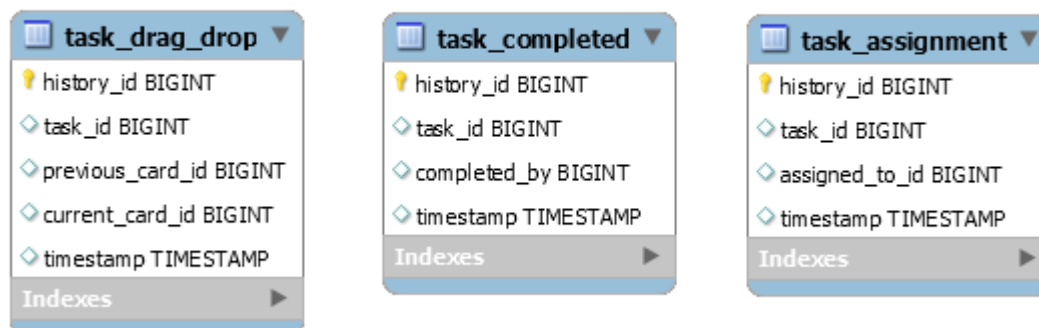
- *board* - informacje na temat tablicy
- *column* - atrybuty kolumny tablicy
- *user* - atrybuty użytkownika wykorzystywane do autentykacji oraz relacje z zadaniami i komentarzami
- *task* - posiadająca atrybuty zadania
- *comment* - tabela komentarzy zadań
- *user\_authority* - tabela mapująca użytkowników do ich uprawnień
- *authority* - tabela opisująca dostępne uprawnienia
- *token* - tabela opisująca indywidualny token użytkownika, który ułatwia użytkownikowi logowanie się na stronę

Oprócz głównych tabel do bazy danych dodane zostały również trzy nierelacyjne tabele, które audytują trzy zachowania, przeciąganie zadania z jednej kolumny na drugą (*task\_drag-drop*), przetrzymywanie daty skończenia (*task\_completed*) i daty przypięcia zadania do użytkownika (*task\_assignment*) 3.5.





Rysunek 3.4: Diagram bazy danych



Rysunek 3.5: Nierelacyjne tabele odpowiedzialne za przetrzymywanie danych potrzebnych do statystyk



### 3.6.2 Normalizacja

Schemat bazy danych stworzony może zostać na wiele, niekoniecznie dobrych sposobów. Normalizacja to czynność, która prowadzi do zoptymalizowania sposobu przetrzymywania danych]. Polega ona na odpowiednich zmianach na tabelach i relacjach, zgodnie z pewnym zbiorem zasad, w celu wyeliminowania niepotrzebnych relacji oraz redundancji danych. Pierwsze trzy zbiory reguł normalizacyjnych nazwane i zdefiniowane zostały przez Edgara Franka Codd. Jest to pierwsza postać normalna (1NF), druga postać normalna (2NF) i trzecia postać normalna (3NF). Zbiory te mają ze sobą pewną zależność. Pierwsza postać normalna jest podzbiorem reguł drugiej postaci normalnej, a druga postać normalna jest podzbiorem trzeciej postaci normalnej. Istnieje więcej postaci normalnych, lecz projekt bazy danych opisywanej aplikacji poddany został normalizacji do 3NF. [10]

#### Pierwsza postać normalna

**Definicja.** Relacja jest w pierwszej postaci normalnej, jeśli każda krotka posiada dokładnie jedną wartość dla danego atrybutu oraz nie istnieją powtarzające się krotki.

Wszystkie tabele spełniają pierwszą postać normalną. Każda tabela dzieli dane na kolumny o atomowych wartościach. Nigdzie nie występują krotki posiadające listę wartości. Wartości typu imię i nazwisko użytkownika zostały podzielone na osobne kolumny. Krotki nie powtarzają się, ponieważ każda tabela posiada klucz główny.

#### Druga postać normalna

**Definicja.** Relacja jest w drugiej postaci normalnej, jeśli jest w pierwszej postaci normalnej oraz żaden atrybut niekluczowy nie jest zależny funkcyjnie od właściwego podzbioru klucza minimalnego

Główne tabele w bazie danych tworzone zostały tak, aby kluczem głównym był tylko jeden atrybut jakim jest identyfikator, więc atrybuty niekluczowe są zależne funkcyjnie tylko od klucza głównego. Sprawdzona zostanie jednocześnie druga i trzecia postać normalna.

#### Trzecia postać normalna

**Definicja.** Relacja jest w trzeciej postaci normalnej wtedy i tylko wtedy, gdy jest w drugiej postaci normalnej oraz wszystkie niekluczowe atrybuty zależą tylko od pełnego klucza minimalnego.

Jedna para atrybutów niekluczowych jest zależna funkcyjnie. Jest to klucz prywatny *column\_id* oraz atrybut *is\_completed* w tabeli *task*. Tabela *board* posiada informacje na temat identyfikatora, która kolumna zamyka zadania. Klucz obcy oznaczający kolumnę, w której zadanie aktualnie przebywa przenosi nam jednocześnie informacje o tym czy dane zadanie jest zamknięte. Jednak na potrzeby czytelności atrybut *is\_completed* pozostał w bazie.

### 3.7 Przypadki użycia

W tym podrozdziale przedstawione zostały najważniejsze przypadki użycia aplikacji.

Rejestracja nowego użytkownika	
Aktorzy:	klient, serwer autoryzacyjny administrator
Warunki wstępne:	sesja klienta nie jest autoryzowana
Scenariusz:	<ol style="list-style-type: none"><li>1. klient wchodzi na panel rejestracji za pomocą linka "Create Account" lub używając przycisku na nagłówku strony</li><li>2. po wypełnieniu przez klienta formularza rejestracyjnego zachodzi proces walidacji danych</li><li>3. gdy walidacja przeszła pomyślnie: system wyświetla komunikat o sukcesie rejestracji i informuje, że potrzebna jest jeszcze aktywacja konta<ol style="list-style-type: none"><li>(a) System zapisuje dane na bazie, hasło zostaje zakodowane.</li><li>(b) System generuje losowy 8-cyfrowy klucz aktywacyjny dla użytkownika i wysyła na podany w formularzu e-mail link aktywacyjny.</li><li>(c) Klient aktywuje konto za pomocą linka, który otrzymał na skrzynkę pocztową e-mail.</li></ol></li><li>4. gdy walidacja zauważyła błędy w formularzu: wyświetla się komunikat błędu, zaznaczone zostają błędne pola</li></ol>

Tabela 3.1: Przypadek użycia - rejestracja nowego użytkownika



Logowanie	
Aktorzy:	klient , serwer autoryzacyjny, system prezentujący
Warunki wstępne:	sesja klienta nie jest autoryzowana, klient posiada aktywne konto
Scenariusz:	<ol style="list-style-type: none"> <li>1. klient wchodzi na panel logowania za pomocą linka "Sign in" lub używając przycisku na nagłówku strony</li> <li>2. klient wprowadza prawidłowe dane, nie zaznacza pola 'remember me' <ol style="list-style-type: none"> <li>(a) system sprawdza czy istnieje użytkownik o podanych informacjach kwalifikujących</li> <li>(b) system autoryzacyjny odnajduje w bazie użytkownika,</li> <li>(c) system wysyła do systemu prezentującego wiadomość o poprawnym procesie weryfikacji, więc użytkownik zostaje przekierowany do widoku głównego po udanym logowaniu i przyznawane są prawa do przeglądania stron odpowiednich dla roli użytkownika, czyli ROLE_ADMIN lub ROLE_USER</li> </ol> </li> <li>3. klient wprowadza prawidłowe dane, zaznacza pole 'remember me' <ol style="list-style-type: none"> <li>(a) system autoryzacyjny odnajduje w bazie użytkownika,</li> <li>(b) system autoryzacyjny tworzy 20-cyfrowy unikalny kod zwany tokenem i umieszcza go dla danego użytkownika</li> <li>(c) system wysyła do systemu prezentującego wiadomość o poprawnym procesie weryfikacji, więc użytkownik zostaje przekierowany do widoku głównego po udanym logowaniu i przyznawane są prawa do przeglądania stron odpowiednich dla roli użytkownika, czyli ROLE_ADMIN lub ROLE_USER</li> <li>(d) sesja użytkownika zostaje zapamiętana</li> </ol> </li> <li>4. klient wprowadza złe dane <ol style="list-style-type: none"> <li>(a) system nie odnajduje użytkownika o podanych informacjach kwalifikujących, komunikuje się z systemem prezentującym o braku podanego użytkownika</li> <li>(b) system prezentujący wyświetla komunikat o prawdopodobnie źle wpisanym loginie lub hasle.</li> </ol> </li> </ol>

Tabela 3.2: Przypadek użycia - Logowanie

Dodawanie Zadania	
Aktorzy:	użytkownik, serwer prezentujący, serwer obsługujący bazę danych
Warunki wstępne:	sesja użytkownika jest autoryzowana i jest on na stronie "Board" lub "Task List"
Scenariusz:	<ol style="list-style-type: none"><li>1. Użytkownik przechodzi do panelu tworzenia nowego zadania po wciśnięciu przycisku "Create Task" zarówno na stronie "Board" jak i "Tasks List"</li><li>2. pojawia się formularz z informacjami do podania na temat zadania, niektóre niezbędne do stworzenia zadania, niektóre opcjonalne</li><li>3. po ukończeniu wypełniania formularza i kliknięciu przycisku "Save" system waliduje poprawność otrzymanych danych:<ol style="list-style-type: none"><li>(a) gdy dane odnośnie zadania zgadzają się, użytkownik przekierowywany jest na stronę, z której otworzył panel tworzenia zadania</li><li>(b) gdy dane odnośnie zadania nie zgadzają się, panel prezentujący wyświetla pola z błędnymi informacjami</li><li>(c) użytkownik może poprawić dane i próbować stworzyć zadanie ponownie</li></ol></li></ol>

Tabela 3.3: Przypadek użycia - Dodawanie Zadania

Tworzenie tablicy	
Aktorzy:	użytkownik, serwer prezentujący,
Warunki wstępne:	sesja użytkownika jest autoryzowana i jest on na stronie "Board", lecz w bazie danych nie ma żadnych rekordów reprezentujących tablicę.
Scenariusz:	<ol style="list-style-type: none"><li>1. Użytkownik przechodzi do panelu "Board", serwer nie znajduje istniejącej tablicy, więc przekierowuje użytkownika do formularza tworzenia Tablicy</li><li>2. pojawia się formularz z informacjami do podania na temat tablicy</li><li>3. potrzebne do wypełnienia pola to nazwa tablicy, kolejno nazwa kolumny, po kliknięciu przycisku "Add Column", wyskakuje kolejne pole z nazwą dla kolejnej kolumny</li><li>4. po dodaniu kolumn wybierane zostają kolumny domyślna i zamykająca zadania</li><li>5. użytkownik po kliknięciu "Save" przekierowany jest na stronę "board", gdzie widzi uprzednio utworzoną tablicę</li></ol>

Tabela 3.4: Przypadek użycia - Tworzenie tablicy



Oglądanie szczegółów Zadania, edytowania zadania i usunięcie zadania	
Aktorzy:	użytkownik, serwer prezentujący, serwer obsługujący bazę danych
Warunki wstępne:	sesja użytkownika jest autoryzowana i jest on na stronie "Board" lub "Task List"
Scenariusz:	<ol style="list-style-type: none"> <li>1. Użytkownik przechodzi do panelu uaktualniania istniejącego zadania po wciśnięciu przycisku "Go To Task", który wyświetlany jest dla każdego zadania na jednej z kolumn listy zadań w panelu "Tasks List"</li> <li>2. Użytkownik może również przejść do panelu uaktualniania za pomocą kliknięcia w kontener wizualizujący zadanie na tablicy kanban w panelu "Board" <ol style="list-style-type: none"> <li>(a) system prezentujący wyświetla wszystkie informacje na temat istniejącego zadania oraz dwa przyciski "Go Back" "Save" <ol style="list-style-type: none"> <li>i. użytkownik używa przycisku Save, informacje zawarte w formularzu wysyłane są do serwera w celu zaktualizowania zadania</li> <li>ii. użytkownik po obejrzeniu informacji na temat zadania może za pomocą przycisku "go Back" wrócić do wcześniej odwiedzanego panelu</li> </ol> </li> </ol> </li> </ol>

Tabela 3.5: Przypadek użycia - Oglądanie szczegółów zadania, edytowania zadania

Przeglądanie listy zadań	
Aktorzy:	użytkownik, serwer prezentujący, serwer obsługujący bazę danych
Warunki wstępne:	sesja użytkownika jest autoryzowana i wyświetlony jest panel listy zadań "Tasks List"
Scenariusz:	<ol style="list-style-type: none"> <li>1. użytkownik jest na panelu "Tasks List"</li> <li>2. wyświetlana jest przez system prezentujący lista zadań na którym widoczne są najważniejsze informacje</li> <li>3. system prezentujący otrzymuje listę ograniczoną parametrami z serwera obsługującego bazę danych</li> <li>4. użytkownik klikając na nazwę kolumny jest w stanie sortować ją rosnąco lub malejąco</li> <li>5. ostatnią kolumną jest przycisk "Go to Task", dzięki któremu możemy przejść do panelu danego taska</li> </ol>

Tabela 3.6: Przypadek użycia -Przeglądanie listy zadań

Przeglądanie listy użytkowników	
Aktorzy:	użytkownik, serwer prezentujący, serwer obsługujący bazę danych
Warunki wstępne:	sesja użytkownika jest autoryzowana i wyświetlony jest panel listy użytkowników "Users List", rolę użytkownika jest "ROLE_USER", czyli uprawnienia zwykłego użytkownika
Scenariusz:	<ol style="list-style-type: none"><li>1. użytkownik jest na panelu "Users List"</li><li>2. wyświetlana jest przez system prezentujący lista zadań na którym widoczne są najważniejsze informacje</li><li>3. system prezentujący otrzymuje listę ograniczoną parametrami z serwera obsługującego bazę danych</li><li>4. użytkownik klikając na nazwę kolumny jest w stanie sortować ją rosnąco lub malejąco</li><li>5. ostatnią kolumną jest przycisk "Go to Task", dzięki któremu możemy przejść do panelu danego taska</li></ol>

Tabela 3.7: Przypadek użycia - Przeglądanie listy użytkowników

Przeglądanie panelu ze statystykami	
Aktorzy:	użytkownik, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana i wypełniona są tabele związane z historią
Scenariusz:	<ol style="list-style-type: none"><li>1. użytkownik jest na panelu "Charts"</li><li>2. wyświetlane są przez system prezentujący informacje na temat projektu i 2 wykresy związane ogólnie ze statystykami projektu</li><li>3. niżej znajdują się dwie listy do wyboru, lista użytkowników i zadań.</li><li>4. użytkownik wybiera odpowiednich użytkowników na liście, aby wyświetlić wykresy związane z ich wynikami w ostatnich 6 miesiącach</li><li>5. użytkownik wybiera odpowiednie zadania na liście, po czym wyświetlane zostają wykresy dla każdego odznaczonego zadania.</li></ol>

Tabela 3.8: Przypadek użycia - Przeglądanie panelu ze statystykami



Aktualizowanie zadania za pomocą tablicy Kanban	
Akotrzy:	użytkownik, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana i na tablicy znajdują się różne zadania
Scenariusz:	<ol style="list-style-type: none"> <li>1. użytkownik jest na panelu "Board" widzi wszystkie kolumny i zadania w nich</li> <li>2. użytkownik za pomocą wydarzenia przycisnij i upuść przeciąga zadanie z kolumny A do kolumny B</li> <li>3. system odpowiednio zapisuje dane wydarzenie, sprawdzając czy kolumna B nie jest równocześnie kolumną zamykającą zadania</li> <li>4. po upuszczeniu na kolumnę B zadanie znajduje się na liście zadań dla kolumny B, serwer zapisuje wydarzenie zarówno w informacjach na temat kolumny i zadania, ale także wprowadza wydarzenie do specjalnej tabeli archiwizującej.</li> </ol>

Tabela 3.9: Przypadek użycia - Aktualizowanie zadania za pomocą tablicy Kanban

Konfiguracja tablicy	
Akotrzy:	administrator, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana jako administrator
Scenariusz:	<ol style="list-style-type: none"> <li>1. użytkownik wchodzi na panelu "Manage Board" z poziomu nagłówka strony</li> <li>2. system wyświetla formularz związany z informacjami tablicy, czyli nazwy i identyfikatory kolumny domyślnej i zamykającej, która ustawia zadanie na ukończone.</li> <li>3. użytkownik zmienia odpowiednie parametry, a końcowe informacje wysyła do serwera po zaakceptowaniu przyciskiem "save"</li> <li>4. serwer weryfikuje poprawność danych i aktualizuje potrzebne informacje</li> </ol>

Tabela 3.10: Przypadek użycia - Konfiguracja tablicy



Edycja użytkowników przez administratora	
Akotrzy:	administrator, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana jako administrator
Scenariusz:	<ol style="list-style-type: none"><li>1. użytkownik wchodzi na panelu "Manage users" z poziomu nagłówka strony</li><li>2. system wyświetla formularz związany z informacjami użytkowników, informacji jest więcej niż na liście udostępnionej dla użytkownika</li><li>3. administrator jest w stanie usunąć, stworzyć nowe konto, ale też dezaktywować istniejące.</li><li>4. gdy administrator kliknie "create user" musi wypełnić wyskakujący formularz, po wypełnieniu serwer wysyła na odpowiedni adres email informacje o utworzeniu konta przez użytkownika</li></ol>

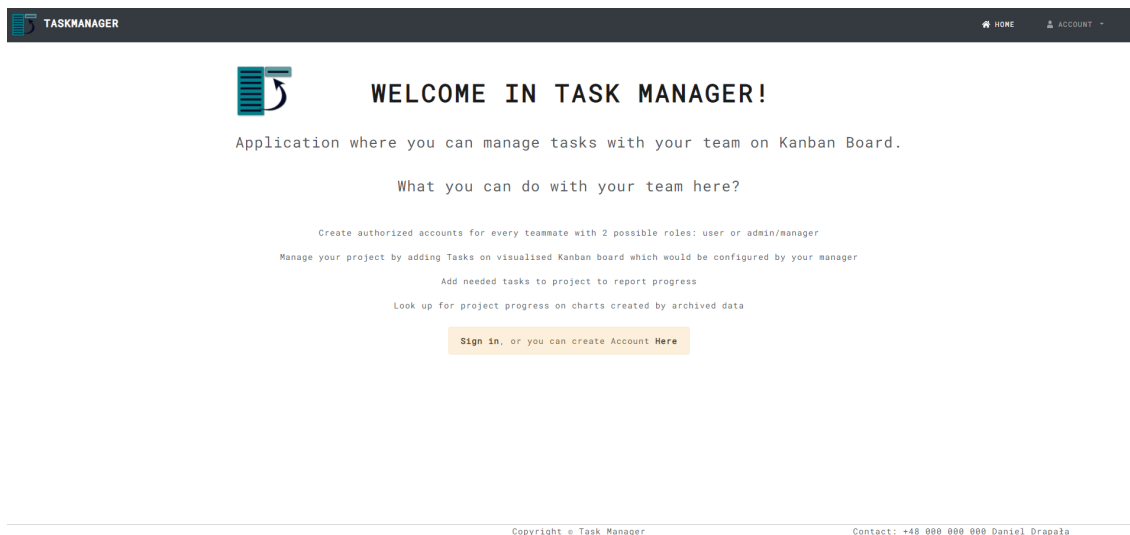
Tabela 3.11: Przypadek użycia - edycja użytkowników przez administratora



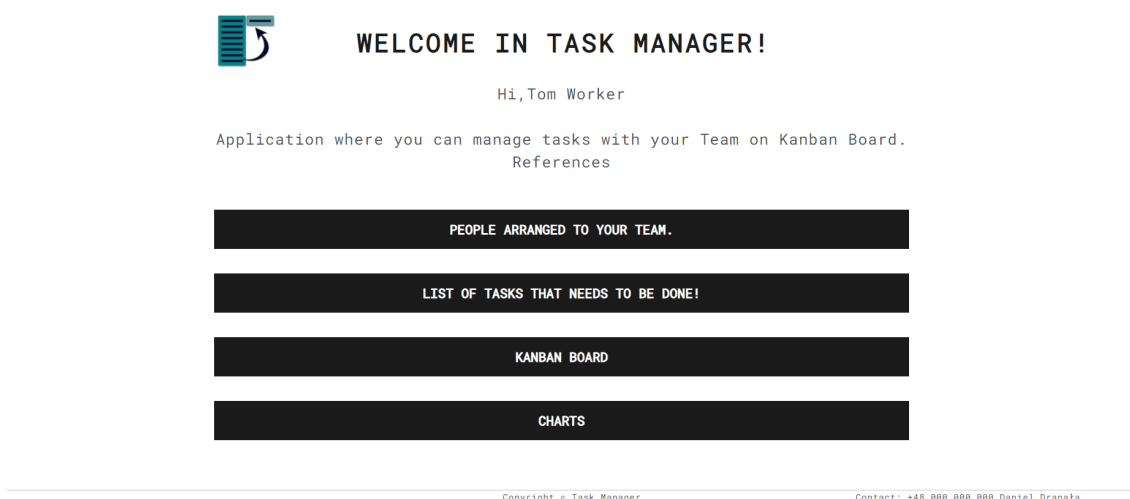
# Interfejs użytkownika

## 4.1 Widok strony głównej

Widok strony głównej zależy od tego czy sesja użytkownika jest autoryzowana. Przed zalogowaniem wyświetlane są informacje na temat aplikacji. Po zalogowaniu zobaczyć możemy przyciski przekierowujące użytkownika do głównych podmiotów aplikacji.



Rysunek 4.1: Widok strony głównej przed zalogowaniem.



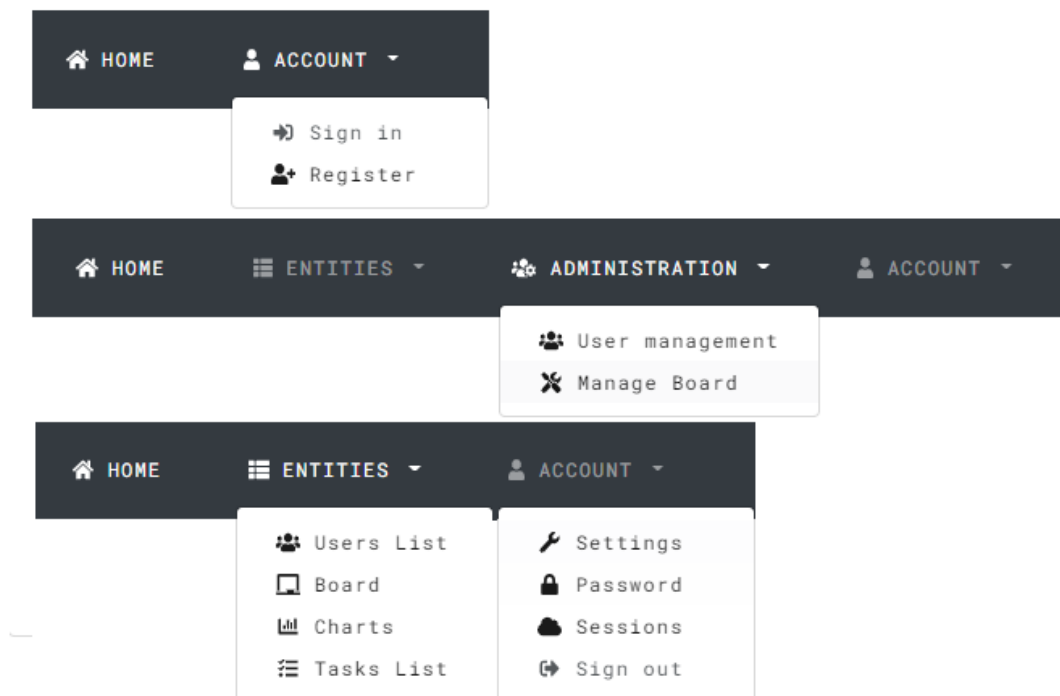
Rysunek 4.2: Widok strony głównej po zalogowaniu.



## 4.2 Widok nagłówka strony

Na rysunku 4.3 przedstawiony został widok nagłówka, który jest elementem umiejscowionym na górze strony na każdym dostępnym widoku. Posiada przyciski przekierowujące do innych podmiotów aplikacji. Stan z najmniejszą liczbą dostępnych opcji jest wyświetlany dla użytkownika niezalogowanego. Jedynymi możliwościami jest wtedy strona startowa, strona rejestracyjna oraz strona z formularzem logowania. Drugi stan jest wyświetlany dla administratora, który oprócz funkcji użytkownika posiada również zakładkę *Administration*, gdzie może przejść do zarządzania użytkownikami lub tablicą. Trzeci stan dla użytkownika o zwykłych uprawnieniach składa się z odnośnika do strony startowej, ale również wszystkie dostępne podmioty strony:

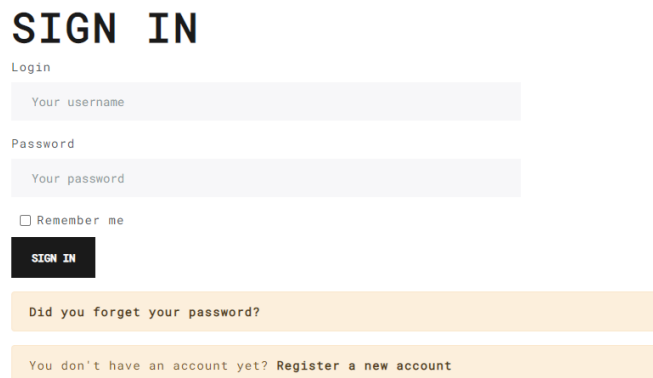
- lista użytkowników, która jest jedynie listą podglądową, nie posiada ona licznych funkcji jak lista dla administratora 4.8
- tablica kanban
- lista zadań
- wykresy
- ustawienia konta
- zmiana hasła
- aktywne, zapisane sesje
- przycisk wylogowania się z sesji



Rysunek 4.3: Widok nagłówka w trzech stanach.

## 4.3 Widoki rejestracji i logowania

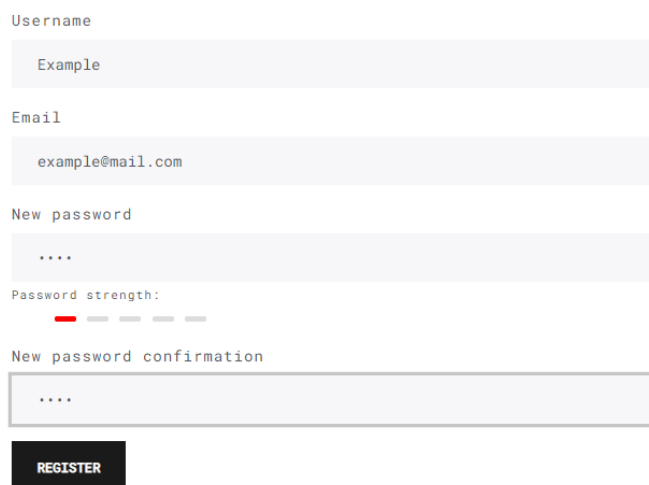
Widok rejestracji i logowania składają się z formularza do wypełnienia, aby dostarczyć serwerowi odpowiednie dane do autoryzacji.



The login form is titled "SIGN IN" in large, bold, black letters. Below the title, the word "Login" is written in a smaller font. There are two input fields: "Your username" and "Your password", both with light gray borders and placeholder text. Below the password field is a checkbox labeled "Remember me". A black button with the text "SIGN IN" in white is positioned below the checkbox. At the bottom of the form, there are two orange links: "Did you forget your password?" and "You don't have an account yet? Register a new account".

Rysunek 4.4: Widok logowania.

## REGISTRATION



The registration form is titled "REGISTRATION" in large, bold, black letters. Below the title, the word "Registration" is written in a smaller font. There are four input fields: "Username" with placeholder text "Example", "Email" with placeholder text "example@mail.com", "New password" with placeholder text "....", and "New password confirmation" with placeholder text "....". Below the "New password" field is a "Password strength:" indicator with a red bar and four gray bars. A black button with the text "REGISTER" in white is positioned below the "New password confirmation" field.

Rysunek 4.5: Widok rejestracji nowego użytkownika.



### 4.3.1 Widok zmiany hasła

Widok 4.6 prezentuje formularz ze zmianą hasła. Ciekawym dodatkiem wprowadzonym do danego widoku jest pasek siły hasła. Jest to styl składający się z pięciu bloków kolorowanych odpowiednio na kolor czerwony, pomarańczowy i zielony w zależności od tego jak silne hasło zostało wprowadzone. Odpowiedzialna za kolorowanie tych bloków metoda analizuje wprowadzone hasło pod kątem skomplikowania za pomocą wyrażeń regularnych. Sprawdzane zostaje czy w hasle użyty został co najmniej jeden symbol specjalny, jedna wielka litera, jedna mała litera oraz liczba.

The screenshot shows a web interface for changing a password. At the top, the title "PASSWORD FOR [ADMIN]" is displayed. Below it, a red error message box states: "The password and its confirmation do not match!". The form contains three input fields: "Current password" (with 5 dots), "New password" (with 8 dots), and "New password confirmation" (with 4 dots). Below the "New password" field, there is a "Password strength:" label followed by a progress bar consisting of five segments: the first four are green and the fifth is grey. At the bottom of the form is a black button with the text "SAVE" in white.

Rysunek 4.6: Widok zmiany hasła.

## 4.4 Widok listy zadań i użytkowników

Widok list dostępnych z aplikacji składają się z responsywnych list, które wyświetlają informacje wyciągane z bazy danych w postaci stron. Serwis wysyłający zapytanie o listę użytkowników bądź zadań dołącza informacje o ograniczeniach. Podane listy ograniczone są do dwudziestu pięciu elementów na stronę. Serwer wysyła maksymalną liczbę elementów oraz informację na temat liczby pozostałych elementów. Za pomocą tych danych otrzymujemy listę podzieloną na strony. Przejście na kolejną stronę skutkowałoby wysłaniem zapytania o kolejne dwadzieścia pięć elementów.

LIST OF ALL TASKS

You can create more tasks or update others on specific task information :

CREATE TASK

ID ^	TASK NAME ↕	STATUS	ASSIGNED TO	OPTIONS
1	Delivery no. 000044441	Free Tasks	Tom Worker	options ▾ Show Task Delete Task
2	Problems with standpoint nr 3	Work blocking issues	Joe Manager	
3	Delivery no. 000044442	Free Tasks	Tom Worker	options ▾
4	Delivery no. 000044440	Completed tasks	Tom Worker	options ▾

Showing 1 - 4 of 4 items.

« « 1 » »

Rysunek 4.7: Widok listy zadań.

USERS

REFRESH LIST + CREATE A NEW USER

ID ^	LOGIN ↕	EMAIL ↕	PROFILES	CREATED DATE ↕	LAST MODIFIED BY ↕	LAST MODIFIED DATE ↕	
1	admin	joe.manager@mail.com	ACTIVATED ROLE_USER ROLE_ADMIN		system		VIEW EDIT DELETE
2	user	tom.worker@mail.com	ACTIVATED ROLE_USER		system		VIEW EDIT DELETE

Showing 1 - 2 of 2 items.

« « 1 » »

Rysunek 4.8: Widok listy użytkowników dla administratora.



## 4.5 Widok Statystyk

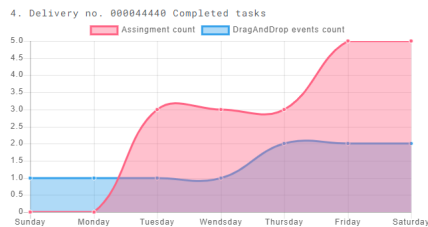
Statystyki przedstawione są na przejrzystym panelu. Na początku strony wyświetlane zostają informacje na temat projektu wraz z wykresami kołowymi przedstawiające ogólny postęp wykonanych zadań. Niżej widnieje lista użytkowników oraz lista zadań. Po wybraniu odpowiednich rekordów z listy wyświetlane zostają po kolei wykresy dla poszczególnych zadań jak na rysunku 4.10

### STATISTICS

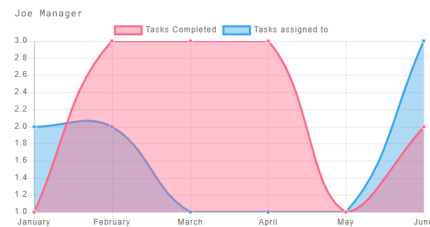
Project : Warehouse Weekly Issues  
Columns:  
Column Completed tasks containing 1 tasks  
Column Free Tasks containing 2 tasks  
Column Tasks in progress containing 0 tasks  
Column Work blocking issues containing 1 tasks



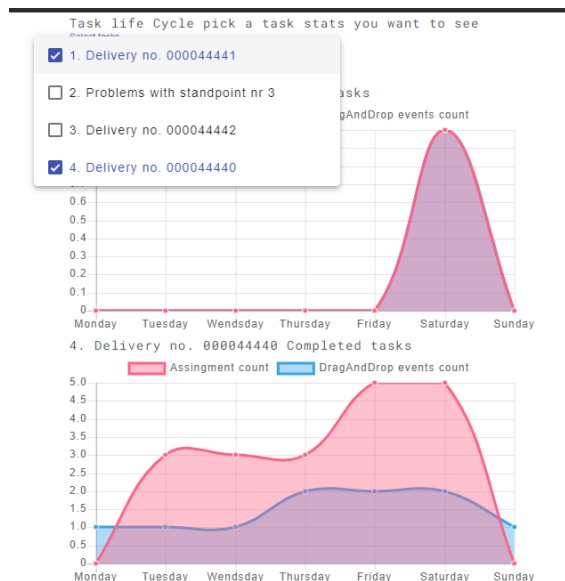
Task life Cycle pick a task stats you want to see  
Select tasks  
4. Delivery no. 00004440



Choose user to see his performance Assigned line chart Completed line chart  
Select users  
Joe Manager



Rysunek 4.9: Widok statystyk.



Rysunek 4.10: Widok statystyk dla wymienionych z listy zadań.



## 4.6 Widoki związane z tablicą

Board

Name

Simple Kanban Board

Columns **Add Column**

Column name:

Free Tasks

Column name:

Tasks in Progress

Column name:

Completed Tasks

Default Column

Free Tasks

Closing Column

Completed Tasks

**SAVE**

Rysunek 4.11: Widok dodawania tablicy.

### EDIT A BOARD

ID

1

Name

Warehouse Weekly Issues

Default Column

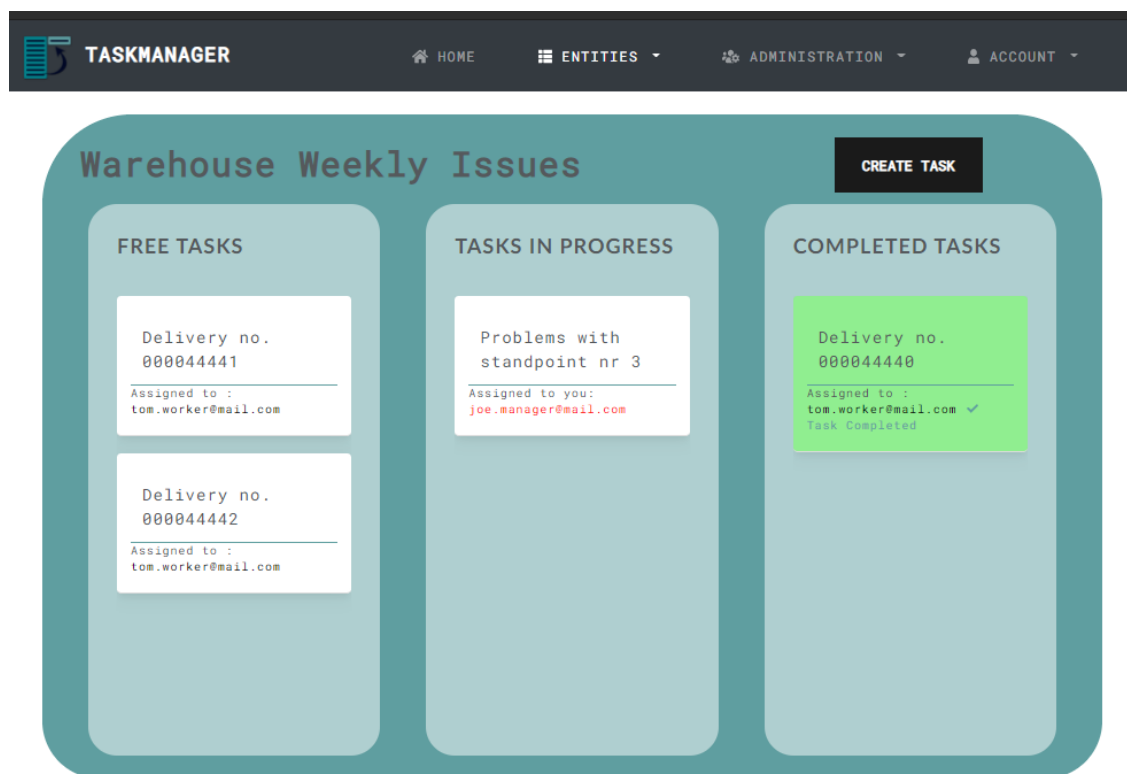
2

Column for task closing (last in project cycle)

1

**CANCEL** **SAVE**

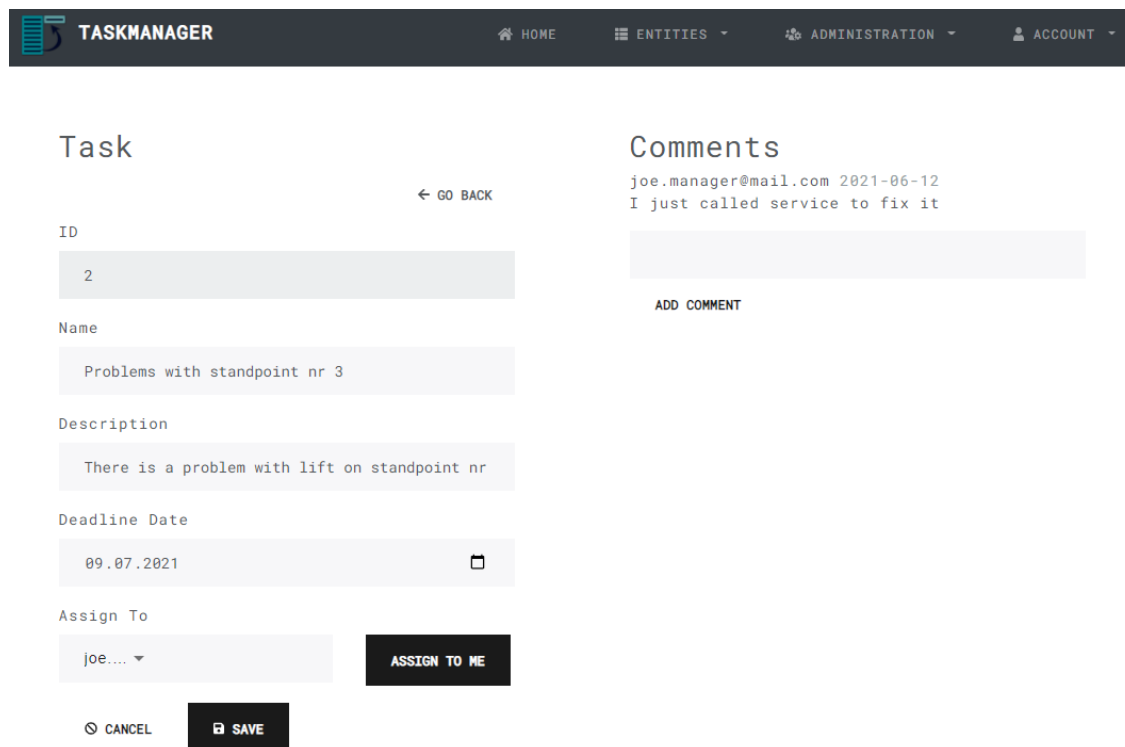
Rysunek 4.12: Widok edycji tablicy.



Rysunek 4.13: Widok tablicy.

## 4.7 Widok Szczegółów Zadania

Widok 4.14 przedstawiający szczegóły zadania za pomocą formularza, który może zostać w każdej chwili zaktualizowany. W szczegółach zadania możemy wprowadzać odpowiednio komentarze dotyczące zadania.



**TASKMANAGER** HOME ENTITIES ADMINISTRATION ACCOUNT

### Task

← GO BACK

ID

2

Name

Problems with standpoint nr 3

Description

There is a problem with lift on standpoint nr

Deadline Date

09.07.2021

Assign To

joe....

ASSIGN TO ME

CANCEL SAVE

### Comments

joe.manager@mail.com 2021-06-12

I just called service to fix it

ADD COMMENT

Rysunek 4.14: Widok szczegółów zadania.



# Implementacja systemu

W tym rozdziale opisane zostały aspekty implementacyjne takie jak wzorzec architektoniczny aplikacji, narzędzia użyte do stworzenia aplikacji i ciekawe problemy napotkane podczas tworzenia serwisu.

## 5.1 Implementacja wzorca MVC

Opis implementacji wzorca projektowego MVC podzielony został na przedstawienie modelu, widoku i kontrolera.

**Model** Model reprezentowany jest przez model w bazie danych oraz przez serwisy na serwerze. Każda tabela posiada swój serwis. Jest to komponent, który odpowiedzialny jest za odpowiednie wprowadzanie i wyciąganie potrzebnych informacji z tabeli, którą reprezentują. Model został opisany wraz z przedstawieniem projektu bazy danych

**View - Widok** Widok zaimplementowany jest przy pomocy szkieletu architektonicznego Angular korzystającego z HTML oraz TypeScript. Podstawowym obiektem w szkielecie jest komponent, na który składa się również skryptowy plik html i css lub scss. Komponent jest definicją jednego z wielu widoków w aplikacji, które Angular przedstawia na głównej stronie

**Controller - Kontroler** Kontroler zaimplementowany jest głównie po stronie serwera, lecz komponenty odpowiedzialne za widok posiadają w sobie wstępne modyfikacje na danych otrzymanych od użytkownika takie jak przykładowo walidacja tych danych. Zaletą podejścia, aby funkcje kontrolera spełnianie były praktycznie po otrzymaniu jakichkolwiek danych jest to, że informacje, które przychodzą do serwera, aby przeszły przez główną logikę serwera są danymi nie podatnymi na błędy, więc serwer odpowiedzialny za logikę, nie będzie musiał skupiać się na odsyłaniu błędnych danych z powrotem do widoku, a o wiele częściej będzie otrzymywać już poprawne dane. Wadą takiego podejścia jest większe skomplikowanie w kodzie, ponieważ walidacja odbywa się wtedy zarówno na serwerze jak i po stronie klienta. Aplikacja odpowiedzialna za widoku musi również posiadać wiedzę na temat modelu bazy danych i poszczególne cechy różnych danych, aby odpowiednia logika mogła zostać zastosowana.

## 5.2 Użyte technologie

Projekt w pełni kompatybilny z systemem Windows 10 wykorzystuje z najważniejszych technologii:

- Java 11
- Angular 11
- Bootstrap 4.6

W języku programowania Java taką pomocniczą technologią jest szkielet architektoniczny Spring. Posiada on dużo przeróżnych gałęzi i jest najpopularniejszym pomocniczym narzędziem dla języka Java. Aplikacja TaskManager korzysta z gałęzi Spring Boot, Spring Security oraz Spring Data JPA.

*Spring Security* zapewnia odpowiednie klasy wspomagające konfigurację autoryzacji i kontroli dostępu do zaimplementowanego serwera. [9]

*Spring Data JPA* ułatwia dostęp do bazy danych udostępniając potrzebne metody wyciągające odpowiednie informacje z bazy danych. [7]



*Spring Boot* to gałąź, która pomaga uporać się ze wszystkimi możliwościami tego szkieletu i pomaga skonfigurować aplikację. W tym celu korzystając z narzędzia *Spring Boot Initializer* [8] cały szkielet projektu został zainicjalizowany i dostępny do pobrania. Kolejnym z ważnych narzędzi pomocniczych użytych podczas implementacji serwera jest *Maven*. Jest to narzędzie, które ułatwia zarządzanie projektem, ujednolica budowanie i kompilowanie zaawansowanego projektu napisanego w języku Java [2].

## 5.3 Narzędzia pomocnicze i środowisko programistyczne

Program napisany został przy pomocy **IntelliJ IDEA**. Jest to środowisko programistyczne, które w pełni wspomaga pisanie dużych projektów w języku Java. Możliwość debugowania kodu i pomocne wtyczki, które przykładowo pomagają utrzymać czystość kodu to ważny element podczas tworzenia aplikacji. Kolejnym niezbędnym narzędziem jest **Postman**, który pomaga testować zapytania do serwerowego API. Aby zobaczyć, czy widok otrzymuje odpowiednie dane od serwera, wymagane jest przetestowanie zapytań, w tym celu zamiast wywoływać ich ręcznie w aplikacji, wykorzystujemy dany program, w którym przybieramy rolę klienta serwera i wysyłamy odpowiednie zapytania, oczekując prawidłowych odpowiedzi. [6] Cały projekt znajdował się podczas produkcji kodu w systemie kontroli wersji Git, w celu ułatwionego dostępu i możliwości powrotu do różnych wersji projektu.[5]

## 5.4 Opis ważniejszych implementacji

### 5.4.1 Bezpieczeństwo aplikacji

#### konfiguracja

Klasa *SecurityConfiguration* przetrzymuje konfiguracje związane z odpowiednim dostępem w zależności od pozwoleń zalogowanego użytkownika. Tak jak widzimy na kawałku kodu 5.1 prawie wszystkie ścieżki są udostępnione dla każdej roli poza ścieżkami *api/admin/\*\** i *management/\*\**, do których dostęp ma jedynie użytkownik z rolą administratora

#### Autoryzacja i tworzenie konta

Tworzenie konta użytkownika polega na wprowadzeniu potrzebnych informacji takich jak login, hasło i email. Po zaakceptowaniu danych przez serwer stworzona zostaje encja użytkownika nieaktywnego. Generowany zostaje klucz, który wysyłany jest w postaci linka na podaną wcześniej skrzynkę pocztową. Po wejściu w odpowiednią ścieżkę przez link z wiadomości użytkownik aktywuje konto i może się zalogować 5.2

```
http
    .authorizeRequests()
    .antMatchers("/api/**").permitAll()
    .antMatchers("/api/admin/**").hasAuthority(AuthoritiesConstants.ADMIN)
    .antMatchers("/api/**").authenticated()
    .antMatchers("/management/**").hasAuthority(AuthoritiesConstants.ADMIN);
```

Kod źródłowy 5.1: Uproszczona metoda określająca dostęp do podanych ścieżek

```
public User registerUser(AdminUserDTO userDTO, String password) {

    User newUser = new User();
    String encryptedPassword = passwordEncoder.encode(password);
    newUser.setLogin(userDTO.getLogin().toLowerCase());
    // new user gets initially a generated password
    newUser.setPassword(encryptedPassword);
    newUser.setFirstName(userDTO.getFirstName());
    newUser.setLastName(userDTO.getLastName());
    if (userDTO.getEmail() != null) {
        newUser.setEmail(userDTO.getEmail().toLowerCase());
    }
    newUser.setImageUrl(userDTO.getImageUrl());
    newUser.setLangKey(userDTO.getLangKey());
    // new user is not active
    newUser.setActivated(false);
    // new user gets registration key
    newUser.setActivationKey(generateRandomAlphanumericString());
    Set<Authority> authorities = new HashSet<>();
    authorityRepository.findById(AuthoritiesConstants.USER)
        .ifPresent(authorities::add);
    newUser.setAuthorities(authorities);
    userRepository.save(newUser);
    log.debug("Created Information for User: {}", newUser);
    return newUser;
}
```

Kod źródłowy 5.2: Metoda rejestrująca użytkownika



### 5.4.2 Tablica przeciągnij i upuść

Główny widok aplikacji to tablica, na której użytkownik jest w stanie za pomocą wydarzenia przyciskiem myszy przeciągnij i upuść (z angielskiego drag and drop), aktualizować zadania do odpowiedniej kolumny. W tym celu użyty został moduł z Angulara o nazwie DragDropModule [4]. Posiada on opisane dyrektywy o określonych funkcjach takich jak:

- `cdkDrag` - nadaje elementom możliwość przeciągnięcia,
- `cdkDropList` - element ten określa listę elementów `cdkDrop`, który odpowiednio je sortuje i jest w stanie wywołać wydarzenie wprowadzenia nowego elementu `cdkDropListDropped`,
- `cdkDropListGroup` - jako rodzic elementów `cdkDropList` ma możliwość połączenia wszystkich elementów `cdkDropList`, aby mogły one współdzielić wszystkie elementy `cdkDrop`, dzięki czemu możliwe jest przeniesienie elementu `cdkDrop` z `cdkDropList A` do `cdkDropList B`

Na załączonym uproszczonym kodzie źródłowym 5.3 kontener `div tasks-container` zawiera wszystkie elementy, które przypisane mają do siebie zadania. Wydarzenie przeciągnięcia zadania na jedną z list wywołuje metodę `drop`, która zapisuje identyfikatory odpowiednio zadania przeciąganego, kolumny z którego zadanie zostało przeciągnięte oraz kolumny, do której przeciągnięto dany element.

```
<div class="board-columns" cdkDropListGroup>
  <div class="board-column" *ngFor="let column of cards">
    <div class="tasks-container"
      cdkDropList
      [cdkDropListData]="column.tasks"
      (cdkDropListDropped)="drop($event)">
      <div class="task" id="{{task.id}}"
        *ngFor="let task of column.tasks"
        (click)="showTask(task)"
        [cdkDragDisabled]="task.completed" cdkDrag>
        <span class="task-child" >{{task.name}}</span>
      </div>
    </div>
  </div>
</div>
```

Kod źródłowy 5.3: uproszczony skrypt odpowiedzialny za widok tablicy



### 5.4.3 Rysowanie wykresów

Ważnym elementem aplikacji jest widok statystyk przedstawianych w postaci wykresów. Do rysowania wykresów użyta została biblioteka Angular Bootstrap 4 Charts. Biblioteka udostępnia nam dyrektywę *mdb-Chart*, która to po podaniu odpowiednich parametrów rysuje wykres na kanwie. [1] Potrzebne parametry to:

- `chartType` typ wykresu, podczas implementacji użyty został typ kołowy `chartType = "pie"` i liniowy `chartType = "line"`
- `datasets` to atrybut odpowiedzialny za dane przedstawione na wykresie. Jest to podpisana tytułem lista wartości.
- `labels` to lista nazewnictwa wartości osi poziomej `x`
- `colors` lista określająca kolory poszczególnych grup wartości
- `options` to lista szczegółowych opcji dostępnych dla wykresu

Kod źródłowy przedstawia inicjalizację wykresu 5.4. Dane pobierane są z bazy danych z trzech nierelacyjnych tabel, które przetrzymują datę i użytkownika wykonującego akcje takie jak przeniesienie zadania z kolumny A do kolumny B, zakończenie zadania czy też przypisanie go do jakiegoś użytkownika. Na podstawie tych tabel inicjalizowana jest lista wartości. Na stronie *statistics* znajdują się cztery typy wykresów:

- kołowy wykres przedstawiający postęp w projekcie dzieląc zadania na zrobione, przypisane i nieprzypisane
- kołowy wykres, który przedstawia liczbę zadań dla każdej kolumny
- liniowy wykres, który dla wymienionego w liście do wybrania zadania pokazuje w ostatnich sześciu miesiącach liczbę przenosin na tablicy oraz liczbę przydziałów do użytkowników
- liniowy wykres dla każdego użytkownika pokazujący liczbę przydziałów dla ostatnich sześciu miesięcy i liczbę zadań zakończonych

```
<div style="display: block; width: 60%;">
  <canvas mdbChart
    [chartType]="chartType"
    [datasets]="chartDatasets"
    [labels]="chartLabels"
    [colors]="chartColors"
    [options]="chartOptions"
    [legend]="true"
    (chartHover)="chartHovered($event)"
    (chartClick)="chartClicked($event)">
  </canvas>
</div>
```

Kod źródłowy 5.4: inicjalizacja wykresu



# Instalacja i wdrożenie

do zrobienia



# Podsumowanie

## 7.1 podsumowanie pracy

Celem niniejszej pracy inżynierskiej było stworzenie aplikacji internetowej wspomagającej zarządzanie zadaniami, w oparciu zastosowaniu metody Kanban. Głównym zadaniem aplikacji było usprawnienie pracy własnej lub pracy w grupie, przy odpowiednim podziale zadań oraz ustalaniu priorytetów. Co ciekawe, aplikacja pozwala zorganizować codzienne obowiązki i efektywnie wykorzystać czas, ale również jest przydatna w przedsiębiorstwach przy organizowaniu i nadzorowaniu pracy zespołu. Zaproponowana aplikacja pozwala na rezygnację z notatek, wszystkie plany i zadania możemy zapisać w jednym miejscu, modyfikować w dowolnym czasie, potrzebujemy jedynie dostęp do urządzenia elektronicznego i Internetu. Popularność istniejących na rynku aplikacji do planowania dowodzi, że istnieje spora grupa użytkowników, która chętnie skorzysta z zaproponowanego rozwiązania.

## 7.2 Plany rozwoju aplikacji

Po zapoznaniu się z opiniami pierwszych użytkowników ważnym kierunkiem rozwoju aplikacji jest ucytelnienie i poprawa interfejsu użytkownika. Po okresie próbnym korzystania z aplikacji przez użytkowników, należy przeprowadzić ankietę lub zapoznać się z opiniami użytkowników na forach. Dzięki analizie zdania użytkowników aplikacja stanie się bardziej intuicyjna. Kolejnym, bardziej przyszłościowym kierunkiem byłaby synchronizacja z kontem Google lub Microsoft. Dzięki temu moglibyśmy rozszerzyć funkcjonalności o posiadanie kalendarza z wyświetlonymi datami z zadaniami jak i informacje odnośnie grafiku pracowniczego. Moglibyśmy również mieć odnośniki przy poszczególnych użytkownikach do ich skrzynki pocztowej email przez co komunikacja stałaby się o wiele łatwiejsza. Jest to oczywiście proces, w którym aplikacja utraci cechę prostej, zwartej aplikacji, ale niestety będzie zależała też od innych, zewnętrznych rozwiązań. Dany pomysł na rozbudowę aplikacji możemy kontynuować za pomocą rozbudowy informacji na temat użytkownika, przypięcie do aplikacji osobistego kanału smtp, aby użytkownicy dostawali przypisane do ich kont skrzynki pocztowe mogli komunikować się między sobą wewnętrznie za pomocą utworzonych służbowych kont. Kalendarz synchronizujący grafik pracowniczy z datami ukończenia poszczególnych zadań też jest funkcjonalnością, która może zostać wdrożona bez obecności zewnętrznych aplikacji.



# Bibliografia

- [1] Angular charts bootstrap. Web pages: <https://mdbootstrap.com/docs/angular/advanced/charts/#docsTabsOverview>.
- [2] Apache maven project documentation. Web pages: <https://maven.apache.org/what-is-maven.html>.
- [3] Diagrams tool. Web pages: <https://www.diagrams.net/>.
- [4] Drag drop event angular overview. Web pages: <https://material.angular.io/cdk/drag-drop/overview>.
- [5] Git version control documentation. Web pages: <https://git-scm.com/doc>.
- [6] Postman documentation. Web pages: <https://www.postman.com/api-documentation-tool/>.
- [7] Spring data overview. Web pages: <https://spring.io/projects/spring-data>.
- [8] Spring initializr. Web pages: <https://start.spring.io>.
- [9] Spring security overview. Web pages: <https://spring.io/projects/spring-security>.
- [10] C. J. Date. *An Introduction to Database Systems*. Pearson, 2003.
- [11] R. T. Fielding, J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, Czerw. 2014.
- [12] B. Mehta. *RESTful Java Patterns and Best Practices*. Packt Publishing, 2014.
- [13] A. Syromiatnikov, D. Weyns. A journey through the land of model-view-design patterns. *2014 IEEE/I-FIP Conference on Software Architecture*, strony 21–30, 2014.





# Spis rysunków

2.1	Prosta tablica <i>Kanban</i> . . . . .	4
2.2	Przykładowa tablica <i>Kanban</i> w aplikacji <i>Trello</i> . . . . .	5
2.3	Przykładowa tablica <i>Kanban</i> w aplikacji <i>LeanKit</i> . . . . .	6
3.1	Komunikacja pomiędzy komponentami we wzorcu MVC . . . . .	7
3.2	Diagram pakietów z których złożona jest cała aplikacja . . . . .	11
3.3	Tabele kontroli wersji . . . . .	12
3.4	Diagram bazy danych . . . . .	13
3.5	Nierelacyjne tabele odpowiedzialne za przetrzymywanie danych potrzebnych do statystyk . . . . .	13
4.1	Widok strony głównej przed zalogowaniem. . . . .	23
4.2	Widok strony głównej po zalogowaniu. . . . .	23
4.3	Widok nagłówka w trzech stanach. . . . .	24
4.4	Widok logowania. . . . .	25
4.5	Widok rejestracji nowego użytkownika. . . . .	25
4.6	Widok zmiany hasła. . . . .	26
4.7	Widok listy zadań. . . . .	27
4.8	Widok listy użytkowników dla administratora. . . . .	27
4.9	Widok statystyk. . . . .	28
4.10	Widok statystyk dla wymienionych z listy zadań. . . . .	28
4.11	Widok dodawania tablicy. . . . .	29
4.12	Widok edycji tablicy. . . . .	29
4.13	Widok tablicy. . . . .	30
4.14	Widok szczegółów zadania. . . . .	31



# Spis kodów źródłowych

3.1	Uproszczona metoda kontrolera z pakietu rest . . . . .	10
5.1	Uproszczona metoda określająca dostęp do podanych ścieżek . . . . .	35
5.2	Metoda rejestrująca użytkownika . . . . .	35
5.3	uproszczony skrypt odpowiedzialny za widok tablicy . . . . .	36
5.4	inicjalizacja wykresu . . . . .	37



# Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.

