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

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 proporcję 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 zarządzaniem 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. Przemyślana organizacja czasu i zarządzenie 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 niefunkcjonalnych 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, jak i 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.

2.1 Problem zarządzania zadaniami

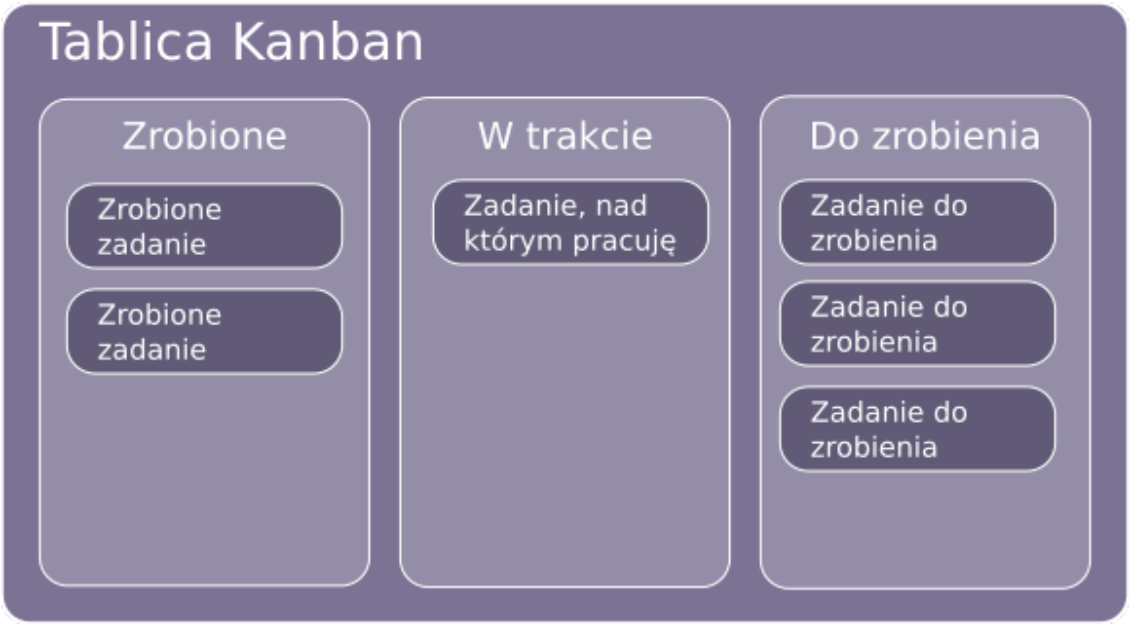
Projekt, jest to przygotowany zbiór aktywności zależnych w jakiś sposób od siebie, zmierzają do wykonania pewnego celu. Może to być naprzykład duże wydarzenie, aplikacja, ulepszenie istniejących systemów czy też sama praca inżynierska. Zazwyczaj taki plan obejmuje duży zakres pracy, rozciągniety w planowanym okresie czasu przez okresloną liczbe osób nazywanych zespołem. Grupa ludzi dostając informacje na temat stanu końcowego, w większosci przypadków nie będą w stanie wyobrazić sobie ukończenia projektu, dlatego potrzebujemy odpowiedniego zarządzania zadaniami. Złożony problem w tym stylu 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. Albowiem, pracownicy zabierają się za zadania, dobierają je z wielkiej listy do zrobienia. Początkowo może się wydawać, że wszystko działa sprawnie. Niestety, w tego typu pracy potrzebna jest wzajemna komunikacja, więc zespół organizuje codziennie dwie godziny gdzie opowiadają 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, niekiedy nawet osób, których zadania nie zależą 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 takiich narzędzi dostęp i mieć możliwosć 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 popularniejszy 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. Taka była geneza powstania metody Kanban.



Rysunek 2.1: Prosta tablica Kanban
Źródło: *https* : *//agilecommander.com/wp − content/uploads/*2018/03/*doc_pl_prosta_tablica_kanban.png*

Metoda Kanban pochodzi z Japonii. Słowo kanban w języku japońskim oznacza szyld, tablicę informacyjną, kartę lub znak. Kluczowym elementem metody jest karta Kanban, która przekazuje informację dotyczącą przeniesienia materiału wewnątrz zakładu lub od zewnętrznego dostawcy. 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-kaban 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 na listę zadań, można określić nad czym pracujemy 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 i 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 programowaniu.

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. Zakładając, że pula zadań jest bardzo duża i nie jest w stanie zmieścić się na tablicy, to zadania, nad którymi aktualnie się pracuje, czy też z nadanymi statusami, będą w stanie wyświetlać się na tablicy w postaci bloków. Bloki można przesuwać po tablicy. Jest to proste zobrazowanie postępu prac nad daną grupą zadań, osoba zarządzająca może zobaczyć podgląd postępów w projekcie, a pracownik, który zrobił postęp w

wybranym zadaniu, z łatwością jest w stanie go udokumentować, wyświetlając go również dla całego zespołu.

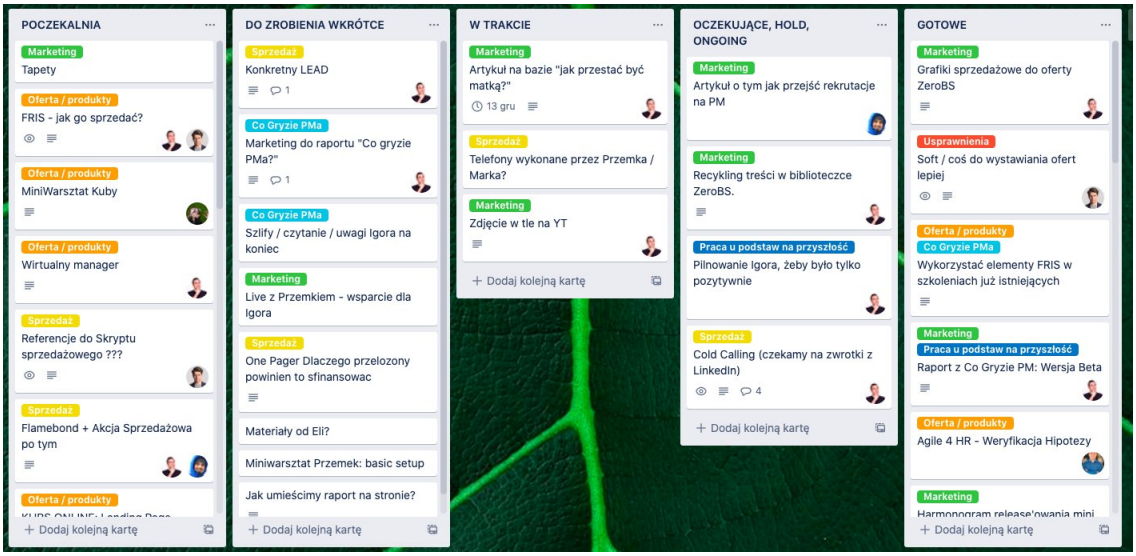
2.4 Porównanie istniejących aplikacji

Istnieje szereg aplikacji o zbliżonej funkcjonalności. Opisane zostaną dwie aplikację: 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, admina i zwykłego użytkownika. Różnią się one jedynie tym, że admin może usuwać i dodawać użytkowników i 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
Źródło: *https : //zerobs.pl/wp − content/uploads/*2019/11/*tablica − operacyjna.jpg*

Kolejną aplikacją jest LeanKit, wybrana została jaka druga aplikacja do porównania, ponieważ tej tablicy nie charakteryzuje prostota. Wręcz przeciwnie, jej atutem są złożone funkcje tworzenia tablicy. Najbardziej charakterystyczną opcją jest skomplikowanie tworzenie karty na tablicy. Kiedy poprzednia aplikacja pozwalała na tablicy utworzyć karty z krótkim tytułem przetrzymujące zadania, które można przenosić z jednej

karty na drugą, ta aplikacja oferuje Tworzenie Karty, podkart karty, podkart podkart, i tak dalej, przez co bardziej skomplikowane do zkategoryzowania 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 wizualizacje 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 bardzo kompleksowe konfiguracje, użytkownik korzystający z programu po raz pierwszy najprawdopodobniej potrzebowałby specjalistycznego szkolenia w celu wykorzystania wszystkich możliwości.

Obie aplikacje opisane zostały, aby ostatecznie wyróżnić różnice w opisywanej w pracy aplikacji. TaskManager również charakteryzuje się prostotą, lecz dopiero podczas korzystania przez zwykłego użytkownika. Proces inicjalizacji tablicy, projektów jest dłuższy, aby potem aplikacja mogła służyć uczestnikom projektu. Pierwszą różnicą jest podział na dwie role, ADMIN, USER. Rola administratora projekt, czyli ADMIN, która posiada pełną kontrole nad dodawaniem użytkowników i zadań, jak i konfiguracji użytkowników, zadań i tablicy. Drugą cechą jest zakładka statystyk, gdzie zobaczyć można ogólne informacje na temat całego projektu, wyników poszczególnych użytkowników, jak i zadań w przeciągu ostatnich sześciu miesięcy. Ostatnią cechą, która może ostatecznie przekonać klienta na aplikację TaskManagaer to utworzenie wewnętrznego projektu niezależnego od innych aplikacji, dlatego wraz z wzrastającymi oczekiwaniami, klient może zlecić rozszerzenie aplikacji na swój sposób, czy to przez podłączenie do kanału komunikacyjnego lub kalendarzu z grafikiem pracowniczym.

Projekt

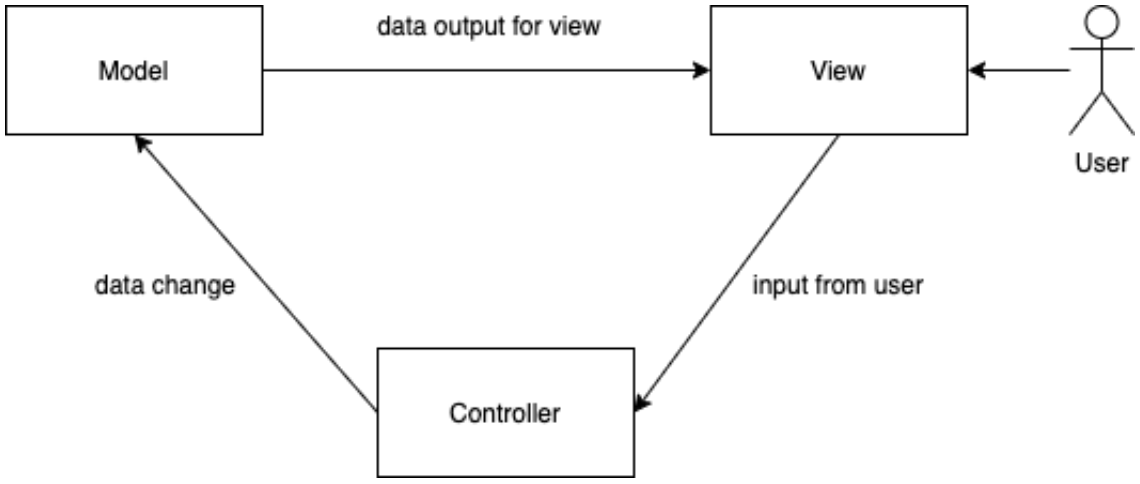
W danym rodziale opisana została aplikacja pod względem technicznym.

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, w momencie w którym 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, zoptymalizowany ułatwiając problemy podczas implementacji, zwiększając czytelność kodu.

3.2 Wzorzec Architektoniczny Model-View-Controller

System zbudowany został w oparciu o model MVC, czyli Model-View-Controller. Jest to podejście architektoniczne, które dzieli aplikacje 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. [8]



Rysunek 3.1: Komunikacja pomiędzy komponentami we wzorcu MVC

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

3.3 Wymagania funkcjonalne i niefunkcjonalne

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

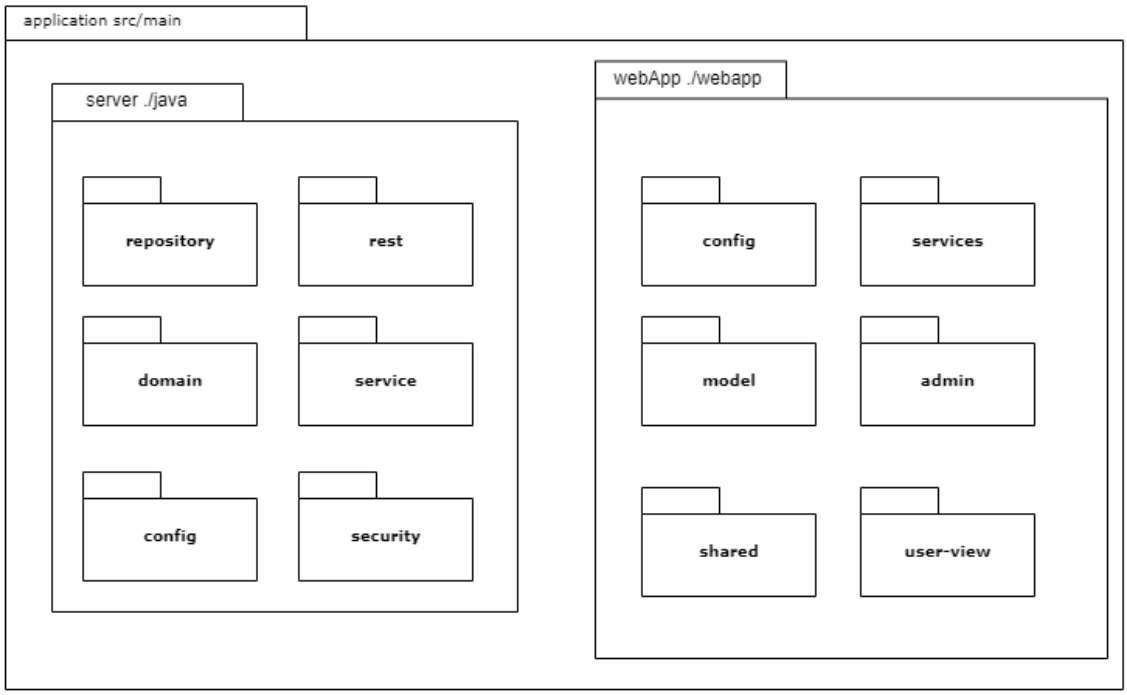
Wymagania funkcjonalne dodatkowe dla administratora

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 (kolumna,która automatycznie zmienia status zadania na skończony), Opcjonalne

Wymagania niefunkcjonalne

1. możliwość użytkowania systemu po godzinnym szkoleniu
2. komunikacja z bazą dancyh nie dłuższa niż 100ms na każde zapytanie
3. aplikacja responsywna, wymagania funkcjonalne możliwe były do wykonania na urządzeniu mobilnym

3.4 Diagram pakietów



Rysunek 3.2: Diagram pakietów z których złożona jest cała aplikacja

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 annotacją @Repository. Są to klasy, które udostępniają gotowe metody do wyciągania potrzebnych danych z bazy

domain definiuje model danych. Są to Tabele i ich wszystkie atrybuty przepisane na klasę z której korzystają klasy z pakietu repository

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 annotacją @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 transfer danych sposobem REST, czyli udostępnia metody korzystające z serwisów na podane URI. Przykładowo aplikacja webowa wysyła na serwer zapytanie HTTP POST na URI *www.taskManager.com/task*. Serwer odczytuje zapytanie i przygotowywuje 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 zwrotną wiadomość. Zobacz przykład 3.1.

security zawiera klasy odpowiedzialne za autentykację połączenia, zanim jakiekolwiek zapytanie trafi do metod z pakietu rest, najpierw serwer oczekuje na odpowiedź ze strony klienta odnośnie sukcesu autoryzacji i pozwoleń jakie zalogowany użytkownik posiada.

```
@PostMapping("/task")
public ResponseEntity<Task> createTask(@Valid @RequestBody TaskDTO taskDTO){
        log.debug("REST request to save Task : {}", taskDTO);
        if (taskDTO.getId() != null) {
                (...)
               }
        return ResponseEntity
        .created(new URI("/task/" + task.getId()))
        .headers(HeaderUtil.createAlert(applicationName))
        .body(task);
        }
```

Kod źródłowy 3.1: Przykładowa metoda controllera 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 potrzebych 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.

3.5 Komunikacja klient - serwer

Zbiór założeń, który definiuje komunikację pomiędzy systemami nazywamy API (z angielskiego Application Programming Interface, czyli interfejs programowania aplikacji). Komunikacja odbywa sie za pomocą protokołów HTTP. Protokół HTTP jest to zestaw reguł odnośnie komunikacji polegający na wysyłaniu żądania i oczekiwaniu na odpowiedź. Klient wysyła zapytania, oczekuje na odpowiedź, a serwer oczekuje na zapytania odsyłając odpowiedź.
Protokół ten składa się z:

- Metoda protokołu i wersja
- Nagłówka (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 (naprzykład 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

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. Klasy w pakiecie *rest* serwera udostępniają metody na poszczególne ścieżki, w których zastosowana jest logika i zwracane zostają odpowiednie dane, wiadomość OK ze statusem 200 [3.1].

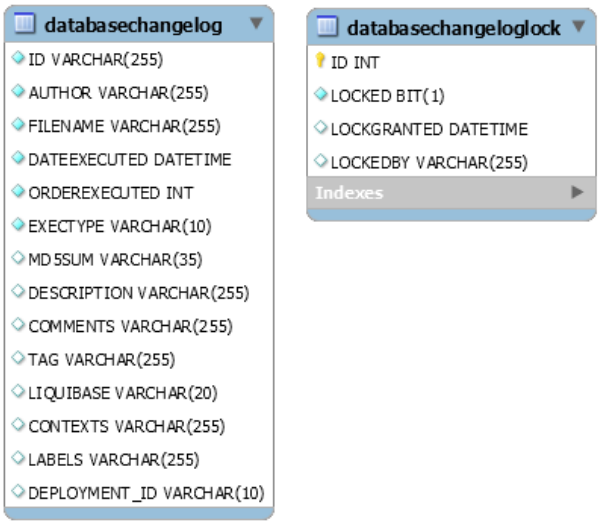
3.6 Projekt bazy danych

Sekcja ta w pełni skupiona jest na budowie bazy danych. Jako baza danych wykorzystana została baza dystrubucji 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 podpięta jest biblioteka zwana Liquibase, która przechowuje wszystkie skrypty związane z tworzeniem bazy danych, modyfikacją tabel lub danych w formacie XML. Liquibase pomaga wprowadzać zmiany na istniejącej bazie danych przy zachowaniu poprzedniego stanu, zostawiając po udanej modyfikacji historię zmian. Bilbioteka liquibase dodaje dwie tabele 3.4, które dotyczą powyższej listy plików

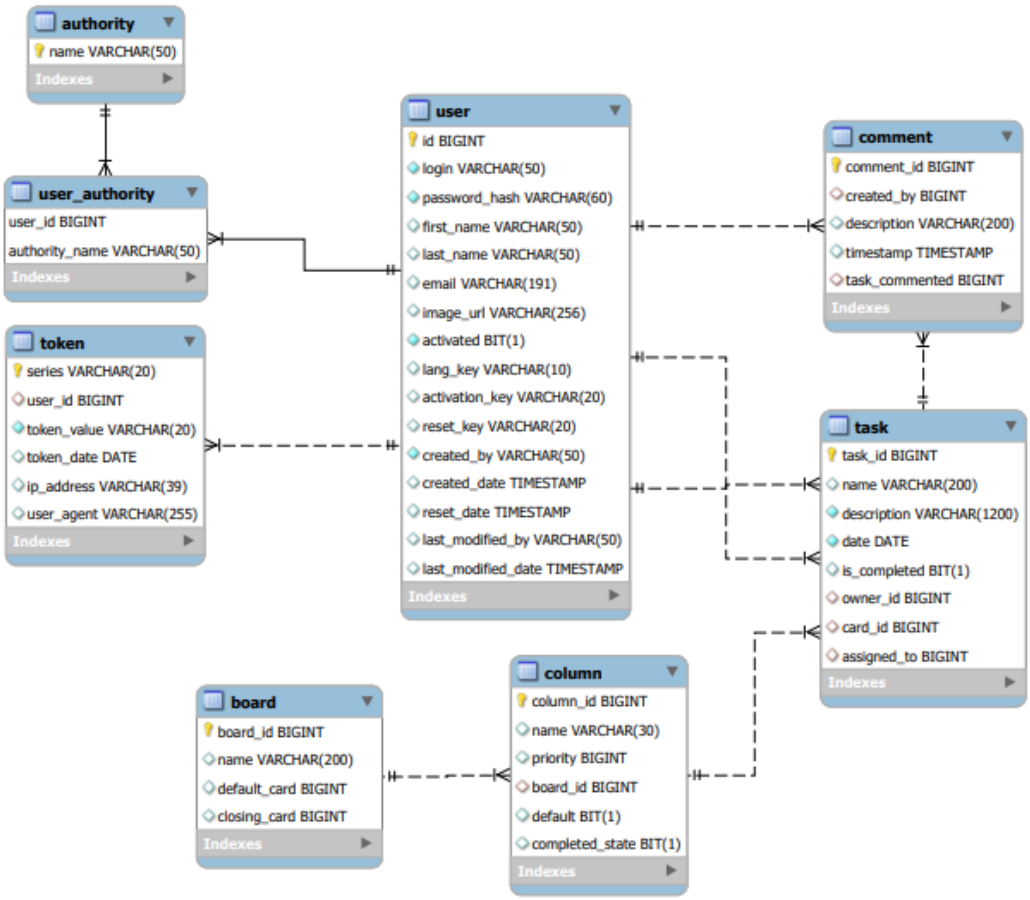
```
<include file="liquibase/changelog/00000000000000_initial_schema.xml" relativeToChangelogFile="false"/>
<include file="liquibase/changelog/00000000000001_addTaskTable_schema.xml" relativeToChangelogFile="false"/>
<include file="liquibase/changelog/00000000000002_addBoardColumnTable_schema.xml" relativeToChangelogFile="false"/>
<include file="liquibase/changelog/00000000000003_addHistoryTables_schema.xml" relativeToChangelogFile="false"/>
```

Rysunek 3.3: Lista plików tworzących bazę danych

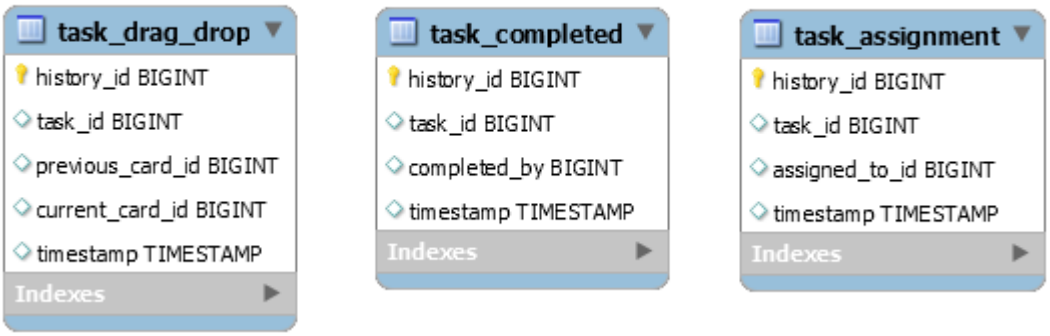
3.3. Tabela zapisuje nazwę pliku i jej sumę kontrolną, więc zapobiega to zmianom na plikach inicjalizujących bazę danych. Trzeba stworzyć następny plik i pozsotawić po sobie ślad. Jest to bardzo pomocna rzecz, służy jako kontrola wersji bazy danych, aby w każdym momencie można było wrócić do wcześniejszych struktur bazy danych. Pierwszym rodzajem danych jest *Tablica* (board), która przetrzymuje informacje o tablicy. Kolejnym typem jest *Kolumna* (column). Ważnym typem danych jest *Użytkownik* (user), to jedna z najbardziej skomplikowanych tabel w całej bazie. Tabela z użytkownikami przetrzmuje informacje dotyczące korzystania z aplikacji i te pozwalające autoryzować sesję po stronie klienta. Posiada również relacje z tabelą authority, która znowu odpowiedzialna jest za nadawnianie odpowiednich uprawnień do korzystania z aplikacji. Ostatnimi danymi są *Zadanie* (task) i *Komentarz* (comment). Wszystkie relacje zostały przedstawione na diagramie 3.5. 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) 3.6.



Rysunek 3.4: Tabele kontroli wersji



Rysunek 3.5: Diagram bazy danych



Rysunek 3.6: Nierelacyjne tabele odpowiedzialne za przetrzymywanie danych potrzebnych do statystyk

3.6.1 Normalizacja bazy danych

3.5

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:	1. klient wchodzi na panel rejestracji za pomocą linka ”Create Account” lub używając przycisku na nagłówku strony 2. po wypełnieniu przez klienta formularzu rejestracyjnego zachodzi proces walidacji danych 3. gdy walidacja przeszła pomyślnie: system wyświetla komunikat o sukcesie rejestracji i informuje,że potrzebna jest jeszcze aktywacja konta (a) System zapisuje dane na bazie, hasło zostaje zakodowane. (b) System generuje losowy 8-cyfrowy klucz aktywacyjny dla użytkownika i wysyła na podany w formularzu e-mail link aktywacyjny. (c) Klient aktywuje konto za pomocą linka, który otrzymał na skrzynkę pocztową e-mail. 4. gdy walidacja zauważyła błędy w formularzu: wyświetla się komunikat błędu, zaznaczone zostają błędne pola

Tabela 3.1: Przypadek użycia - rejestracja nowego użytkownika

Logowaniea	
Aktorzy:	klient , serwer autoryzacyjny, system prezentujący
Warunki wstępne:	sesja klienta nie jest autoryzowana, klient posiada aktywne konto
Scenariusz:	1. klient wchodzi na panel logowania za pomocą linka ”Log int” lub używając przycisku na nagłówku strony 2. klient wprowadza prawidłowe dane, nie zaznacza pola 'remember me' (a) system sprawdza czy istnieje użytkownik o podanych informacjach kwalifikujących (b) system autoryzacyjny odnajduje w bazie użytkownika, (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 3. klient wprowadza prawidłowe dane, zaznacza pole 'remember me' (a) system autoryzacyjny odnajduje w bazie użytkownika, (b) system autoryzacyjny tworzy 20-cyfrowy unikalny kod zwany tokenem i umieszcza go dla danego użytkownika (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 4. klient wprowadza złe dane (a) system nie odnajduje użytkownika o podanych informacjach kwalifikujących , komunikuje się z systemem prezentującym o braku podanego użytkownika (b) system prezentujący wyświetla komunikat o prawdopodobnie źle wpisanym loginie lub haśle.

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:	1. Użytkownik przechodzi do panelu tworzenia nowego zadania po wciśnięciu przycisku ”Create Task” zarówno na stronie ”Board” jak i ”Tasks List” 2. pojawia się formularz z informacjami do podania na temat zadania, niektóre niezbędne do stworzenia zadania, niektóre opcjonalne 3. po ukończeniu wypełniania formularza i kliknięciu przycisku ”Save” system waliduje poprawność otrzymanych danych: (a) gdy dane odnośnie zadania zgadzają się, użytkownik przekierowywany jest na stronę, z której otworzył panel tworzenia zadania (b) gdy dane odnośnie zadania nie zgadzają się, panel prezenetujący wyświetla pola z błędnymi informacjami (c) użytkownik może poprawić dane i próbować stworzyć zadanie ponownie

Tabela 3.3: Przypadek użycia - Dodawanie Zadania

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:	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” 2. Użytkownik może również przejść do panelu uaktualniania za pomocą kliknięcia w kontener wizualizujący zadanie na tablicy kanban w panelu ”Board” (a) system prezentujący wyświetla wszystkie informacje na temat istniejącego zadania oraz dwa przyciski ” Go Back” ”Save” i. użytkownik używa przycisku Save, informacje zawarte w formularzu wysyłane są do serwera w celu zaktualizowania zadania ii. użytkownik po obejrzeniu informacji na temat zadania może za pomocą przycisku ” go Back” wrócić do wcześniej odwiedzanego panelu

Tabela 3.4: 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:	1. użytkownik jest na panelu "Tasks List" 2. wyświetlana jest przez system prezentujący lista zadań na którym widoczne są najważniejsze informacje 3. system prezentujący otrzymuje listę ograniczoną parametrami z serwera obsługującego bazę danych 4. użytkownik klikając na nazwę kolumny jest w stanie sortować ją rosnąco lub malejąco 5. ostatnią kolumną jest przycisk "Go to Task", dzięki któremu możemy przejść do panelu danego taska

Tabela 3.5: 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", rolą użytkownika jest "ROLE_USER" , czyli uprawnienia zwykłego użytkownika
Scenariusz:	1. użytkownik jest na panelu "Users List" 2. wyświetlana jest przez system prezentujący lista zadań na którym widoczne są najważniejsze informacje 3. system prezentujący otrzymuje listę ograniczoną parametrami z serwera obsługującego bazę danych 4. użytkownik klikając na nazwę kolumny jest w stanie sortować ją rosnąco lub malejąco 5. ostatnią kolumną jest przycisk "Go to Task", dzięki któremu możemy przejść do panelu danego taska

Tabela 3.6: Przypadek użycia - Przeglądanie listy użytkowników

Przeglądanie panelu ze statystykami	
Akotrzy:	użytkownik, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana i wypełniona są tabele związane z historią
Scenariusz:	1. użytkownik jest na panelu ” Charts” 2. wyświetlane są przez system prezentujący informacje na temat projektu i 3 wykresy związane ogólnie ze statystykami projektu 3. niżej znajdują się dwie listy do wyboru, lista użytkowników i zadań. 4. użytkownik wybiera odpowiednich użytkowników na liście, aby wyświetlić wykresy związane z ich wynikami w ostatnich 6 miesiącach 5. użytkownik wybiera odpowiednie zadania na liście, po czym wyświetalne zostają wykresy dla każdego odznaczonego zadania.

Tabela 3.7: 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:	1. użytkownik jest na panelu ”Board” widizi wszystkie kolumny i zadania w nich 2. użytkownik za pomocą wydarzenia przyciśnij i upuść przeciąga zadanie z kolumny A do kolumny B 3. system odpowiednio zapisuje dane wydarzenie, sprawdzając czy kolumna B nie jest równocześnie kolumną zamykającą zadania 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.

Tabela 3.8: Przypadek użycia - Aktualizowanie zadania za pomocą tablicy Kanban

Konfiguracja tablicy	
Akotrzy:	administrator, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana jako administrator
Scenariusz:	1. użytkownik wchodzi na panelu ”Manage Board” z poziomu nagłówka strony 2. system wyświetla formularz związany z informacjami tablicy, czyli nazwy i identyfikatory kolumny domyślnej i zamykającej, która ustawiaa zadanie na ukończone. 3. użytkownik zmienia odpowiednie parametry, a końcowe informacje wysyła do serwera po zaakceptowaniu przyciskiem ”save” 4. serwer weryfikuje poprawność danych i aktualizuje potrzebne informacje

Tabela 3.9: Przypadek użycia - Konfiguracja tablicy

Edycja użytkowników przez administratora	
Akotrzy:	administrator, serwer
Warunki wstępne:	sesja użytkownika jest autoryzowana jako administrator
Scenariusz:	1. użytkownik wchodzi na panelu ”Manage users” z poziomu nagłówka strony 2. system wyświetla formularz związany z informacjami użytkowników, informacji jest więcej niż na liście udostępnionej dla użytkownika 3. administrator jest w stanie usunąć, stworzyć nowe konto, ale też dezaktywować istniejące. 4. gdy administrator kliknie ”create user” musi wypełnić wyskakujący formularz, po wypełnieniu serwer wysyła na odpowiedni adres email informacje o utworzeniu konta przez użytkownika

Tabela 3.10: Przypadek użycia - edycja użytkowników przez administratora

Interfejs użytkownika

4.1 Widok Logowania

4.2 Widok Listy Zadań

4.3 Widok Admina

4.4 Widok Statystyk

4.5 Widok Tablicy

4.6 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.

5.1.1 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

5.1.2 View - Widok

Widok zaimplementowany jest przy pomocy frameworku Angular korzystającego z HTML oraz TypeScript. Podstawowym obiektem w framweroku jest component, 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

5.1.3 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 podatynymi 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

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. *Spring Data JPA* ułatwia dostęp do bazy danych udostępniając potrzebne metody wyciągające odpowiednie informacje z bazy danych.[5] [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 [6] 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 kompliowanie zaawansowanego projektu napisanego w języku Java [1].

5.3 Narzędzia pomocnicze i środowisko programistyczne

Program napisany został przy pomocy **Inteliij 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 REST. Aby zobaczyć czy widok otrzymuje odpowiednie dane od serwera, wymagane jest przetestowanie zapytań, w tym celu zamiast wywoływać ich ręcznie w aplikacji, wykorzystujemy program Postman, w którym przybieramy rolę klienta serwera i wysyłamy zapytanie, oczekując odpowiedzi. [4] Cały projekt znajdował się podczas produkcji kodu w systemie kontroli wersji Git, w celu ułatiwionego dostępu i możliwości powrotu do różnych wersji projektu.[3]

5.4 Opis ważniejszych implementacji

5.4.1 Autoryzacja

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

```
            http
              .authorizeRequests()
.antMatchers("/api/board").permitAll()
.antMatchers("/api/task/complete").permitAll()
.antMatchers("/api/stats/generalcount").permitAll()
.antMatchers("/api/task/comment").permitAll()
.antMatchers("/api/board/card").permitAll()
.antMatchers("/api/board/task").permitAll()
.antMatchers("/api/board/tasks").permitAll()
.antMatchers("/api/authenticate").permitAll()
.antMatchers("/api/task").permitAll()
.antMatchers("/api/register").permitAll()
.antMatchers("/api/activate").permitAll()
.antMatchers("/api/account/reset-password/init").permitAll()
.antMatchers("/api/account/reset-password/finish").permitAll()
.antMatchers("/api/admin/**").hasAuthority(AuthoritiesConstants.ADMIN)
.antMatchers("/api/**").authenticated()
.antMatchers("/management/health").permitAll()
.antMatchers("/management/health/**").permitAll()
.antMatchers("/management/info").permitAll()
.antMatchers("/management/prometheus").permitAll()
.antMatchers("/management/**").hasAuthority(AuthoritiesConstants.ADMIN);
```

Kod źródłowy 5.1: Metoda określająca dostęp do podanych ścieżek dla zdefiniowanych pozwoleń

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. Generwoany zostaje klucz, który wysyłany jest w postaci linka na podaną wcześniej skrzynkę pocztową. Po wejściu w odpowiednią ścieżkę przez link z wiadomośći użytkownik aktywuje konto i może się zalogować 5.2

```
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 ang. drag and drop), aktualizować zadania do odpowiedniej kolumny. W tym celu użyty został moduł z Angulara o nazwie DragDropModule [2]. Posiada on opisane dyrektywy o określonych funkcjach takich jak:

- cdkDrag - nadaje elementom możliwość przeciągniecia,
- 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* łączy wszystkie elementy wewnątrz, pozwalając im na siebie współoddziaływać, naprzykład przenieść element *cdkDrop* z *cdkDropList* A do *cdkDropList* B

Element *div tasks-container* przetrzymuje wszystkie elementy, które przypisane mają do siebie zadania, a pod wydarzenie przecięgniecia zadania na jedną z list wywoływana jest metoda *drop*, która zapisuje identyfikatory odpowiednio zadania przeciąganego, kolumny z którego zadanie zostało przeciągniete oraz kolumny do której przeciągnieto dany 5.3.

```
<div class="board-columns" cdkDropListGroup>
  <div class="board-column" *ngFor="let column of cards">
    <div class="column-title"> {{ column.name }}</div>
    <div class="tasks-container"
       cdkDropList
       id = "{{column.id}}"
       [cdkDropListData]="column.tasks"
       (cdkDropListDropped)="drop($event)">
       <div class="task" id="{{task.id}}"
         [ngStyle]="{'background-color': task.completed ? '#90EE90':'white'}"
         *ngFor="let task of column.tasks"
         (click)="showTask(task)"
         [cdkDragDisabled]="task.completed" cdkDrag>
          <span class="task-child" >{{task.name}}</span>
          <div class="verticalLine">
           Assigned to
           <span *ngIf="task.assignedTo?.email == this.user.email"> you </span>:
           <span *ngIf=task.assignedTo class="assignedTo"
             [ngStyle]="{'color': task.assignedTo.email == this.user.email ?
                                  'red':'black'}" >
                  {{task.assignedTo.email}}
                 </span>
            <span *ngIf="task.completed"
                  style="margin-left:1%;
                  color: cadetblue; ">
                        <fa-icon icon="check"></fa-icon>  Task Completed
            </span>
       </div>
     </div>
  </div>
</div>
```

Kod źródłowy 5.3: skrypt odpowiedzialny za widok tablicy

5.4.3 Rysowanie wykresów

Instalacja i wdrożenie

W tym rozdziale należy omówić zawartość pakietu instalacyjnego oraz założenia co do środowiska, w którym realizowany system będzie instalowany. Należy przedstawić procedurę instalacji i wdrożenia systemu. Czynności instalacyjne powinny być szczegółowo rozpisane na kroki. Procedura wdrożenia powinna obejmować konfigurację platformy sprzętowej, OS (np. konfiguracje niezbędnych sterowników) oraz konfigurację wdrażanego systemu, m.in. tworzenia niezbędnych kont użytkowników. Procedura instalacji powinna prowadzić od stanu, w którym nie są zainstalowane żadne składniki systemu, do stanu w którym system jest gotowy do pracy i oczekuje na akcje typowego użytkownika.

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 uczytelnienie 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 funcjonalności o posiadanie kalendarzu 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 kontynuuować 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] Apache maven project documentation. Web pages: https://maven.apache.org/what-is-maven.html.

[2] Drag drop event angular overview. Web pages: https://material.angular.io/cdk/drag-drop/overview.

[3] Git version control documentation. Web pages: https://git-scm.com/doc.

[4] Postman documentation. Web pages: https://www.postman.com/api-documentation-tool/.

[5] Spring data overview. Web pages: https://spring.io/projects/spring-data.

[6] Spring initializr. Web pages: https://start.spring.io.

[7] Spring security overview. Web pages: https://spring.io/projects/spring-security.

[8] A. Syromiatnikov, D. Weyns. A journey through the land of model-view-design patterns. *2014 IEEE/IFIP Conference on Software Architecture*, strony 21–30, 2014.

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.