

**Wydział Inżynierii Mechanicznej i Robotyki**

Katedra Robotyki i Mechatroniki

Praca dyplomowa

*Opracowanie i realizacja webowego systemu zarządzania projektami*

*Development and implementation of a web-based project management system*

Autor: *Bartosz Stelmach*

Kierunek studiów: Automatyka Przemysłowa i Robotyka

Opiekun pracy: *dr inż. Lucjan Miękina*

Kraków, 2024

Spis treści

[**Wstęp** 2](#_Toc176439133)

[**1.** **Teoretyczne podstawy zarządzania projektami software’owymi** 4](#_Toc176439134)

[**1.1.** **Narzędzia automatyzacji i ciągłej integracji w projektach** 6](#_Toc176439135)

[**1.1.1.** **Czym jest pipeline CI/CD** 6](#_Toc176439136)

[**1.1.2.** **Narzędzia wspierające automatyzację pipeline CI/CD** 7](#_Toc176439137)

[**1.1.3.** **Przyszłość CI/CD i automatyzacji w zarządzaniu projektami** 7](#_Toc176439138)

[**2.** **Przegląd używanych technologii** 8](#_Toc176439139)

[**2.1.** **Git jako system kontroli wersji** 8](#_Toc176439140)

[**2.2.** **Maven jako narzędzie do zarządzania i budowy projektu** 9](#_Toc176439141)

[**2.3.** **Spring Boot jako framework do tworzenia aplikacji** 10](#_Toc176439142)

[**2.4.** **Relacyjne bazy danych i ich zastosowanie w projektach** 11](#_Toc176439143)

[**2.5.** **Jenkins dla ciągłej integracji i dostarczania (CI/CD)** 12](#_Toc176439144)

[**2.6.** **Postman jako narzędzie do testowania i dokumentacji interfejsów API** 13](#_Toc176439145)

[**3.** **Analiza wymagań** 15](#_Toc176439146)

[**3.1.** **Wymagania funkcjonalne systemu zarządzania projektami** 15](#_Toc176439147)

[**3.2.** **Wymagania niefunkcjonalne systemu zarządzania projektami** 15](#_Toc176439148)

[**4.** **Architektura i projekt systemu** 17](#_Toc176439149)

[**4.2.** **Wybór i zastosowanie PostreSQL jako systemu bazy danych** 31](#_Toc176439150)

[**5.** **Implementacja** 32](#_Toc176439151)

[**5.1.** **Implementacja kluczowych funkcjonalności systemu** 32](#_Toc176439152)

[**5.2.** **Umieszczenie kodu w zdalnym repozytorium** 37](#_Toc176439153)

[**5.3.** **Konfiguracja Jenkinsa do automatycznego testowania i deploymentu** 38](#_Toc176439154)

[**5.4.** **Użycie Mavena do zarządzania zależnoścami i budowy projektu** 41](#_Toc176439155)

[**6.** **Testowanie** 43](#_Toc176439156)

[**6.1.** **Planowanie testów (Integracyjne oraz Jednostkowe)** 44](#_Toc176439157)

[**6.2.** **Automatyzacja testów z użyciem Jenkinsa** 46](#_Toc176439158)

[**Podsumowanie** 48](#_Toc176439159)

[**Bibliografia** 50](#_Toc176439160)

# **Wstęp**

Efektywne zarządzanie projektami jest fundamentem w erze nowoczesnych technologii. W miarę jak rozwój oprogramowania staje się coraz bardziej złożony, kluczową rolę zaczynają odgrywać aplikacje, które umożliwiają stałą kontrolę i zarządzanie procesami. Aplikacja zaproponowana w ramach tej pracy dyplomowej koncentruje się na tworzeniu i wdrażaniu internetowego zarządzania projektami, którego celem jest usprawnienie organizacji i zarządzanie projektami w bardziej efektywny sposób.

Zakres tej pracy obejmuje analizę teoretycznych podstaw zarządzania projektami software’owymi, która służą do zrozumienia podstawowych zasad, metodologii oraz praktyk niezbędnych do powodzenia w danym projekcie. Pierwszy dział koncentruje się na ważnych aspektach, omawiając kluczowe pojęcia, takie jak ciągła integracja i ciągłe dostarczanie (CI/CD), systemy kontroli wersji oraz role narzędzi w automatyzacji.

Po zapoznaniu się z podstawami teoretycznymi, praca analizuję konkretne technologie wykorzystane do stworzenia webowego systemu zarządzania projektami. Drugi rozdział zawiera szczegółowy przegląd wykorzystywanych narzędzi i frameworków, takich jak Git do kontroli wersji, Maven do automatyzacji kompilacji projektów oraz Spring Boot jako podstawowy framework do tworzenia aplikacji w języku Java. Dodatkowo, rozdział ten analizuje wykorzystanie relacyjnych baz danych, Jenkinsa do procesów CI/CD oraz Postmana do testowania i dokumentowania API (Application Programming Interface). Dzięki temu podkreślamy, w jaki sposób dane technologie przyczyniają się do ogólnej funkcjonalności systemu.

Trzeci rozdział poświęcony jest analizie wymagań systemowych. Przedstawiono w nim wymagania zarówno funkcjonalne, czyli takie dotyczące jakości działania systemu oraz niefunkcjonalne które określają, co system powinien robić, aby spełniać potrzeby biznesowe.

W rozdziale czwartym przedstawiono architekturę i projekt systemu, kładąc nacisk na wykorzystanie Spring Boota do tworzenia aplikacji. W praktyczny sposób został pokazane jak wykorzystać narzędzia do automatyzacji procesów i zwiększenie wydajności systemu. Rozdział również zawiera szczegółowe omówienie schematu bazy danych i modeli danych, które są niezbędne do efektywnego zarządzania.

Piąty rozdział zagłębia się w proces implementacji, szczegółowo opisując proces rozwoju, integrację kluczowych funkcjonalności oraz konfigurację różnych narzędzi. Omówione również zostało praktyczne wykorzystanie Mavena do zarządzania zależnościami i budowania projektu.

Testowanie jest kluczowym aspektem dla każdego systemu oprogramowania, dlatego rozdział szósty skupia się na strategiach testowania projektu. Przedstawiono tam planowanie i wykonania różnych testów, w tym testów jednostkowych i integracyjnych, a także wykorzystanie Jenkinsa do automatyzacji procesu testowania.

Praca kończy się kompleksowym podsumowaniem wyników, odzwierciedlającym różne wyzwania napotkane w trakcje projektu i strategii zastosowane w celu ich przezwyciężenia. Wnioski wskazują również różne kierunki rozwoju aplikacji obejmujące rozbudowanie interfejsu użytkownika dające mu większe możliwości oraz ewentualnie większą skalowalność w kontekście używania aplikacji w większych korporacjach.

1. **Teoretyczne podstawy zarządzania projektami software’owymi**

Zarządzanie projektami software’owymi to złożona sztuka, która wymaga harmonijnego połączenia relacji międzyludzkich, zaawansowanych technologii oraz zarządzania budżetem w ramach specyficznych potrzeb projektu. Celem jest bowiem osiągnięcie maksymalnej efektywności oraz ułatwienie realizacji działań.

Jednym z aktualnie najtrudniejszych zadań jest spełnienie oczekiwań każdej ze stron, w tym zespołów developerskich, zespołów DevOps, zarządu, klientów oraz użytkowników końcowych. Często przez przymus połączenia oczekiwań rodzą się napięcia między potrzebą wprowadzenia nowych funkcjonalności do aplikacji a ograniczeniami budżetowymi, czy między innowacjami a koniecznością refaktoryzacji istniejącego kodu.

Przykłady konfliktów, które mogą się zrodzić podczas dyskusji wśród różnych grup zaangażowanych w projekt, to:

* **Developerzy aplikacji** mogą odczuwać frustrację, ponieważ zdają sobie sprawę, iż jakość kodu bazy danych mogłaby być lepsza.
* **Zespoły DevOps** są sfrustrowani, ponieważ ograniczony budżet na bezpieczeństwo może narazić na większe ryzyko ataków.
* **Zespoły projektowe** odczuwają presję, ponieważ ich projekty mogą zostać odrzucane przez programistów ze względu na zbyt ambitne plany.
* **Menedżerowie** są niezadowoleni, ponieważ nie widzą oczekiwanych zwrotów z inwestycji.
* **Klienci** są rozczarowani, ponieważ obiecane funkcje nie są dostarczane na czas.

Te wszystkie problemy, które zostały tutaj wyżej wymienione stanowią sedno sprawy związanych z zarządzeniem projektami. W odpowiedzi na te wyzwania, teoria W, znana również jako teoria zarządzania projektami, oferuje zestaw zasad i wytycznych, które pomagają w skutecznym radzeniu sobie z trudnościami pojawiającymi się podczas procesu rozwoju projektu.

Teoria W została opracowana przez Barry’ego Boehma, jednego z najbardziej wpływowych badaczy w dziedzinie inżynierii oprogramowania. Jego prace są szeroko cytowane na międzynarodowych konferencjach czy spotkania oraz w prestiżowych czasopismach poświęconych zarządzaniu projektami. Ma on również na swoim koncie liczne książki dotyczące ekonomicznych aspektów zarządzania projektami oprogramowania, które były publikowane przez różne wydawnictwa.

Teoria W postała jako odpowiedź na potrzebę wypracowania zasad i wytycznych, które miałyby pomóc menadżerom projektów w radzeniu sobie z trudnościami związanymi z zarządzaniem oprogramowaniem. Teoria W zakłada, że głównym zadaniem menedżera projektu jest zagwarantowanie faktu, iż wszystkie osoby zaangażowane w projekt będą „wygranymi” w procesie tworzenia oprogramowania.

Podczas gdy niektóre metody zarządzania projektami skupiają się głównie na potrzebach sponsorów i użytkowników, a podejścia do zarządzania produktami biorą pod uwagę cały cykl życia produktu, Teoria W przyjmuję zupełnie inne podejście. Fundamentem tej teorii są koncepcje naukowego zarządzania Fredrica Taylora, który pokazał, że najskuteczniejsze osiąganie celów polega na precyzyjnych analizach ruchów, organizacji zadań oraz na wyznaczaniu konkretnych ram czasowych.

Zasada przewodnia mówi, ze menadżerowie projektów mogą odnieść sukces tylko wtedy, gdy zapewniają, że wszyscy uczestnicy procesu tworzenia oprogramowania będą „wygranymi” niezależnie od ich statusu.

Chociaż Teoria W promuje sytuacje „win – win”, może wydawać się, że jej osiągniecie jest trudne. W wielu przypadkach warunki są często postrzegane jako „zero – sum”, gdzie istnieje wygrany i przegrany. Każda osoba zaangażowana w proces może coś stracić, aby inna mogła zyskać. Jednak według Teorii W zakłada się, że nie musi dochodzić do sytuacji, w której jedna strona traci, aby inna mogła wygrać – wszystkie strony powinny odnosić korzyści. Przykładowo, jeżeli produkt jest tworzony szybko i niskim kosztem, dostawca może na tym zyskać. Niemniej jednak, taka sytuacja może nie zadowolić osób odpowiedzialnych za utrzymanie i użytkowników, ponieważ produkt może wymagać częstych napraw i wiązać się z wyższymi kosztami utrzymania, co jest niekorzystne dla użytkownika.

Dlatego ważnym jest, aby menadżerowie projektów mogli osiągać wyniki „win – win” muszą uwzględniać interes wszystkich ludzi zamieszanych w dany projekt. Przykładowo, podczas tworzenia oprogramowania, ustalenia dotyczące podziału zysków mogą zmotywować firmę do stworzenia innowacyjnego i wysokiej jakości produktu. W rezultacie zwiększa się sprzedaż, a zyski rosną dla obu stron.

Zgodnie z teorią Boehma, menedżerowie projektów powinni wiedzieć, jak każda ze stron chce osiągnąć swoje cele. Ważne jest, aby zidentyfikować kluczowe osoby zaangażowane w projekt, przeprowadzając analizę klienta. Badania pokazują, że projekty oprogramowania często kończą się niepowodzeniem, ponieważ nie uwzględniono kluczowych osób podczas planowania strategii "win-win". Ważne jest także, aby zrozumieć, czego naprawdę chcą poszczególne strony, zamiast opierać się na przypuszczeniach. Dzięki temu menedżer projektu będzie mógł opracować rozwiązania, które spełnią oczekiwania wszystkich zaangażowanych.

Menedżerowie projektów muszą ustalać realistyczne oczekiwania. Jak już wspomniano, menedżer projektu powinien działać jako osoba wyznaczająca cele. Problemy często pojawiają się, gdy niektóre osoby nie rozumieją, jak trudne lub łatwe jest osiągnięcie danego celu. W rezultacie stawia się nierealistyczne cele i oczekiwania, często bazując na błędnych założeniach. Boehm sugeruje, że ważne jest zebranie wszystkich ludzi zaangażowanych w projekt, wysłuchanie ich oczekiwań i zrozumienie różnic między nimi. Należy spojrzeć na problemy z różnych perspektyw i pomóc wszystkim zrozumieć, jakie cele są realistyczne. Proces ten powinien uwzględniać doświadczenia wszystkich stron oraz dobrze dobrane modele działania. Kluczowe jest także rozwiązywanie konfliktów i budowanie współpracy w zespole.

Boehm uważa, że kluczowe jest rozbicie opcji na mniejsze części i zidentyfikowanie najlepszych sposobów na osiągnięcie sytuacji "win-win" dla wszystkich uczestników. Następnie należy dostosować wybrane rozwiązania tak, aby były korzystne dla wszystkich stron. Rolą menedżera projektu oraz innych specjalistów jest zapewnienie wszystkim stronom odpowiedniego wsparcia, aby mogły one przeprowadzić analizę i znaleźć rozwiązanie "win-win". To wsparcie może obejmować szkolenia dla interesariuszy. Dzięki tym krokom łatwiej jest osiągnąć sytuację "win-win"[1].

* 1. **Narzędzia automatyzacji i ciągłej integracji w projektach**

Automatyzacja procesów jest kluczowym elementem w procesie CI/CD. Umożliwia ona zapewnienie, że wszystkie etapy tworzenia oprogramowania, od integracji po wdrożenie, są realizowane zgodnie z wymaganiami, a jednocześnie pomaga zredukować koszty i czas związany z ręcznymi operacjami.

Automatyzacja odgrywa istotną rolę w poprawie jakości produktów oprogramowania, co ostatecznie przekłada się na większą satysfakcję klienta. Polega ona na wykorzystaniu narzędzi i skryptów do automatyzacji rutynowych i powtarzalnych zadań, takich jak budowanie aplikacji, konfiguracja środowiska, oraz wdrażanie nowych wersji.

Zautomatyzowane procesy mogą być stosowane do weryfikacji i integracji zmian wprowadzonych w kodzie źródłowym, a ich wykonanie odbywa się automatycznie po każdej modyfikacji, bez konieczności interwencji człowieka. Taka automatyzacja zapewnia spójność i stabilność środowiska produkcyjnego, redukując ryzyko błędów wynikających z ręcznych działań.

* + 1. **Czym jest pipeline CI/CD**

CI/CD to proces automatyzacji różnych etapów tworzenia, integracji i wdrażania oprogramowania. Wykorzystuje on różnorodne narzędzia, które pozwalają na zautomatyzowanie tych działań, dzięki czemu cały proces staje się bardziej efektywny i spójny. Pipeline CI/CD, czyli ciągły proces integracji i dostarczania, jest fundamentalnym elementem każdego projektu programistycznego, ponieważ umożliwia automatyzację wszystkich kluczowych etapów, od tworzenia aż po wdrożenie.

Główne korzyści wynikające z automatyzowanego pipeline CI/CD obejmują poprawę jakości końcowego produktu, szybsze identyfikowanie i naprawianie problemów oraz pewność, że dostarczane oprogramowanie spełnia oczekiwania użytkowników. Automatyzacja procesów w pipeline CI/CD usprawnia cały proces tworzenia oprogramowania, co pozwala na szybsze wprowadzanie innowacji i bardziej elastyczne reagowanie na zmieniające się potrzeby rynkowe.[2]

W pipeline CI/CD stosuje się różnorodne techniki automatyzacji, które mogą obejmować:

* Automatyzacja budowania: Proces tworzenia skompilowanej wersji aplikacji z kodu źródłowego, który jest w pełni zautomatyzowany i uruchamiany automatycznie po każdej zmianie w repozytorium kodu.
* Automatyzacja integracji: Automatyczne łączenie i testowanie różnych modułów aplikacji w celu upewnienia się, że współpracują ze sobą prawidłowo.
* Automatyzacja wdrażania: Proces wdrażania aplikacji na środowisko produkcyjne jest zautomatyzowane, co minimalizuje ryzyko błędów związanych z ręcznym wdrażaniem.
  + 1. **Narzędzia wspierające automatyzację pipeline CI/CD**

Do pisania pipelinów CI/CD kluczową rolę odgrywają narzędzia, które wspierają automatyzację poszczególnych etapów procesu tworzenia oprogramowania. Narzędzi pozwalają na automatyczne budowanie, testowanie oraz wdrażanie aplikacji.

Do najczęściej stosowanych narzędzi do tworzenia pipeline CI/CD należą:

* **Jenkins:** Jest to jedno z najbardziej popularnych narzędzi CI/CD, które umożliwia automatyzację różnych zadań w procesie tworzenia oprogramowania. Jenkins oferuje szeroki zakres pluginów, które wspierają różne technologie, co sprawia, że jest bardzo elastyczny i może być dostosowany do specyficznych potrzeb zespołów projektowych. [3]
* **GitLab CI/CD**: GitLab oferuje wbudowane funkcje CI/CD, które są zintegrowane z systemem kontroli wersji. Dzięki temu programiści mogą łatwo skonfigurować pipeline CI/CD bez potrzeby korzystania z zewnętrznych narzędzi. GitLab CI/CD wspiera również automatyzację testów oraz wdrożeń, co czyni go kompletnym rozwiązaniem dla zespołów DevOps.[4]
* **CircleCI**: To narzędzie CI/CD jest często wykorzystywane przez zespoły, które preferują szybkie i łatwe konfiguracje pipeline'ów. CircleCI wspiera automatyzację testów oraz wdrożeń, a jego integracja z GitHubem i Bitbucketem ułatwia zarządzanie kodem źródłowym.[5]
  + 1. **Przyszłość CI/CD i automatyzacji w zarządzaniu projektami**

Technologie związane z CI/CD oraz automatyzacją rozwijają się dynamicznie, a ich przyszłość wydaje się być bardzo obiecująca. Zespoły programistyczne coraz częściej korzystają z zaawansowanych narzędzi wspierających automatyzację, które integrują się z innymi systemami, takimi jak monitorowanie wydajności aplikacji czy zarządzanie infrastrukturą w chmurze.

W przyszłości możemy spodziewać się jeszcze większej integracji CI/CD z procesami DevOps oraz rosnącego znaczenia automatyzacji testów opartych na sztucznej inteligencji (AI), które będą w stanie jeszcze lepiej przewidywać i wykrywać potencjalne problemy w kodzie. Automatyzacja zadań związanych z zarządzaniem projektami będzie odgrywać kluczową rolę w dostarczaniu wysokiej jakości oprogramowania w coraz krótszych cyklach czasowych. [6]

1. **Przegląd używanych technologii**

W ramach projektu zastosowano zestaw technologii, które pełnią kluczową rolę w jego realizacji. Technologie te wspierają różne aspekty tworzenia i zarządzania oprogramowaniem, od kontroli wersji, poprzez automatyzację budowania i zarządzanie zależnościami, aż po testowanie i integrację. Każda z wybranych technologii, takich jak Git, Maven, Spring Boot, czy Jenkins, została dobrana z myślą o zwiększeniu efektywności pracy zespołu deweloperskiego oraz zapewnieniu wysokiej jakości końcowego produktu. W kolejnych sekcjach omówione zostaną szczegółowo funkcje i zalety tych narzędzi.

* 1. **Git jako system kontroli wersji**

Git to jedno z najczęściej używanych współcześnie narzędzi do kontroli wersji, zyskało ogromną popularność na całym świecie. Git został stworzony w 2005 roku przez Linusa Torvaldsa, znanego jako twórca jądra systemu operacyjnego Linux. Jest to projekt open source, który jest aktywnie rozwijany i utrzymywany. Wiele projektów oprogramowania, zarówno komercyjnych, jak i open source, polega na Git w zakresie kontroli wersji. Programiści z doświadczeniem w pracy z Git stanowią dużą część dostępnych specjalistów w dziedzinie rozwoju oprogramowania, a Git działa dobrze na szerokiej gamie systemów operacyjnych oraz zintegrowanych środowisk programistycznych (IDE). Git jest przykładem rozproszonego systemu kontroli wersji (DVCS), co oznacza, że każdy programista posiada swoją kopię repozytorium, która zawiera pełną historię zmian projektu. W przeciwieństwie do tradycyjnych systemów kontroli wersji, takich jak CVS czy Subversion (SVN), Git nie polega na jednym centralnym repozytorium[26].

Git został zaprojektowany z myślą o wydajności, bezpieczeństwie i elastyczności. W porównaniu z innymi systemami kontroli wersji, Git charakteryzuje się bardzo wysoką wydajnością, szczególnie w obszarze zatwierdzania zmian, tworzenia gałęzi, łączenia oraz porównywania poprzednich wersji kodu. Algorytmy zastosowane w Git są zoptymalizowane pod kątem efektywnego zarządzania dużymi drzewami plików źródłowych, które są często modyfikowane. Git koncentruje się na rzeczywistej zawartości plików, co oznacza, że zmiany nazw plików, ich podziały czy przestawienia nie wpływają na integralność historii wersji. Zastosowanie zapisywania zawartości katalogów i metadanych wersji pozwala na optymalne zarządzanie zasobami.

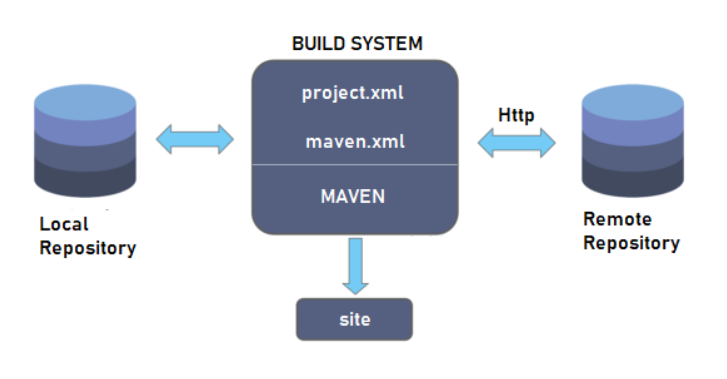
Jednym z priorytetów Git jest zapewnienie integralności zarządzanego kodu źródłowego. Wszystkie obiekty w repozytorium Git, takie jak pliki, katalogi, wersje, tagi i commit'y, są zabezpieczone kryptograficznie za pomocą algorytmu SHA1. Dzięki temu historia zmian jest odporna na przypadkowe lub celowe modyfikacje, co gwarantuje pełną ścieżkę audytu. Git oferuje wysoką elastyczność, wspierając różne typy przepływów pracy, zarówno w małych, jak i dużych projektach. System ten pozwala na efektywne zarządzanie gałęziami i tagami, a operacje takie jak łączenie gałęzi czy cofanie zmian są integralną częścią historii zmian. W porównaniu do innych systemów kontroli wersji, Git oferuje zaawansowane możliwości śledzenia zmian i zarządzania projektem. [7]

* 1. **Maven jako narzędzie do zarządzania i budowy projektu**

Maven to potężne narzędzie do zarządzania projektami, które upraszcza proces budowania projektów opartych na języku Java. Jest to oprogramowanie zarządzające projektem, oparte na koncepcji modelu obiektu projektu (POM - Project Object Model). Maven automatyzuje wiele zadań związanych z budowaniem, raportowaniem oraz dokumentacją projektu, co pozwala programistom skupić się na rozwijaniu kodu zamiast na konfiguracji. Maven umożliwia programistom automatyzację wielu zadań, takich jak tworzenie struktury katalogów, kompilacja kodu, testowanie, pakowanie oraz wdrażanie aplikacji. Dzięki temu znacząco skraca ilość ręcznych kroków, które wcześniej były niezbędne w procesie budowania oprogramowania, czyniąc go prostym i efektywnym. Jednym z najważniejszych aspektów Mavena jest zarządzanie zależnościami za pomocą repozytoriów. Maven automatycznie pobiera potrzebne pliki jar z centralnego repozytorium do lokalnej maszyny, co znacznie ułatwia proces budowy projektu. Maven może być również używany do budowania i zarządzania projektami napisanymi w innych językach programowania, takich jak C# czy Ruby.

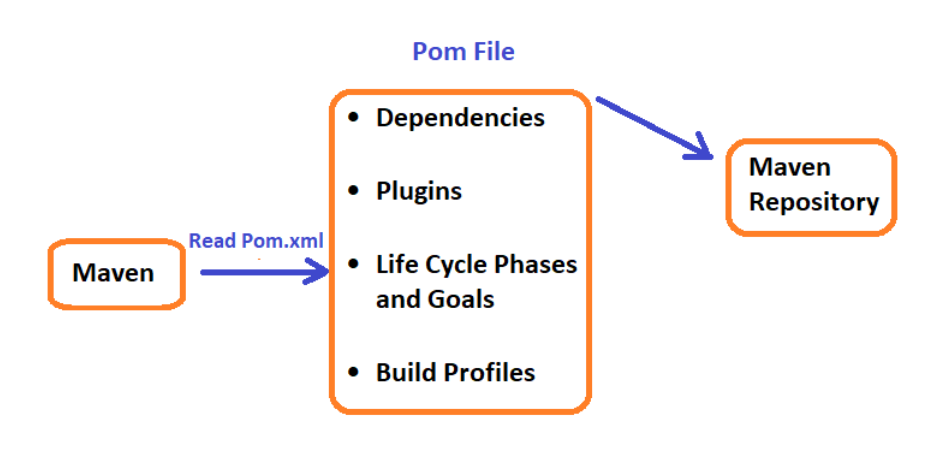
Maven oferuje wiele korzyści, które przyczyniają się do usprawnienia procesu tworzenia oprogramowania:

* **Minimalna konfiguracja**: Maven wykorzystuje podejście "konwencja ponad konfiguracją", co oznacza, że większość ustawień jest domyślna, a programista musi konfigurować jedynie niestandardowe aspekty projektu.
* **Zarządzanie zależnościami**: Maven automatycznie zarządza bibliotekami potrzebnymi w projekcie, pobierając je z centralnego repozytorium.
* **Automatyzacja**: Dzięki wbudowanej automatyzacji, Maven umożliwia wielokrotne i powtarzalne budowanie projektów bez potrzeby ręcznego zarządzania procesem. [8]



Rysunek 1 Schemat działania systemu Maven [8]

Ilustracja przedstawia interakcję systemu Maven z lokalnym i zdalnym repozytorium. Pliki project.xml oraz maven.xml są wykorzystywane przez Maven do zarządzania projektem, w tym do pobierania zależności zdalnego repozytorium przez http i przechowywania ich w lokalnym repozytorium.



Rysunek 2 Struktura piliku POM w Mavenie [8]

Diagram przedstawia sposób, w jaki Maven wykorzystuje pliki POM do zarządzania zależnościami, wtyczkami, fazami cyklu życia oraz profilami budowania.

* 1. **Spring Boot jako framework do tworzenia aplikacji**

Spring Boot to narzędzie open-source, które znacznie ułatwia korzystanie z frameworków opartych na Javie do tworzenia mikrousług i aplikacji webowych. Aby zrozumieć znaczenie Spring Boot, należy zacząć od samego języka Java, jednego z najpopularniejszych i najczęściej używanych języków programowania na świecie. Java jest ceniona przez programistów za swoją elastyczność i przyjazność, co sprawia, że jest powszechnie wykorzystywana do tworzenia różnych aplikacji, od mediów społecznościowych i gier, po zaawansowane aplikacje korporacyjne.

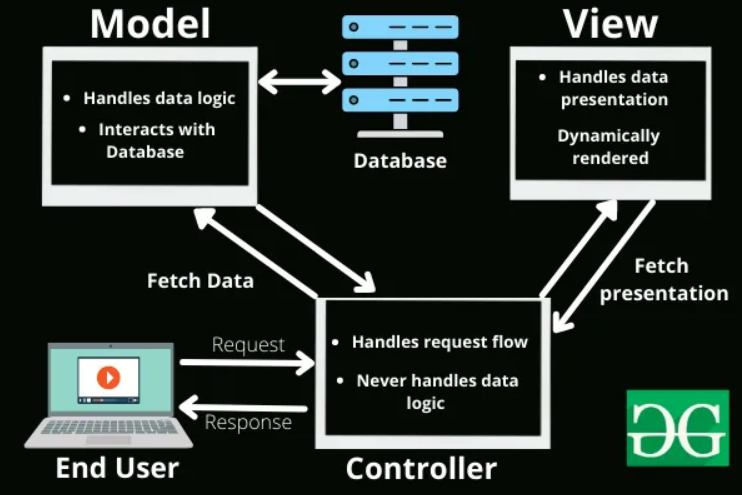
Spring Boot to nie tylko kolejny framework – to rewolucja w świecie programowania w Javie. Upraszcza on wiele złożonych i czasochłonnych aspektów tworzenia aplikacji, co pozwala programistom skupić się na budowaniu solidnych, skalowalnych i innowacyjnych rozwiązań. W tym przeglądzie przyjrzymy się głównym założeniom Spring Boot, odkrywając, dlaczego zyskał tak dużą popularność oraz jak może pomóc w realizacji pomysłów programistycznych w sposób prosty i elegancki.

Architektura Spring Boot:

Spring Boot korzysta z architektury Model-View-Controller (MVC), która jest kluczowym wzorcem projektowym stosowanym w tworzeniu oprogramowania. MVC dzieli aplikację na trzy współpracujące komponenty: Model, Widok (View) i Kontroler. Każdy z tych elementów pełni określoną rolę, co ułatwia zarządzanie kodem i jego rozwój.

* **Model:** Odpowiada za logikę biznesową i operacje na danych. W Spring Boot model integruje się z bazą danych, upraszczając zarządzanie danymi i dostęp do nich.
* **Widok:** Zajmuje się logiką interfejsu użytkownika, generując interfejs na podstawie danych dostarczonych przez model. Spring Boot wspiera różne technologie widoków, takie jak Thymeleaf, FreeMarker czy JSP.
* **Kontroler:** Obsługuje interakcje użytkownika, przetwarza dane z modelu i przekazuje je do widoku.

Architektura MVC w Spring Boot promuje przejrzystość kodu, umożliwia modularne podejście do programowania oraz ułatwia współpracę w zespole, co zwiększa możliwości ponownego wykorzystania kodu.[9]



Rysunek 3 Schemat architektury Model – View – Controller (MVC) w aplikacji Spring Boot [9]

* 1. **Relacyjne bazy danych i ich zastosowanie w projektach**

Relacyjny System Zarządzania Bazą Danych (RDBMS) to system, który wykorzystuje model relacyjny do przechowywania danych w tabelach. Tabele składają się z wierszy i kolumn, gdzie każda kolumna zawiera dane dla określonej kategorii, a każdy wiersz reprezentuje pojedynczy rekord danych dla tej kategorii. Niezależnie od tego, czy tworzymy aplikację, czy stronę internetową, rolą programisty jest zaprojektowanie witryny, aby była atrakcyjna wizualnie, zapewnienie płynnego korzystania z niej oraz połączenie jej z wydajną bazą danych, która będzie przechowywać dane tej witryny. Istnieje wiele rodzajów baz danych, takich jak bazy danych hierarchiczne, sieciowe, obiektowe oraz relacyjne. Każdy z tych typów baz danych ma swoje specyficzne zastosowania, a wybór odpowiedniego systemu należy do programisty, który musi zdecydować, który z nich najlepiej spełni wymagania jego oprogramowania. SQL (Structured Query Language) jest standardowym interfejsem użytkownika oraz aplikacji dla relacyjnych baz danych. Większość współczesnych systemów IT oraz aplikacji opiera się na relacyjnych bazach danych.

Relacyjne bazy danych są powszechnie stosowane, ponieważ pozwalają na efektywne zarządzanie dużymi ilościami danych oraz obsługę złożonych zapytań. Tabele w relacyjnej bazie danych mogą być powiązane na różne sposoby, co ułatwia zrozumienie powiązań między różnymi elementami danych oraz przetwarzanie zapytań. Przed wyborem relacyjnego systemu zarządzania bazą danych (RDBMS) warto wziąć pod uwagę kilka istotnych aspektów:

**Konfiguracja początkowa**: Upewnij się, że system bazy danych wspiera synchronizację z innymi platformami oraz integrację z istniejącą architekturą danych. Elementy te są niezbędne do optymalizacji pracy DBMS oraz zapewnienia ciągłości operacji.

**Bezpieczeństwo danych:** Każdy system DBMS oferuje różne mechanizmy zabezpieczeń. Wybierając system bazy danych, warto sprawdzić, czy oferuje takie funkcje, jak szyfrowanie, programowalne procedury oraz uprawnienia dostępu. Ważne jest również, aby system miał mechanizmy autoryzacji i uwierzytelniania dostępu, które pozwalają programistom na definiowanie uprawnień.

Wśród popularnych relacyjnych systemów zarządzania bazami danych (RDBMS) wykorzystywanych w wielu znanych aplikacjach można wymienić:

* **Oracle**: Komercyjny, zaawansowany RDBMS, szeroko stosowany w dużych przedsiębiorstwach.
* **MySQL:** Popularny, otwartoźródłowy system zarządzania bazą danych, często wykorzystywany w aplikacjach internetowych.
* **MariaDB:** Rozgałęzienie MySQL, które oferuje dodatkowe funkcje i wydajność.
* **Microsoft SQL Server:** Komercyjny RDBMS oferowany przez firmę Microsoft, często stosowany w aplikacjach korporacyjnych.
* **PostgreSQL:** Wydajny, otwartoźródłowy RDBMS, znany ze swojej elastyczności, niezawodności oraz zgodności z zasadami ACID.[10]

Najlepszym wyborem do aplikacji zarządzania projektami jest PostreSQL i to on został użyty w pracy dyplomowej. PostgreSQL to potężny system zarządzania relacyjnymi bazami danych, który obsługuje język SQL i oferuje zaawansowane funkcje umożliwiające bezpieczne przechowywanie i zarządzanie najbardziej wymagającymi obciążeniami danych. Jest znany z architektury, niezawodności, integralności danych oraz bogatego zestawu funkcji, a także zaangażowania społeczności open source w dostarczanie innowacyjnych i wysokowydajnych rozwiązań. [22]

* 1. **Jenkins dla ciągłej integracji i dostarczania (CI/CD)**

Jenkins to narzędzie open-source do automatyzacji procesów ciągłej integracji i dostarczania (CI/CD), napisane w języku Java. Służy do implementacji przepływów pracy CI/CD, które nazywane są pipeline'ami. Pipeline'y automatyzują testowanie i raportowanie zmian w bazie kodu w czasie rzeczywistym, a także ułatwiają integrację różnych gałęzi kodu do głównej gałęzi. Pozwalają również na szybkie wykrywanie błędów w kodzie, budowanie oprogramowania, automatyzację testów, przygotowanie kodu do wdrożenia oraz ostateczne wdrożenie na serwerach fizycznych, wirtualnych oraz w chmurze. Istnieje kilka komercyjnych wersji Jenkins, ale tutaj opisano projekt Open Source. Z czasem do Jenkins zostały dodane funkcje ciągłego dostarczania i wdrażania. Ciągłe dostarczanie to proces automatyzacji budowania i pakowania kodu, który następnie jest wdrażany w środowiskach testowych i produkcyjnych. Ciągłe wdrażanie automatyzuje ostatni krok, czyli wdrożenie kodu na docelowe środowisko produkcyjne. Automatyzacja w Jenkins minimalizuje liczbę błędów, ponieważ wszystkie kroki i najlepsze praktyki są zakodowane w systemie. Jenkins określa stan docelowy, a serwer automatyzacji dba o to, aby został on osiągnięty. Dodatkowo, automatyzacja przyspiesza tempo wdrożeń, ponieważ nie jest ograniczona dostępnością personelu. Wreszcie, Jenkins redukuje stres w zespołach deweloperskich i operacyjnych, eliminując potrzebę wdrożeń w nocy czy w weekendy. Jenkins działa jako serwer na różnych platformach, w tym Windows, MacOS, Unix i szczególnie Linux. Wymaga maszyny wirtualnej Java 8 lub nowszej i może być uruchamiany na Oracle JRE lub OpenJDK. Zazwyczaj Jenkins działa jako servlet Java w serwerze aplikacji Jetty, ale może być także uruchamiany na innych serwerach aplikacji Java, takich jak Apache Tomcat. Ostatnio Jenkins został dostosowany do pracy w kontenerze Docker, a gotowe obrazy Jenkins są dostępne w repozytorium Docker Hub. Aby korzystać z Jenkins, tworzy się tzw. pipeline'y. Pipeline to seria kroków, które serwer Jenkins wykona, aby zrealizować wymagane zadania w procesie CI/CD. Te kroki są zapisywane w pliku tekstowym Jenkinsfile, który używa składni przypominającej JSON. Serwer Jenkins odczytuje Jenkinsfile i wykonuje zawarte w nim polecenia, przesuwając kod przez kolejne etapy pipeline'u – od kodu źródłowego aż po wdrożenie w środowisku produkcyjnym. [3]

* 1. **Postman jako narzędzie do testowania i dokumentacji interfejsów API**

Postman to narzędzie do tworzenia, testowania i modyfikowania API (interfejsów programistycznych), które wspomaga programistów w pracy nad API. Jest to wszechstronne narzędzie, które zawiera prawie wszystkie funkcje potrzebne programistom do efektywnej pracy z API. Każdego miesiąca korzysta z niego ponad 5 milionów programistów, co sprawia, że rozwój API staje się prosty i intuicyjny. Postman umożliwia wykonywanie różnych typów zapytań HTTP (GET, POST, PUT, PATCH), zapisanie środowisk do późniejszego użycia oraz konwersję API na kod w różnych językach, takich jak JavaScript i Python.

Postman jest nieocenionym narzędziem we współczesnym rozwoju API, oferującym szereg funkcji, które usprawniają ten proces. Oto kluczowe aspekty, które czynią Postman potężnym narzędziem w dziedzinie rozwoju API:

**Wszechstronne metody zapytań**: Postman obsługuje szeroki wachlarz metod zapytań HTTP, takich jak GET, POST, PUT, DELETE i PATCH, co pozwala programistom na pełną interakcję z API.

**Elastyczne formaty treści zapytań:** Programiści mogą korzystać z różnych formatów treści zapytań, takich jak form-data, dane kodowane w URL, dane surowe (raw) oraz dane binarne, co umożliwia dostosowanie zapytań do specyficznych wymagań różnych API.

**Uproszczone uwierzytelnianie:** Postman upraszcza proces uwierzytelniania, oferując wsparcie dla różnych metod, takich jak klucze API, OAuth oraz Basic Auth. To ułatwia zabezpieczenie interakcji z API, zapewniając solidne i bezpieczne środowisko programistyczne.

* **Zorganizowane testowanie API:** Kolekcje w Postman służą jako potężne narzędzie organizacyjne, umożliwiając programistom kategoryzowanie i zarządzanie zapytaniami API w sposób efektywny.
* **Efektywna dokumentacja:** Postman doskonale sprawdza się w generowaniu dokumentacji API bezpośrednio z zapytań i kolekcji. [11]

1. **Analiza wymagań**

Analiza wymagań jest kluczowym etapem w procesie tworzenia oprogramowania, który polega na zbieraniu, dokumentowaniu i weryfikowaniu oczekiwań oraz potrzeb użytkowników i klientów względem systemu. Celem analizy jest precyzyjne określenie, jakie funkcje i cechy powinien mieć system, aby spełniać swoje zadania oraz zapewnić wysoką jakość i użyteczność. Poprawnie przeprowadzona analiza wymagań minimalizuje ryzyko niepowodzenia projektu oraz pomaga w dokładnym określeniu zakresu i budżetu projektu.

* 1. **Wymagania funkcjonalne systemu zarządzania projektami**

Dokładne zdefiniowanie wymagań funkcjonalnych jest fundamentem sukcesu każdego projektu inżynierii oprogramowania. Wymagania funkcjonalne określają, co system musi wykonywać, aby spełniać swoje założenia i realizować cele biznesowe. W kontekście systemu zarządzania projektami, wymagania te odnoszą się do specyficznych funkcji, które są niezbędne do skutecznego zarządzania projektami, takich jak zarządzanie zadaniami, monitorowanie postępów, raportowanie, oraz zarządzanie zasobami. Wymagania funkcjonalne to konkretne czynności, które system musi realizować, aby działać zgodnie z oczekiwaniami. Ich brak lub niewłaściwa implementacja skutkuje nieprawidłowym działaniem systemu. W praktyce oznacza to, że jeśli system nie spełnia wymagań funkcjonalnych, nie będzie w stanie prawidłowo realizować swoich podstawowych funkcji. Co więcej, precyzyjne zdefiniowanie tych wymagań ułatwia komunikację między zespołami technicznymi a klientami, co zwiększa szanse na udane wdrożenie i eksploatację systemu.

Przykłady wymagań funkcjonalnych w systemie zarządzania projektami

* System musi umożliwiać tworzenie, edytowanie i usuwanie projektów oraz zadań.
* System musi zapewniać autoryzację użytkowników na różnych poziomach dostępu, na przykład administratora, menedżera projektu i członka zespołu.
* System powinien umożliwiać generowanie raportów o postępach w realizacji projektu.
* System musi śledzić czas pracy zespołu nad poszczególnymi zadaniami oraz umożliwiać analizę efektywności.[12]
  1. **Wymagania niefunkcjonalne systemu zarządzania projektami**

Wymagania niefunkcjonalne opisują, jak system powinien działać, a nie co powinien robić. Choć mogą wydawać się mniej istotne niż wymagania funkcjonalne, ich spełnienie jest kluczowe dla jakości użytkowania systemu. Wymagania te wpływają na takie aspekty jak wydajność, niezawodność, użyteczność oraz dostępność systemu. Wymagania niefunkcjonalne koncentrują się na sposobie działania systemu. Nie dotyczą one bezpośrednio funkcji systemu, ale mają istotny wpływ na to, jak system będzie odbierany przez użytkowników. Brak spełnienia wymagań niefunkcjonalnych może prowadzić do frustracji użytkowników i ograniczenia skuteczności systemu. Co więcej, ich odpowiednie zdefiniowanie jest kluczowe dla zapewnienia, że system będzie nie tylko funkcjonalny, ale również wydajny i łatwy w użyciu. Niefunkcjonalne aspekty, takie jak szybkość działania czy łatwość obsługi, mają bezpośredni wpływ na satysfakcję użytkowników i sukces całego projektu.

Przykłady wymagań niefunkcjonalnych w systemie zarządzania projektami

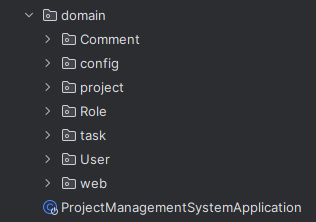
* System powinien działać z szybkością, która umożliwia płynne wykonywanie operacji, na przykład czas ładowania strony powinien wynosić poniżej 2,5 sekund.
* System musi być dostępny 24/7, z minimalnymi przestojami na konserwację.
* System musi obsługiwać jednocześnie co najmniej 1000 aktywnych użytkowników.
* System powinien być łatwy w obsłudze, z intuicyjnym interfejsem, który nie wymaga długiego szkolenia użytkowników.[12]

1. **Architektura i projekt systemu**

Architektura systemu zarządzania projektami opiera się na nowoczesnych technologiach webowych i wzorcach projektowych, które umożliwiają elastyczne i skalowalne rozwiązania. System jest zbudowany z wykorzystaniem Spring Boot, co pozwala na szybkie tworzenie aplikacji w oparciu o model MVC (Model-View-Controller). Model ten wspiera oddzielenie logiki biznesowej od warstwy prezentacji, co znacząco ułatwia zarządzanie kodem i jego rozwój. Centralnym elementem systemu są encje reprezentujące kluczowe zasoby, takie jak User, Project, Task, Role, oraz Comment. Każda encja odpowiada tabeli w relacyjnej bazie danych, a ich relacje są odwzorowane za pomocą mapowań JPA (Java Persistence API). System integruje się z bazą danych za pomocą zdefiniowanego modelu danych, co umożliwia efektywne zarządzanie danymi i ich relacjami. W implementacji wykorzystano także Maven do zarządzania zależnościami oraz Jenkins do automatyzacji procesów CI/CD, co zapewnia płynność w integracji kodu, testowaniu i wdrażaniu nowych wersji aplikacji. Integracja tych narzędzi pozwala na utrzymanie wysokiej jakości kodu oraz szybką reakcję na zmieniające się wymagania projektowe. W dalszych sekcjach szczegółowo omówiono strukturę aplikacji, modele danych oraz procesy związane z automatyzacją wdrożeń.

* 1. **Architektura aplikacji z wykorzystaniem Spring Boot**

Struktura aplikacji jest zorganizowana w dobrze przemyślaną strukturę pakietów, co jest kluczowe dla utrzymania czytelności oraz skalowalności kodu.



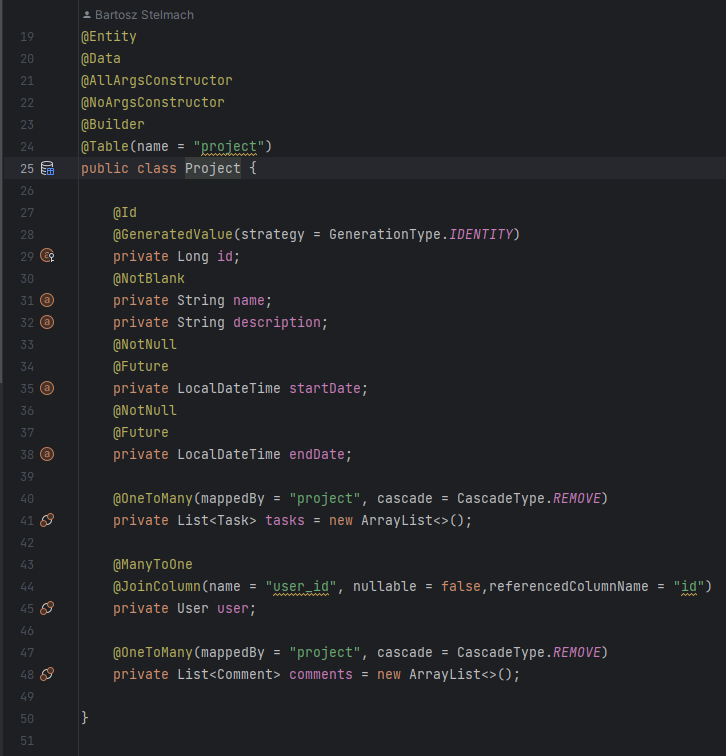
Rysunek 4 Struktury projektu [13]

Każda klasa encji i związane z nią komponenty (DTO, repozytorium, mapper, serwis) są zorganizowane w osobne pakiety, co ułatwia zarządzanie kodem.

* Comment – Zawiera klasy związane z zarządzaniem komentarzami w systemie.
* Project - Podobnie jak powyżej, ten pakiet obejmuje zarządzanie projektami
* Role - Pakiet do zarządzania rolami użytkowników
* Task - Odpowiada za zarządzanie zadaniami.
* User - Zajmuje się użytkownikami systemu
* Config - Pakiet ten zawiera klasy odpowiedzialne za konfigurację bezpieczeństwa
* Web - Znajdują się tu klasy Rest Controlerów odpowiedzialne za obsługę żądań HTTP

Praktycznie każdy tutaj blok zawiera komponenty takie jak encje, repozytoria, serwisy, mapery i kontrolery. Warto więc wyjaśnić czym one są i jaką pełnią funkcje.

* **Encja (Entity)** - Encja to klasa reprezentująca tabelę w bazie danych. Każda instancja tej klasy to pojedynczy rekord w tej tabeli. Encje są centralnym elementem w mapowaniu obiektowo-relacyjnym (ORM), które umożliwia mapowanie obiektów w kodzie na rekordy w bazie danych. Encje są adnotowane w JPA, na przykład **@Entity**, co wskazuje, że klasa ta jest mapowana na tabelę bazy danych. [14]



Rysunek 5 Implementacja encji [13]

Na załączonym wyżej obrazie widzimy klasę Project, która jest jedną z głównych encji w systemie zarządzania projektami. Ta klasa została zdefiniowana przy użyciu adnotacji z biblioteki Lombok oraz JPA ( Java Persistence API), co znacząco upraszcza kod, eliminując potrzebę ręcznego pisania konstruktorów, metod ‘getter’, ‘setter’ oraz innych typowych metod.

Adnotacje JPA:

* **@Entity:** Określa, że klasa Project jest encją i będzie mapowana na tabelę w bazie danych.
* **@Table(name = "project"):** Określa nazwę tabeli w bazie danych, na którą mapowana jest klasa Project.
* **@Id:** Wskazuje, że pole id jest kluczem głównym tej tabeli.
* **@GeneratedValue(strategy = GenerationType.IDENTITY):** Wskazuje, że wartość klucza głównego jest automatycznie generowana przez bazę danych.

Relacje między encjami:

* **@OneToMany:** Klasa Project jest powiązana z wieloma encjami Task oraz Comment, co oznacza, że jeden projekt może mieć wiele zadań i komentarzy.
* **@ManyToOne:** Wskazuje, że wiele projektów może być przypisanych do jednego użytkownika (User).

Pola encji:

Kluczowe pola takie jak name, description, startDate, endDate są odpowiednio adnotowane, co pozwala na ich walidację oraz prawidłowe przechowywanie w bazie danych.

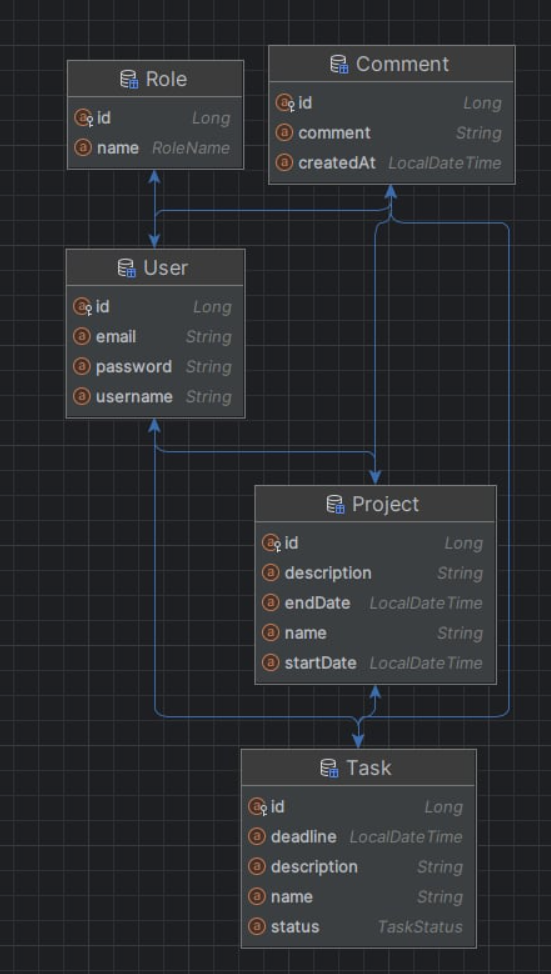
Relacje między projektami, zadaniami oraz użytkownikami są odpowiednio zdefiniowane, co umożliwia mapowanie związków między tymi encjami.

Biblioteka Lombok:

* **@Data:** Generuje automatycznie metody getter, setter, toString, equals i hashCode, co znacząco upraszcza kod.
* **@AllArgsConstructor oraz @NoArgsConstructor:** Tworzy odpowiednie konstruktory – jeden z wszystkimi parametrami, drugi bez parametrów.
* **@Builder:** Umożliwia wykorzystanie wzorca projektowego Builder, który ułatwia tworzenie instancji klasy Project.

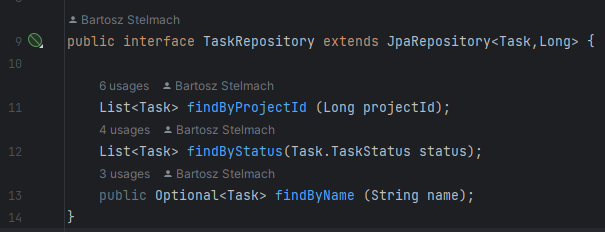
Encja Project reprezentuje projekt w systemie zarządzania projektami. Jest ona kluczowa dla funkcjonowania systemu, ponieważ przechowuje wszystkie informacje dotyczące projektu, takie jak nazwa, opis, daty rozpoczęcia i zakończenia oraz powiązania z użytkownikami, zadaniami i komentarzami. Dzięki zastosowaniu Lomboka, kod jest bardziej przejrzysty i mniej podatny na błędy wynikające z ręcznego pisania standardowych metod.

Diagram przedstawia relacje między encjami w systemie zarządzania projektami. Encje takie jak User, Role, Project, Task, oraz Comment są ze sobą powiązane, co umożliwia zarządzanie użytkownikami, ich rolami, projektami oraz przypisanymi do nich zadaniami i komentarzami. Diagram ukazuje kluczowe zależności między tymi encjami, takie jak relacja jeden-do-wielu między User a Project, oraz Project a Task, co odzwierciedla strukturę danych w aplikacji.



Rysunek 6 Powiązania między encjami [13]

* **Repozytorium (Repository)** - Repozytorium to wzorzec projektowy w architekturze oprogramowania, który pełni funkcję warstwy dostępu do danych. Jego głównym zadaniem jest oddzielenie logiki biznesowej od operacji na danych, co pozwala na bardziej przejrzysty i elastyczny kod. W kontekście aplikacji, repozytorium odpowiada za interakcje z bazą danych, realizując operacje takie jak zapisywanie, odczytywanie, aktualizowanie i usuwanie danych. W praktyce, repozytorium działa jako pośrednik między bazą danych a pozostałymi warstwami aplikacji, takimi jak serwis czy kontroler. Dzięki temu, logika biznesowa (umieszczona w serwisach) nie musi bezpośrednio korzystać z złożonych operacji na bazie danych, co ułatwia zarządzanie kodem i zwiększa jego czytelność. W Spring Data JPA, repozytorium jest zwykle interfejsem, który rozszerza gotowe interfejsy takie jak JpaRepository lub CrudRepository, co znacząco upraszcza implementację i pozwala na korzystanie z gotowych metod do zarządzania encjami. [15]



Rysunek 7 Implementacja repozytorium [13]

Na załączonym obrazku widzimy interfejs TaskRepository, który rozszerza interfejs JpaRepository z biblioteki Spring Data JPA. Dzięki temu interfejsowi możemy zarządzać encją Task w sposób bezpośredni, korzystając z gotowych metod CRUD (Create, Read, Update, Delete) oraz dodatkowych metod definiowanych przez użytkownika.

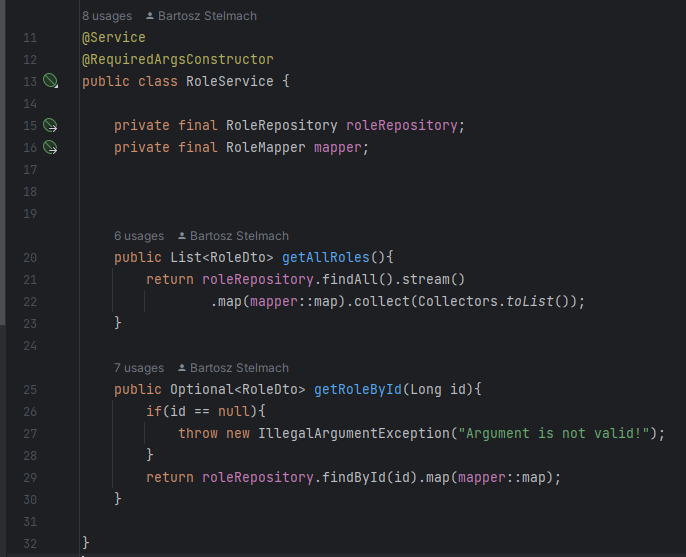
**JpaRepository<Task, Long>:** Umożliwia wykonywanie operacji na encjach Task w bazie danych. Pierwszy parametr (Task) określa typ encji, a drugi (Long) typ klucza głównego.

Definiowane metody:

* **List<Task> findByProjectId(Long projectId):** Zwraca listę zadań przypisanych do konkretnego projektu, identyfikowanego przez projectId.
* **List<Task> findByStatus(Task.TaskStatus status):** Zwraca listę zadań, które mają określony status (TaskStatus).
* **Optional<Task> findByName(String name):** Zwraca opcjonalne zadanie, którego nazwa odpowiada podanemu ciągowi znaków (name).

Interfejs TaskRepository umożliwia efektywne wyszukiwanie i zarządzanie zadaniami w systemie zarządzania projektami. Dzięki metodom definiowanym w interfejsie można łatwo wyszukiwać zadania na podstawie projektu, statusu czy nazwy. Jest to kluczowe dla implementacji logiki biznesowej w warstwie serwisowej, która z kolei wykorzystuje te metody do realizacji operacji związanych z zadaniami.

* **Service (Serwis) -** Serwis w kontekście architektury aplikacji to warstwa, która skupia się na logice biznesowej systemu. Jego głównym zadaniem jest przetwarzanie danych, które pochodzą z warstwy dostępu do danych (czyli repozytoriów), oraz koordynowanie interakcji między różnymi komponentami aplikacji. Serwis działa jako mediator, który umożliwia manipulację danymi, ich przetwarzanie i przygotowanie do dalszego wykorzystania, na przykład przez kontrolery, które obsługują żądania użytkowników.

****

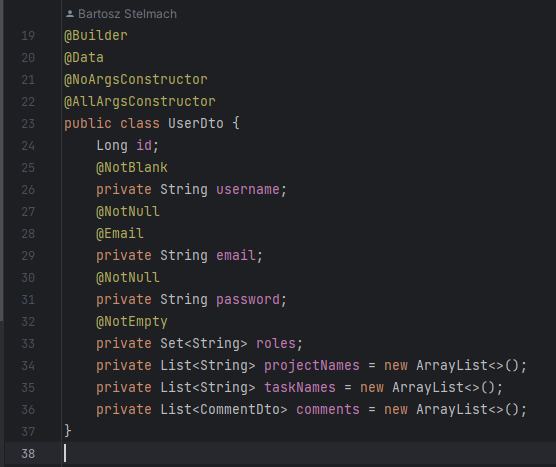
Rysunek 8 Implementacja serwisu [13]

Na załączonym obrazku widzimy klasę RoleService, która jest przykładem serwisu odpowiedzialnego za zarządzanie danymi związanymi z rolami użytkowników w systemie. **@Service**: Wskazuje, że klasa RoleService jest serwisem i powinna być traktowana jako komponent Spring, co umożliwia jej automatyczne wstrzykiwanie w inne komponenty aplikacji.

* **RoleRepository roleRepository:** Repozytorium, które zapewnia dostęp do danych przechowywanych w bazie danych, związanych z rolami.
* **RoleMapper mapper:** Mapper odpowiedzialny za konwersję danych między encjami a obiektami DTO (Data Transfer Object), co jest przydatne podczas przesyłania danych między warstwami aplikacji.
* **getAllRoles():** Zwraca listę wszystkich ról w systemie, korzystając z repozytorium do pobrania danych, a następnie mapując je na obiekty DTO przy użyciu mappera.
* **getRoleById(Long id):** Zwraca opcjonalną rolę na podstawie jej identyfikatora, najpierw sprawdzając poprawność argumentu, a następnie używając repozytorium do wyszukania roli i mappera do konwersji wyniku na DTO.

Ten serwis może być dalej używany w kontrolerach, które obsługują żądania użytkowników, na przykład podczas tworzenia nowych ról, aktualizacji istniejących, czy pobierania listy dostępnych ról w systemie.

**DTO (Data Transfer Object) -** DTO (Data Transfer Object) to wzorzec projektowy używany do przenoszenia danych między różnymi warstwami aplikacji. Jest to obiekt, który zazwyczaj nie zawiera logiki biznesowej, a jedynie pola służące do przenoszenia informacji. DTO jest używany głównie do redukcji liczby zapytań do bazy danych i zwiększenia wydajności aplikacji przez grupowanie danych potrzebnych do wyświetlenia lub przetwarzania w jednym obiekcie.[17]

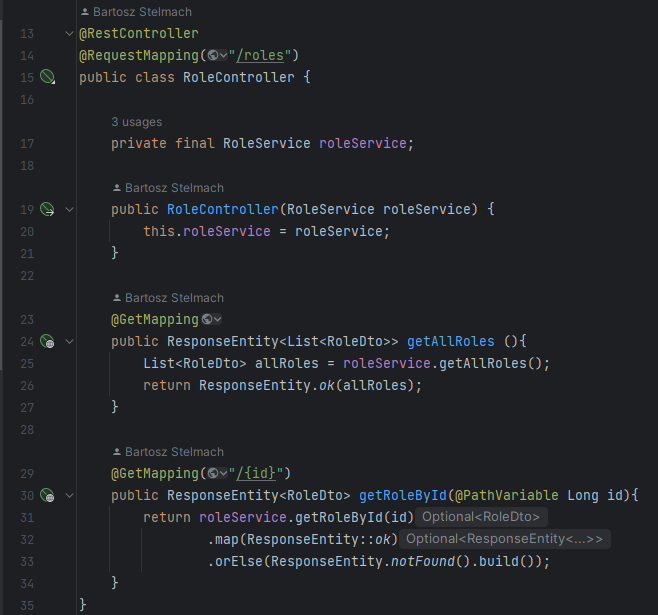


Rysunek 9 Implementacja DTO [13]

Na załączonym obrazku widzimy klasę UserDto, która reprezentuje dane użytkownika w formie DTO, używaną do przenoszenia informacji o użytkownikach w systemie zarządzania projektami.

* **id:** Identyfikator użytkownika.
* **username:** Nazwa użytkownika, nie może być pusta (adnotacja **@NotBlank**).
* **email:** Adres email użytkownika, musi być prawidłowy (adnotacja **@Email**).
* **password:** Hasło użytkownika, nie może być puste (adnotacja **@NotNull**).
* **roles:** Zbiór ról przypisanych do użytkownika, nie może być pusty (adnotacja **@NotEmpty**).
* **projectNames:** Lista nazw projektów przypisanych do użytkownika.
* **taskNames:** Lista nazw zadań przypisanych do użytkownika.
* **comments:** Lista komentarzy przypisanych do użytkownika, reprezentowana przez obiekty CommentDto.

**Rest Controller -** RestController to specjalizacja adnotacji **@Controller** używana w ramach Spring Framework. Jest ona wykorzystywana do tworzenia RESTful web services, czyli usług sieciowych, które komunikują się za pomocą protokołu HTTP i wykorzystują formaty wymiany danych takie jak JSON lub XML. RestController łączy funkcjonalność kontrolera i **@ResponseBody**, co oznacza, że metody kontrolera bezpośrednio zwracają dane jako odpowiedzi HTTP, zamiast próbować znaleźć widok (np. stronę HTML). Jest to niezwykle przydatne w aplikacjach, gdzie backend dostarcza dane dla frontendowych aplikacji lub mobilnych, zamiast renderowania interfejsu użytkownika.



Rysunek 10 Implementacja RestControllera [13]

Na załączonym obrazku widzimy implementację kontrolera RoleController, który zarządza rolami w systemie.

Adnotacje:

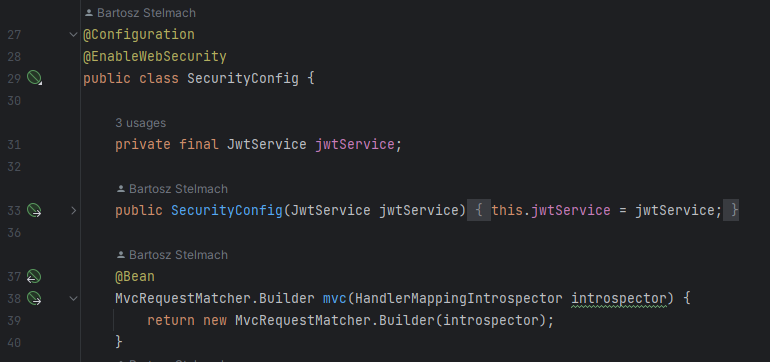
* **@RestController:** Oznacza klasę jako kontroler REST, co sprawia, że metody tej klasy bezpośrednio zwracają odpowiedzi HTTP w formacie JSON.
* **@RequestMapping("/roles"):** Ustawia bazową ścieżkę URL dla wszystkich endpointów w tej klasie. Wszystkie mapowania wewnątrz tej klasy będą miały prefiks /roles.

Pola:

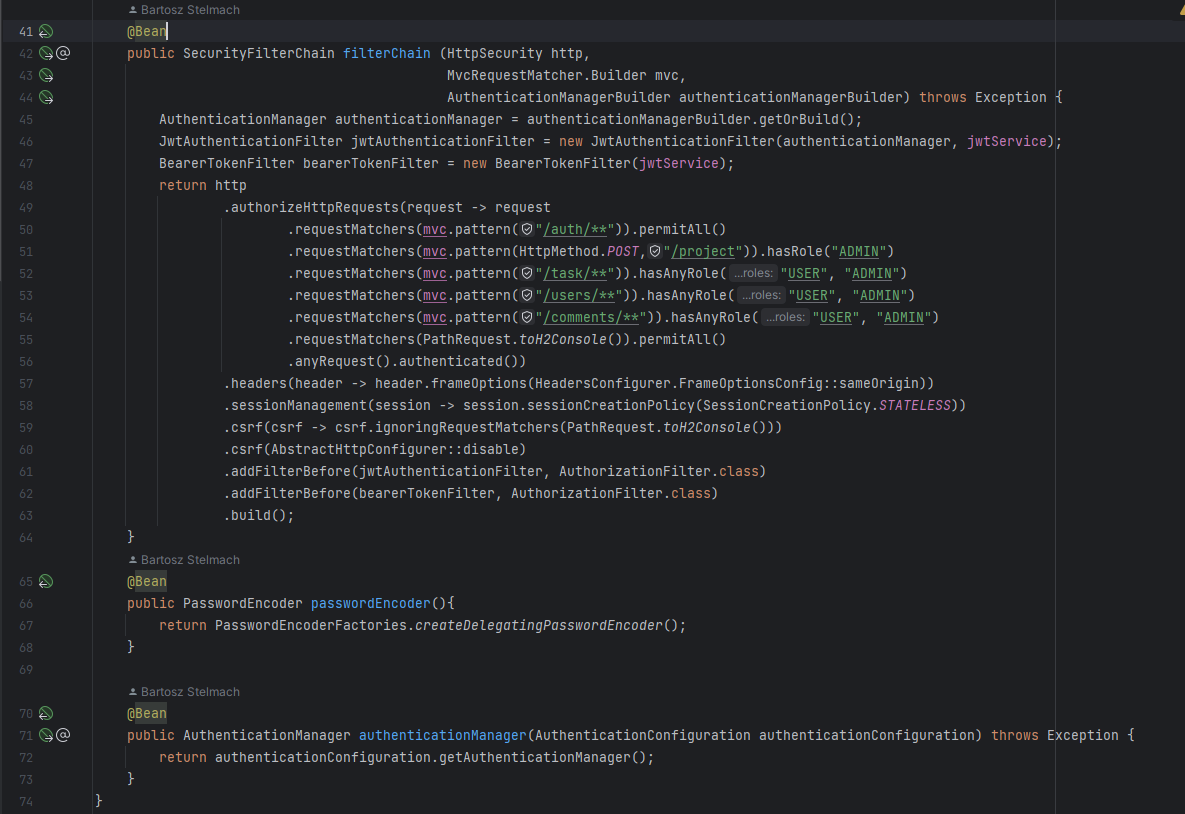
* **roleService:** To zależność wstrzykiwana przez Springa (przy pomocy adnotacji **@Autowired** lub przez konstruktor), która zarządza logiką biznesową związaną z rolami.

Metody:

* **getAllRoles():** Metoda obsługująca żądania GET pod /roles, zwraca listę wszystkich ról w systemie. Dane są pobierane przez serwis RoleService i zwracane jako ResponseEntity z listą obiektów RoleDto.
* **getRoleById(Long id):** Metoda obsługująca żądania GET pod /roles/{id}, która zwraca rolę na podstawie identyfikatora. Jeśli rola nie zostanie znaleziona, zwracana jest odpowiedź 404 (Not Found).



Rysunek 11 Konfiguracja bezpieczeństwa [13]



Rysunek 12 Konfiguracja bezpieczeństwa [13]

W konfiguracji zabezpieczeń systemu, klasa SecurityConfig pełni kluczową rolę w zarządzaniu dostępem do zasobów aplikacji oraz w implementacji mechanizmów uwierzytelniania użytkowników. Wykorzystuje ona możliwości oferowane przez bibliotekę Spring Security, która jest standardowym narzędziem w aplikacjach Java do zapewnienia odpowiedniego poziomu bezpieczeństwa.

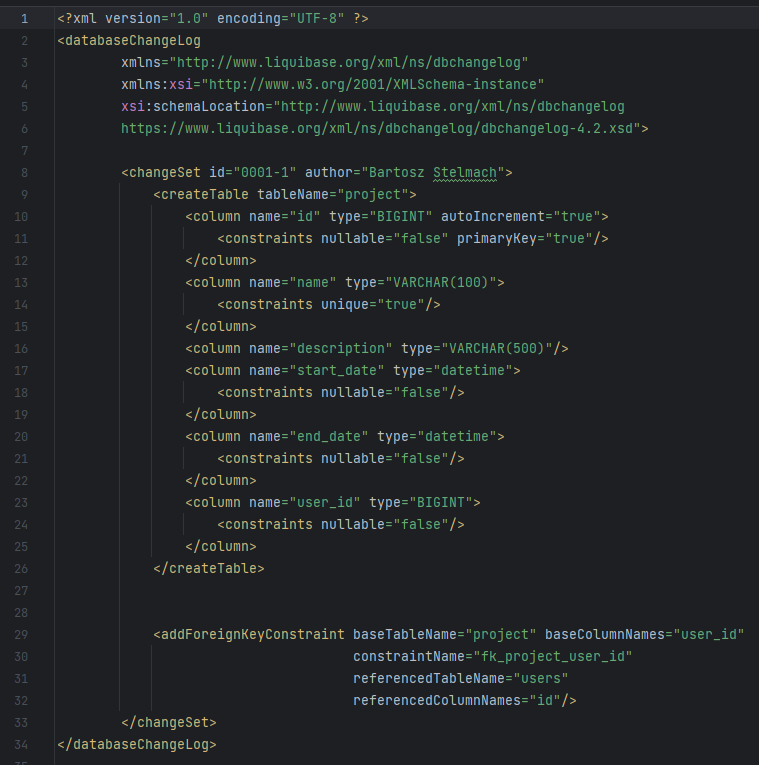
Główne elementy klasy SecurityConfig:

* **JwtService:** To serwis odpowiedzialny za obsługę tokenów JWT (JSON Web Token). Tokeny te są powszechnie stosowane w aplikacjach internetowych do autoryzacji użytkowników po ich uwierzytelnieniu. Serwis zajmuje się generowaniem tokenów dla poprawnie uwierzytelnionych użytkowników oraz ich weryfikacją podczas próby dostępu do zabezpieczonych zasobów.
* **SecurityFilterChain:** Ta metoda definiuje główny łańcuch filtrów bezpieczeństwa w aplikacji. W jej obrębie określane są zasady dostępu do poszczególnych zasobów aplikacji na podstawie przypisanych użytkownikom ról. Przykładowo, dostęp do endpointów związanych z autoryzacją (/auth/\*\*) jest otwarty dla wszystkich, podczas gdy dostęp do zarządzania projektami, zadaniami i użytkownikami (/project, /task, /users) jest ograniczony do użytkowników posiadających odpowiednie uprawnienia (USER, ADMIN).
* **PasswordEncoder:** Jest to komponent odpowiedzialny za szyfrowanie haseł użytkowników przed ich zapisaniem w bazie danych. Wykorzystanie enkodera haseł zwiększa bezpieczeństwo aplikacji, chroniąc dane użytkowników przed potencjalnymi atakami.
* **AuthenticationManager:** Jest centralnym punktem zarządzania uwierzytelnianiem w aplikacji. Koordynuje on procesy związane z weryfikacją danych logowania użytkowników, integrując różne mechanizmy uwierzytelniania, takie jak tokeny JWT.

Dzięki tej konfiguracji aplikacja uzyskuje solidne zabezpieczenie przed nieautoryzowanym dostępem, co jest kluczowe w przypadku systemów zarządzania projektami, gdzie przechowywane i przetwarzane są wrażliwe dane użytkowników oraz informacje projektowe. Implementacja takich mechanizmów bezpieczeństwa, jak opisane powyżej, pozwala na stworzenie środowiska, które jest zarówno bezpieczne, jak i wydajne, zapewniając użytkownikom możliwość swobodnego korzystania z funkcjonalności systemu bez obaw o bezpieczeństwo swoich danych.

Dodatkowo wykorzystana jest migracja bazy danych za pomocą Liquibase.  
Liquibase to narzędzie open-source służące do zarządzania zmianami w schematach baz danych, szczególnie przydatne w środowiskach CI/CD, gdzie umożliwia automatyzację procesu wprowadzania zmian w bazie danych równolegle z aktualizacjami aplikacji. Dzięki Liquibase, zmiany w bazie danych mogą być definiowane w uporządkowany sposób (za pomocą formatów takich jak XML, YAML, JSON lub SQL) i śledzone w systemach kontroli wersji, takich jak Git. [27]

W projekcie Liquibase jest wykorzystywany do zarządzania ewolucją schematu bazy danych. Struktura projektu obejmuje pliki XML, które odpowiadają za tworzenie tabel, takich jak project, task, user, role, i comment, co jest widoczne w katalogu resources/db/changelog. Każdy plik XML odpowiada konkretnemu zestawowi zmian (changeset), który Liquibase aplikuje w określonej kolejności.

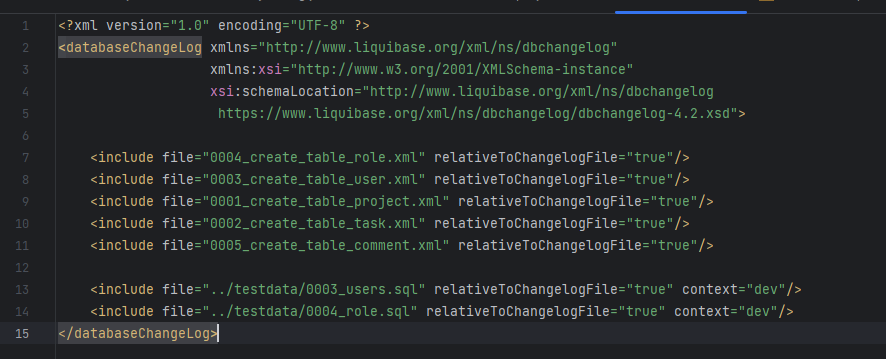


Rysunek 13 Implementacja Liquibase [13]

Zaprezentowany plik XML (0001\_create\_table\_project.xml) to przykład changesetu w Liquibase, który tworzy tabelę project. W pliku określone są kolumny takie jak id, name, description, start\_date, end\_date oraz user\_id, a także związane z nimi ograniczenia, takie jak klucze główne i obce. Ten changeset zapewnia, że tabela project zostanie utworzona z odpowiednią strukturą i relacjami, co jest realizowane przez tag addForeignKeyConstraint, który łączy kolumnę user\_id z kolumną id w tabeli users.

Dzięki użyciu Liquibase zapewniamy że schemat bazy danych jest wersjonowany, śledzony i może być automatycznie aktualizowany w różnych środowiskach, co zmniejsza ryzyko błędów i niespójności. Tego rodzaju automatyzacja i kontrola wersji są kluczowe dla utrzymania spójności bazy danych z rozwijającym się kodem aplikacji, co jest nieocenione w profesjonalnym zarządzaniu projektami IT.

Ważną kwestią jest również plik master.xml. Plik master.xml w Liquibase pełni kluczową rolę jako centralny punkt zarządzania migracjami bazy danych w projekcie. Jest to nadrzędny plik, który koordynuje uruchamianie poszczególnych zmian strukturalnych bazy danych, takich jak tworzenie tabel czy definiowanie relacji między nimi. W projekcie, plik master.xml odwołuje się do szeregu innych plików XML, które zawierają szczegółowe instrukcje dotyczące tworzenia konkretnych tabel, na przykład: project, task, user, role, oraz comment.[27]



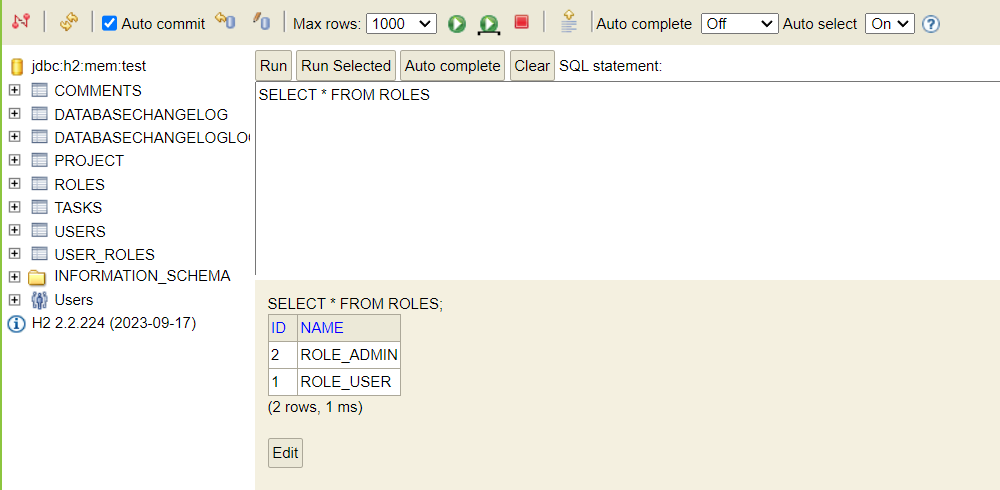
Rysunek 14 Implementacja pliku master.xml

W projekcie oprócz definiowania struktury bazy danych za pomocą plików ‘changelog’, istotną rolę odgrywają dane testowe, które są ładowane do bazy danych w środowisku developerskim. Dane te zapisane są w plikach SQL, takich jak 0003\_role.sql. Umożliwiają one szybkie zapełnienie bazy danych kluczowymi rekordami, które są niezbędne do przeprowadzania testów funkcjonalnych aplikacji.



Rysunek 15 Implementacja danych testowych [13]

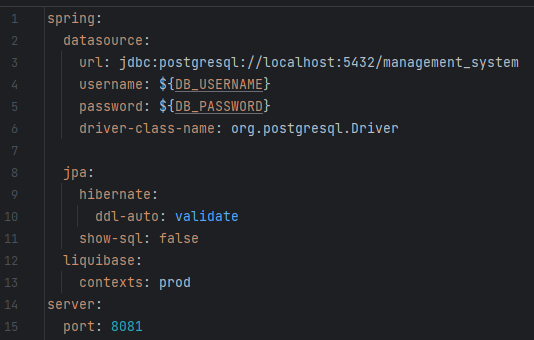
W trybie developerskim wykorzystywana jest baza danych H2, która jest lekka i łatwa do skonfigurowania. Dzięki niej, dane testowe mogą być szybko ładowane i wykorzystywane bez konieczności uruchamiania pełnoprawnej, zewnętrznej bazy danych. Baza H2 pozwala na przeprowadzenie testów w izolowanym środowisku, co znacząco przyspiesza procesy rozwojowe – umożliwia szybkie iterowanie nad kodem, jednocześnie zapewniając wysoką jakość i stabilność tworzonych funkcjonalności.[19]



Rysunek 16 Implementacja bazy H2

Kluczowe jest zarządzanie konfiguracją w sposób, który umożliwia łatwe i bezpieczne dostosowanie aplikacji do różnych środowisk (np. deweloperskiego, testowego, produkcyjnego). W tym celu często wykorzystuje się pliki konfiguracyjne w formacie YAML oraz zmienne środowiskowe. Plik YAML jest popularnym formatem, który pozwala na zdefiniowanie konfiguracji w sposób przejrzysty i łatwy do edycji. Dzięki swojej prostocie i czytelności, YAML jest szeroko stosowany do definiowania parametrów konfiguracji, które mogą się różnić w zależności od środowiska, w którym aplikacja jest uruchamiana.[20]

Zmienne środowiskowe z kolei pozwalają na dynamiczne przekazywanie wartości do aplikacji w trakcie jej uruchamiania. Są one szczególnie przydatne do przechowywania poufnych informacji, takich jak hasła, które nie powinny być bezpośrednio zapisane w plikach konfiguracyjnych.[21]



Rysunek 17 Implementacja pliku YAML [13]

Plik YAML przedstawia konfigurację aplikacji Spring Boot, która jest dostosowana do środowiska produkcyjnego. Poniżej znajduje się szczegółowa interpretacja poszczególnych sekcji tego pliku:

**Baza danych (Datasource):**

* Aplikacja korzysta z bazy danych PostgreSQL, co jest widoczne w polu url. Jest ona uruchomiona na lokalnym serwerze, na porcie 5432, a baza danych nazywa się management\_system.
* Dane logowania do bazy danych, takie jak nazwa użytkownika i hasło, są przechowywane jako zmienne środowiskowe (${DB\_USERNAME} oraz ${DB\_PASSWORD}). Dzięki temu są one bezpiecznie oddzielone od kodu i można je zmieniać bez potrzeby modyfikacji pliku YAML.

**Hibernate:**

* Hibernate jest używany do zarządzania encjami JPA. W produkcji ustawienie ddl-auto: validate powoduje, że Hibernate weryfikuje, czy schemat bazy danych jest zgodny z encjami, ale nie dokonuje żadnych modyfikacji bazy danych.
* Wyłączono logowanie zapytań SQL (show-sql: false), co jest zalecane w środowisku produkcyjnym, aby nie zalewać logów.

**Liquibase:**

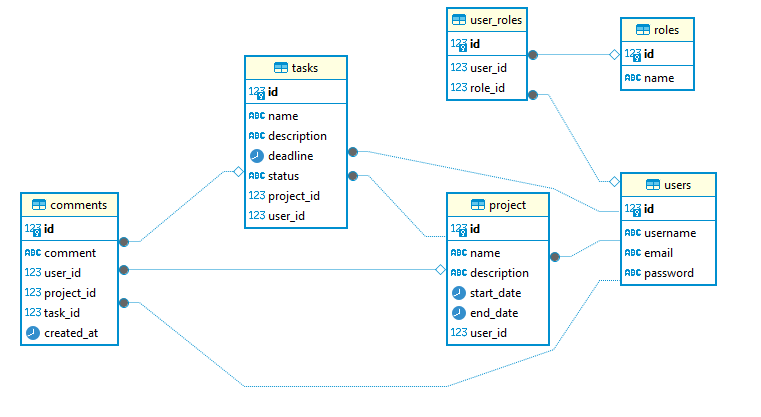
Liquibase to narzędzie do migracji schematu bazy danych. W konfiguracji zdefiniowano, że migracje będą uruchamiane w kontekście produkcyjnym (contexts: prod).

**Serwer:**

Aplikacja zostanie uruchomiona na porcie 8081. Jest to inne ustawienie niż domyślny port 8080, co może być użyteczne np. do uniknięcia konfliktu z innymi aplikacjami.

* 1. **Wybór i zastosowanie PostreSQL jako systemu bazy danych**

PostgreSQL jest jednym z najbardziej zaawansowanych systemów baz danych dostępnych na rynku, oferującym szeroką gamę funkcji, które sprawiają, że jest doskonałym wyborem zarówno dla prostych aplikacji, jak i złożonych systemów produkcyjnych. Jednym z głównych powodów wyboru PostgreSQL jest jego zgodność ze standardami SQL oraz otwartość na rozszerzenia, co pozwala na dostosowanie bazy danych do specyficznych potrzeb projektu. PostgreSQL zapewnia pełną transakcyjność zgodną z zasadami ACID, co gwarantuje integralność danych, nawet w przypadku awarii systemu. To sprawia, że jest idealnym rozwiązaniem dla aplikacji, które wymagają niezawodności i spójności danych. Wysoka wydajność PostgreSQL, wspierana przez zaawansowane mechanizmy optymalizacji zapytań i równoległego przetwarzania, umożliwia obsługę dużej liczby jednoczesnych połączeń oraz przetwarzanie złożonych operacji w czasie rzeczywistym. Dodatkowo, PostgreSQL jest systemem otwartoźródłowym, co oznacza brak kosztów licencyjnych oraz dostęp do szerokiej społeczności użytkowników i programistów, którzy regularnie przyczyniają się do rozwoju i wsparcia tego systemu. Dzięki temu PostgreSQL jest nie tylko ekonomicznym wyborem, ale również technologią, która stale się rozwija i dostosowuje do najnowszych standardów branżowych.[22]



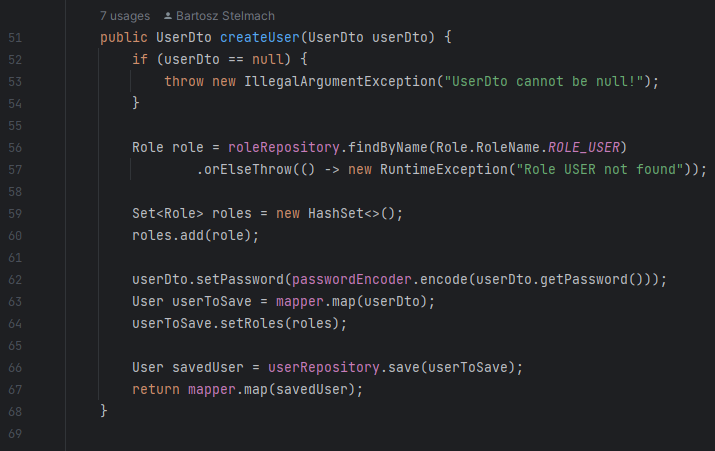
Rysunek 18 Schemat bazy danych [13]

1. **Implementacja**

W dziale "Implementacja" skupiamy się na praktycznym wdrożeniu kluczowych funkcji systemu, konfiguracji narzędzi wspomagających procesy wytwórcze oraz na zarządzaniu zależnościami i budową projektu. Główne aspekty tego etapu obejmują implementację funkcjonalności, które są niezbędne do poprawnego działania systemu, a także zautomatyzowanie procesów testowania i wdrażania za pomocą narzędzia Jenkins. Ponadto, wykorzystanie Mavena do zarządzania zależnościami oraz automatyzacji procesu budowy projektu stanowi integralną część tego etapu, zapewniając płynność i efektywność pracy zespołu developerskiego. Implementacja obejmuje również integrację z systemem kontroli wersji, co umożliwia łatwe śledzenie zmian w kodzie oraz sprawne wdrażanie aktualizacji.

* 1. **Implementacja kluczowych funkcjonalności systemu**

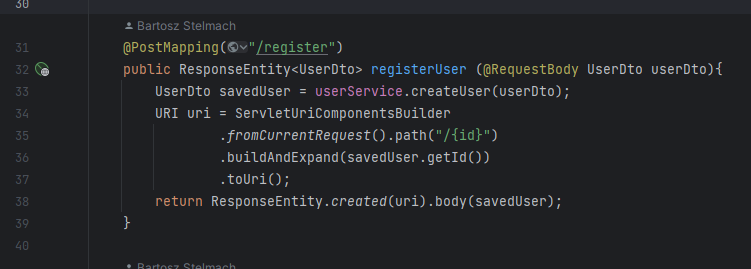
Kluczową funkcjonalnością w systemie zarządzania projektami jest zarządzanie użytkownikami, które obejmuje rejestrację, uwierzytelnianie, oraz przypisywanie odpowiednich ról użytkownikom.



Rysunek 19 Implementacja logiki tworzenia użytkownika [13]

W tej części kodu mamy metodę createUser, która odpowiada za tworzenie nowych użytkowników w systemie. Cały proces zaczyna się od sprawdzenia, czy przekazany obiekt UserDto nie jest null, co zabezpiecza przed niepoprawnymi danymi wejściowymi. Jeśli użytkownik jest prawidłowy, serwis wyszukuje rolę "ROLE\_USER" z repozytorium RoleRepository. W przypadku, gdy rola nie zostanie znaleziona, metoda rzuca wyjątek RuntimeException, co zapewnia, że każdy nowy użytkownik będzie miał przypisaną odpowiednią rolę w systemie.

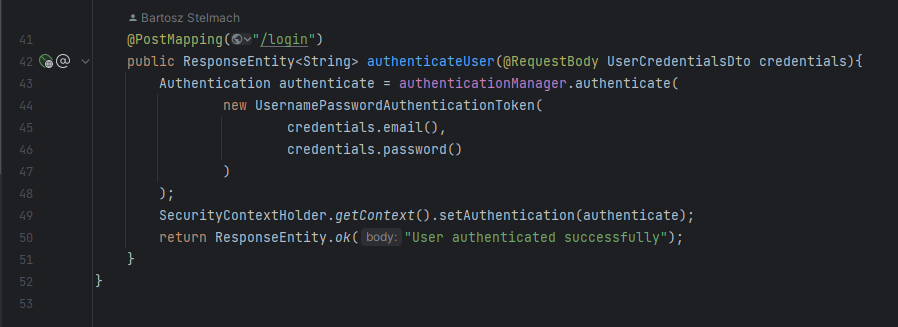
Następnie rola "ROLE\_USER" jest dodawana do zbioru ról przypisanego użytkownikowi. Hasło użytkownika jest hashowane za pomocą PasswordEncoder, co zapewnia bezpieczeństwo przechowywanych danych. Obiekt UserDto jest następnie mapowany na encję User, co umożliwia zapisanie go w bazie danych za pomocą UserRepository. Ostatecznie metoda zwraca zapisany obiekt użytkownika, ponownie zmapowany na UserDto. Kod ten jest kluczowy dla mechanizmu rejestracji użytkowników w systemie, zapewniając zarówno bezpieczeństwo (poprzez hashowanie hasła), jak i integralność danych (poprzez przypisanie odpowiednich ról).



Rysunek 20 Implementacja rejestracji użytkownika [13]

Metoda registerUser odpowiada za rejestrację nowego użytkownika. Przyjmuje ona jako argument dane użytkownika w postaci obiektu UserDto, który jest adnotowany adnotacją **@RequestBody**, co oznacza, że dane te są przesyłane w ciele żądania HTTP. Metoda wywołuje następnie serwis UserService, który za pomocą metody createUser tworzy nowego użytkownika w bazie danych.

Po pomyślnym zapisaniu użytkownika, metoda zwraca status HTTP 201 Created oraz URI nowo utworzonego użytkownika. Warto zwrócić uwagę na to, że adres URI jest generowany dynamicznie przy pomocy ServletUriComponentsBuilder, co jest dobrą praktyką w RESTful API, umożliwiającą klientom łatwe odnalezienie zasobów, które zostały stworzone.

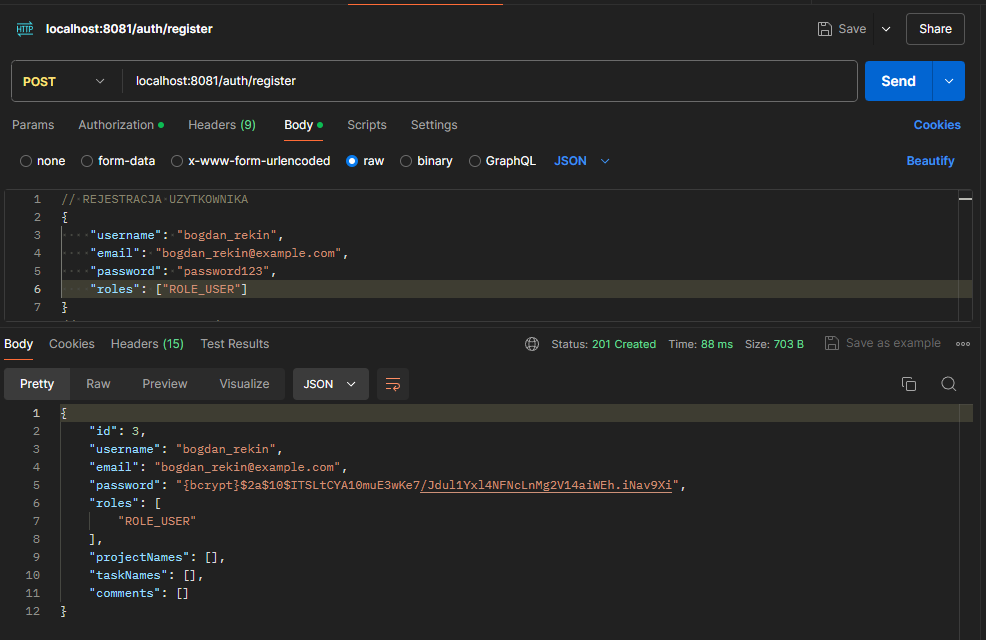


Rysunek 21 Implementacja logowania [13]

Metoda authenticateUser odpowiada za logowanie użytkownika. Przyjmuje ona dane uwierzytelniające w postaci UserCredentialsDto, który zawiera email i hasło. Następnie, przy pomocy AuthenticationManager, metoda sprawdza, czy podane dane są poprawne i czy użytkownik może zostać uwierzytelniony.

Jeśli dane są poprawne, użytkownik zostaje zalogowany, a kontekst bezpieczeństwa SecurityContextHolder jest odpowiednio ustawiony. Na koniec metoda zwraca odpowiedź z komunikatem o pomyślnym zalogowaniu.

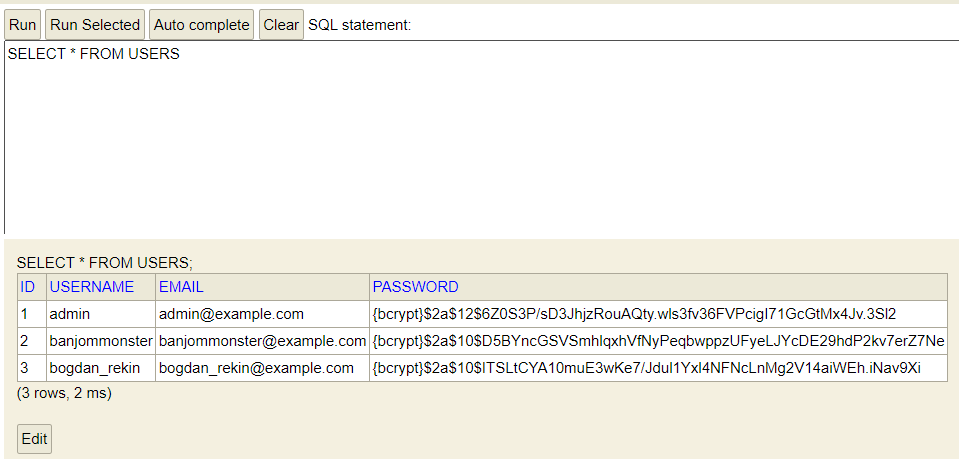
Ten kontroler jest kluczowym elementem mechanizmu zarządzania użytkownikami w aplikacji, obsługując dwa fundamentalne procesy: rejestrację oraz logowanie, które są niezbędne do zapewnienia bezpieczeństwa i zarządzania dostępem w systemie.



Rysunek 22 Rejestracja użytkownika [13]

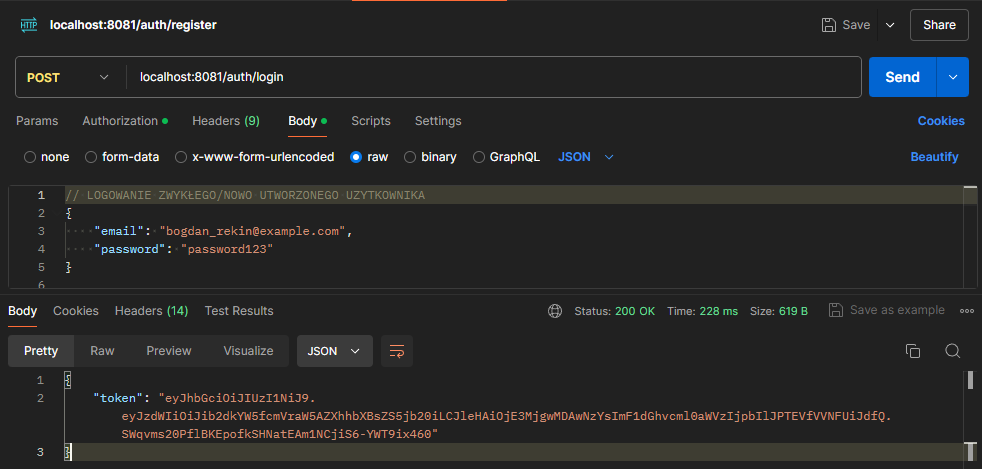
Rysunek 22 przedstawia przykład rejestracji nowego użytkownika za pomocą Postmana w aplikacji webowej. Żądanie typu POST zostało wysłane na endpoint localhost:8081/auth/register, który odpowiada za rejestrację użytkownika w systemie. W sekcji Body wysłano dane użytkownika w formacie JSON. W odpowiedzi serwer zwrócił status 201 Created, co oznacza, że użytkownik został pomyślnie zarejestrowany. W odpowiedzi znajduje się również zaszyfrowane hasło użytkownika oraz przypisane do niego role, a także puste listy projektów, zadań i komentarzy, które mogą być uzupełnione później w trakcie korzystania z aplikacji.

Taki zapis demonstruje, że rejestracja przebiegła poprawnie i użytkownik został dodany do bazy danych.



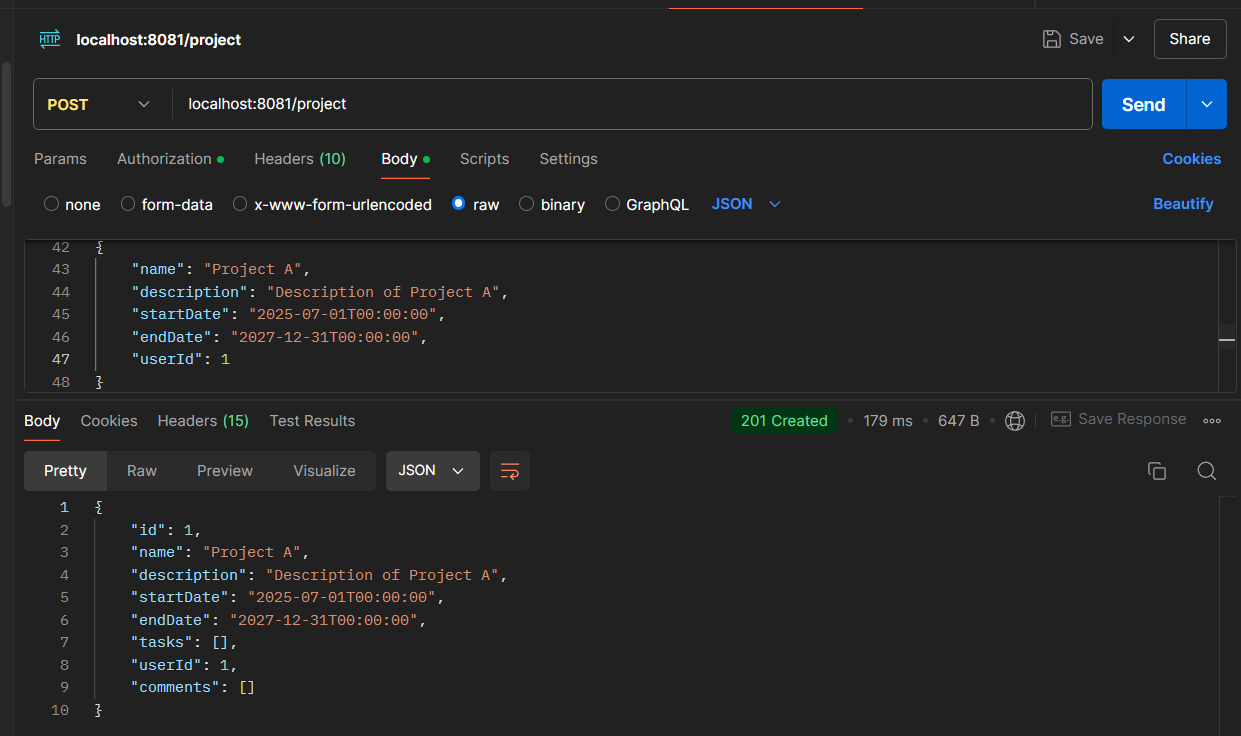
Rysunek 23 Zapisany użytkownik do bazy danych [13]

Teraz podczas logowania się do systemu użytkownika, po poprawny uwierzytelnieniu, serwer zwraca odpowiedź w formie tokenu JWT. Token jest używany w systemie do autoryzacji kolejnych żądań, na przykład dostępu do chronionych zasobów.



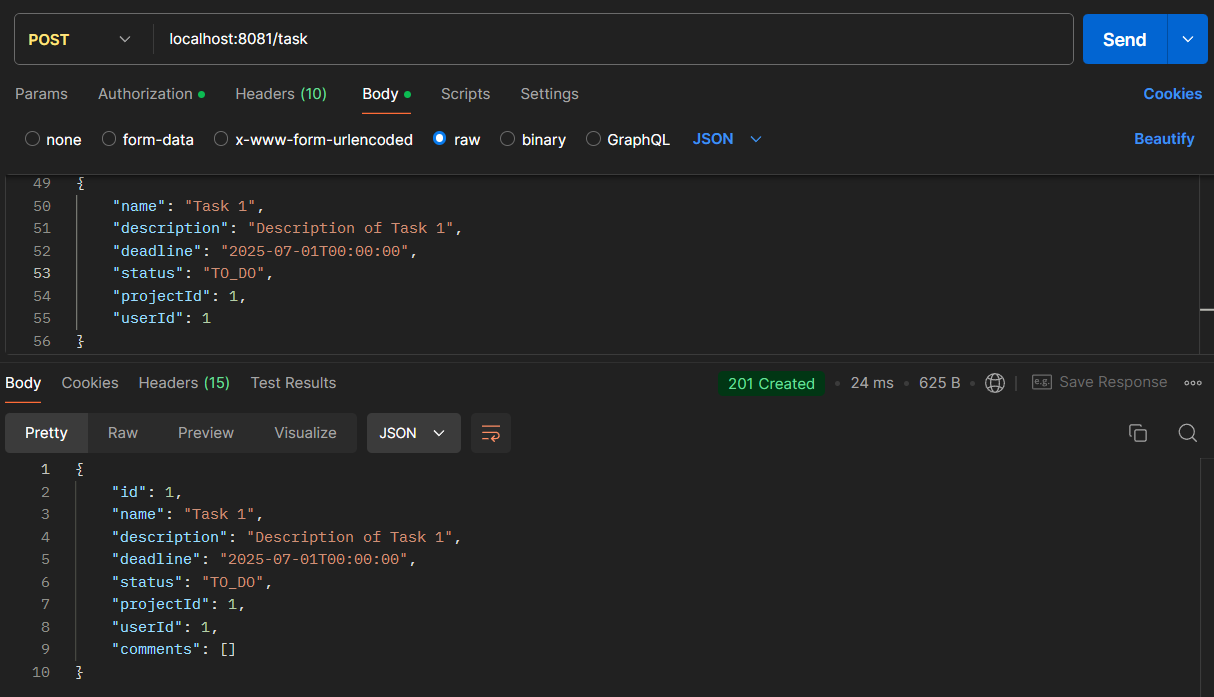
Rysunek 24 Poprawne uwierzytelnienie użytkownika [13]

Niemniej ważną funkcją jest zarządzanie projektami i zadania. Dlatego po tym jak uzyskaliśmy token możemy przejść do funkcji tworzenia, przypisywania czy komentowania projektów i zadań. Dzięki temu pozwalamy użytkownikom na śledzenie postępów ale także zobaczenie czego jeszcze im brakuje.



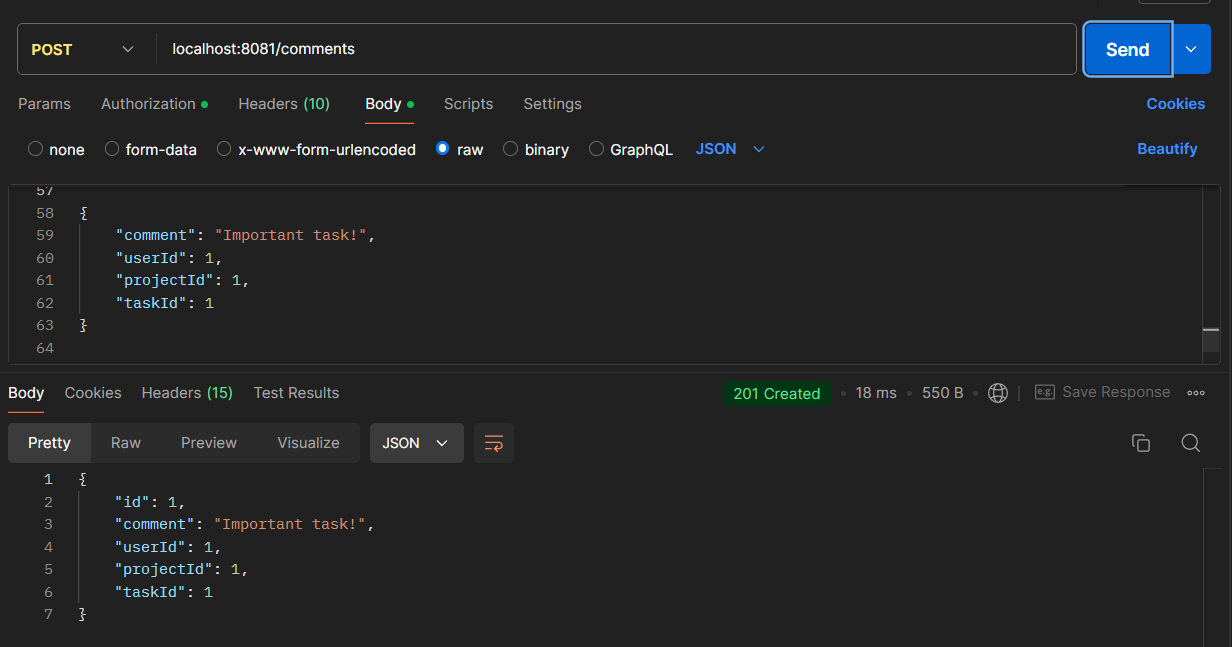
Rysunek 25 Utworzenie nowego projektu [13]

Tutaj widzimy proces tworzenie nowego projektu poprzez aplikacje Postman. Żądanie skierowane pod endpoint */project* z danymi w formacie JSON tworzy nowy projekt, do którego jeszcze nie ma przypisanych zadań. W odpowiedzi na wysłane żądanie API zwraca status 201 Created oraz szczegóły nowo utworzonego projektu.



Rysunek 26 Utworzenie nowego zadania [13]

Kolejny krokiem było utworzenie nowego zadania, które zostało skutecznie przypisane do dane projektu. Jest też jasno określone do którego użytkownika się to odnosi i kto jest za nie odpowiedzialny.

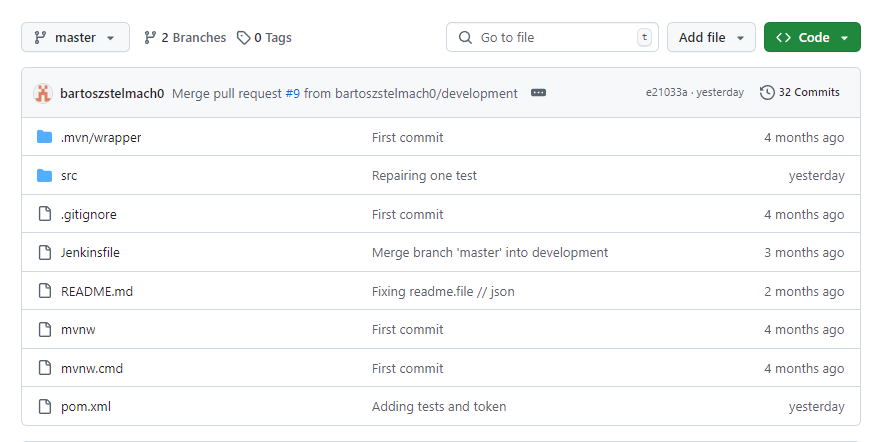


Rysunek 27 Wstawienie komentarzy do projektów i zadań [13]

Jak zostało to opisane wyżej jest możliwość dodania komentarza do określonego projektu z powiązanym zadaniem. Należy również pamiętać, iż każdy wykonany krok powyżej został od razu zapisany do bazy danych, w której są przechowywane wszystkie dane.

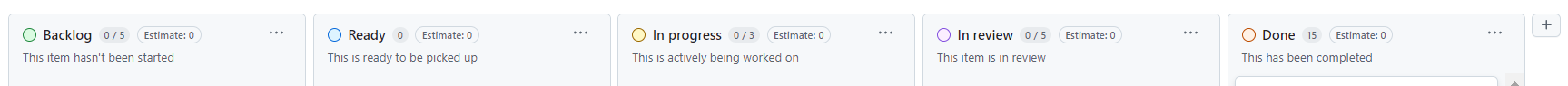
* 1. **Umieszczenie kodu w zdalnym repozytorium**

Jedną z najważniejszy rzeczy podczas budowy aplikacji jest umieszczenie kodu źródłowego w zdalnym repozytorium Git, które pozwala na łatwą współpracę między członkami zespołu oraz śledzenie zmian w kodzie. Jest to również niezbędne do późniejszego tworzenia automatyzacji do której nie będzie potrzebny człowiek. Zdalne repozytorium umożliwia kontrolę wersji, co jest szczególnie istotne podczas rozwijania aplikacji, gdyż pozwala na wygodne zarządzanie gałęziami (branchami) oraz szybkie cofanie zmian w razie wystąpienia błędów. Repozytorium został utworzone na platformie GitHub ponieważ gwarantuje ona wiele funkcji dodatkowych, które ułatwiają pracę programiście.



Rysunek 28 Kod na platformie GitHub[13]

GitHub oferuje nie tylko miejsce na przechowywanie i zarządzanie kodem, ale także dodatkowe funkcje, które wspierają zarządzanie projektem, takie jak tablica zadań (Project Board).

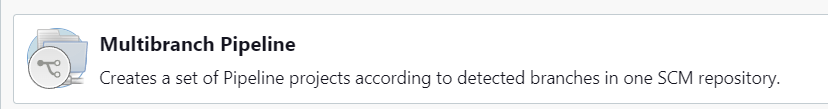


Rysunek 29 Tablica zadań [13]

Dzięki temu, że posiada ona różne opcje może ona ułatwić planowanie przyszłych zadań ale również pokazuje co zostało już osiągnięte w projekcie. Te funkcje pozwalają lepiej koordynować zadania, a osoby zamieszane w dany projekt maja pełny obraz nad postępami w czasie rzeczywistym.

* 1. **Konfiguracja Jenkinsa do automatycznego testowania i deploymentu**

W pracy dyplomowej wykorzystano narzędzie Jenkins do automatycznego testowania aplikacji oraz jej wdrażania.



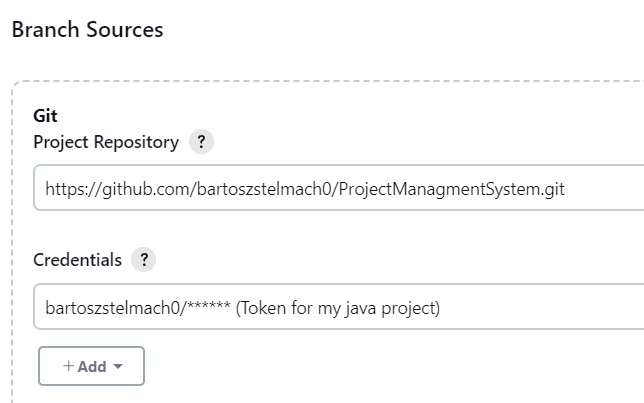
Rysunek 30 Rodzaj pipelineu [13]

Bardzo ważnym aspektem jest wybór „Multibranch, ponieważ automatycznie wykrywa i tworzy oddzielne pipeliny dla każdej gałęzi repozytorium w systemie kontroli wersji. To pozwala na testowanie i budowanie kodu dla różnych gałęzi w sposób równoległy, co zwiększa efektywność procesu CI/CD. Dodatkowo, Multibranch Pipeline zapewnia lepszą integrację z systemami takimi jak Git, umożliwiając automatyczne uruchamianie zadań w odpowiedzi na zmiany w każdej gałęzi, co eliminuje konieczność ręcznego definiowania i utrzymywania pipeline dla każdej z nich.



Rysunek 31 Fragment skryptu Jenkinsfile [13]

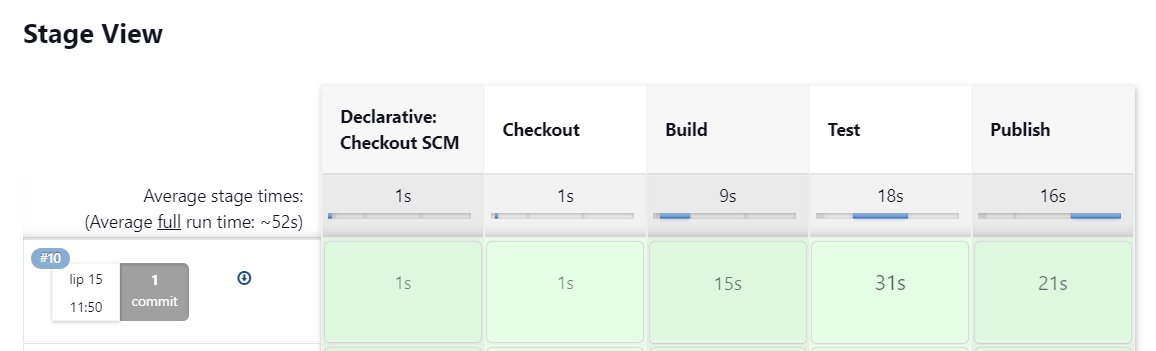
Ten fragment kodu przedstawia etap "Checkout" w skrypcie Jenkins Pipeline, który odpowiada za pobranie kodu źródłowego z repozytorium Git. Wykorzystuje klasę GitSCM do skonfigurowania połączenia z repozytorium GitHub za pomocą podanego adresu URL oraz identyfikatora poświadczeń (credentialsId). Dzięki temu Jenkins może automatycznie pobierać najnowsze zmiany z repozytorium przed przystąpieniem do budowy, testowania i publikacji projektu.



Rysunek 32 Fragment konfiguracji uwierzytelnienia w Jenkinsie [13]

Uwierzytelnienie musi być zrobione w dwóch miejscach. Jedno w kodzie, tak jak pokazano to na rysunek 31. Drugi zaś w samym Jenkinsie. Taka konfiguracja pozwala nam na efektywną pracę z oboma środowiskami.

Kolejną rzeczą są etapy wykonywania się w pipeline.



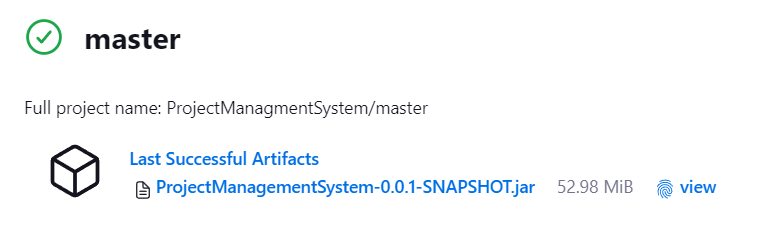
Rysunek 33 Etapy w Jenkinsie [13]

**Checkout –** czyli pobieranie najnowszej wersji kodu z repozytorium GitHub, używając zdalnego repozytorium skonfigurowanego w Jenkinsie.

**Build -**  Projekt jest kompilowany za pomocą Mavena, który zarządza zależnościami i procesem budowania.

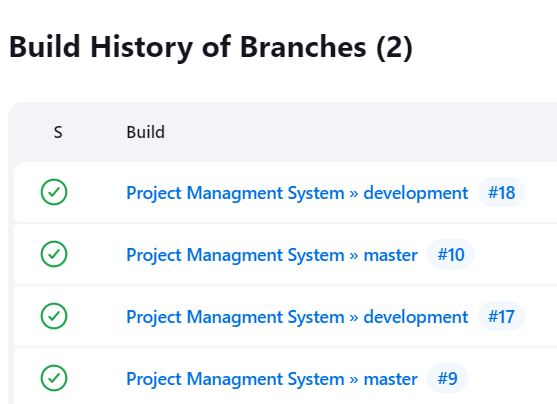
**Test-** Jenkins uruchamia testy jednostkowe, analizując ich wyniki, a następnie raportuje je w formacie JUnit.

**Publish-** Gdy testy zakończą się sukcesem, artefakty (np. plik .jar) są publikowane i przechowywane w Jenkinsie.



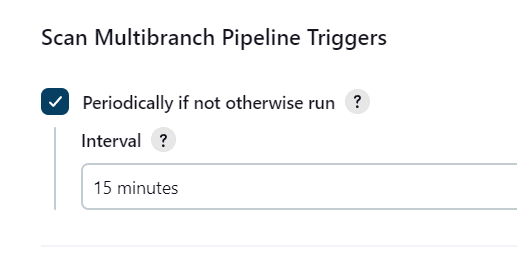
Rysunek 34 Zbudowany artefakt [13]

Na końcu procesu budowania i testowania projektu w Jenkinsie uzyskujemy artefakt w formie pliku JAR (jak pokazano na obrazie). Ten artefakt zawiera skompilowaną wersję aplikacji, gotową do wdrożenia. Możemy go pobrać bezpośrednio z Jenkinsa po każdym udanym cyklu pipeline, co pozwala na łatwą dystrybucję i wdrożenie aplikacji na różnych środowiskach, np. testowych, stagingowych czy produkcyjnych. Taki plik JAR można uruchomić jako samodzielną aplikację Spring Boot lub wdrożyć na serwerze aplikacyjnym.



Rysunek 35 Historia pipelinenów [13]

Na załączonym zrzucie ekranu widzimy historię buildów dla różnych gałęzi projektu w Jenkinsie, takich jak master i development. Każdy build jest oznaczony numerem (np. #10, #18) i wskazuje, czy zakończył się sukcesem (zielony znaczek). Taka historia jest przydatna, ponieważ umożliwia śledzenie wszystkich przeprowadzonych buildów dla każdej gałęzi kodu. Dzięki temu zespół deweloperski może szybko zidentyfikować, kiedy i gdzie pojawiły się błędy oraz który build był ostatnim działającym. W razie problemów można wrócić do wcześniejszej, stabilnej wersji builda.



Rysunek 36 Ustawienie triggera w Jenkinsie [13]

Skonfigurowanie opcji „Scan Multibranch Pipeline Triggers” w Jenkinsie z interwałem skanowania co 15 minut pozwala na automatyczne przeszukiwanie repozytorium Git w poszukiwaniu nowych zmian w różnych gałęziach projektu. Oznacza to, że co 15 minut Jenkins automatycznie wykryje nowe commity lub nowe gałęzie, a następnie uruchomi odpowiednie procesy budowania (pipeline) dla zaktualizowanych gałęzi. Taka konfiguracja pomaga w szybszym wykrywaniu i testowaniu zmian, co jest kluczowe w projektach, gdzie częste aktualizacje są standardem. Teoretycznie istnieje możliwość ustawienie go na krótszy okres czasu, jednak czasami ze względu na wielkość danej aplikacji nie zdąży wykonać się jeden pełny cykl, po czym zaczyna się kolejny co może skutkować narastającą ilością błędów. Dlatego najbezpieczniejszym rozwiązaniem jest ustawienie intervala na okres 15 minut.

* 1. **Użycie Mavena do zarządzania zależnoścami i budowy projektu**

Maven jest popularnym narzędziem do zarządzania zależnościami oraz budowania projektów opartych na Javie. Jego głównym zadaniem jest uproszczenie procesu zarządzania bibliotekami i pakietami, które aplikacja potrzebuje do działania, oraz automatyzacja procesu kompilacji, testowania i wdrażania aplikacji. W projekcie wykorzystującym Spring Boot, Maven odgrywa kluczową rolę w zarządzaniu zewnętrznymi zależnościami, jak również w zapewnieniu spójności w procesie budowy aplikacji. Plik pom.xml (Project Object Model) to centralny element konfiguracji projektu Mavena. W tym pliku deklarowane są wszystkie zależności, które projekt wykorzystuje, a także dodatkowe informacje, takie jak wersja Javy, pluginy, które są wykorzystywane w procesie budowy oraz konfiguracje do uruchamiania testów. [8]



Rysunek 37 Wybrane zależności w Mavenie [13]

Spring Boot: W projekcie używane są różne moduły Spring Boot, takie jak spring-boot-starter-web do obsługi REST API, spring-boot-starter-security do zarządzania autoryzacją i autentykacją, oraz spring-boot-starter-data-jpa do integracji z bazą danych za pomocą JPA. Te zależności pozwalają na szybkie i efektywne zarządzanie różnymi aspektami działania aplikacji, takimi jak obsługa zapytań HTTP, zarządzanie sesjami użytkowników, czy dostęp do bazy danych.

1. **Testowanie**

Testowanie oprogramowania jest nieodłącznym elementem procesu tworzenia systemów informatycznych, szczególnie w kontekście dużych projektów, gdzie niezawodność i stabilność aplikacji są kluczowe. Testowanie pozwala na identyfikację błędów w kodzie, sprawdzenie poprawności działania poszczególnych funkcji oraz zapewnienie, że system spełnia założenia funkcjonalne i niefunkcjonalne. Bez odpowiedniego testowania niemożliwe byłoby zidentyfikowanie wielu problemów przed wdrożeniem aplikacji na środowisko produkcyjne, co mogłoby prowadzić do awarii i utraty zaufania użytkowników. W projekcie, który został tutaj wyżej przedstawiony zostały wykorzystane narzędzia takie jak Mockito, JUnit czy AssertJ do pisania testów jednostykowych oraz integracyjnych., które są potem automatycznie uruchamiane w ramach procesu Continous Integration (CI) za pomocą Jenkinsa [23]. W pracy zostało zastosowane podejście TDD, czyli Test-Driven Development. Test-Driven Development (TDD) to podejście do tworzenia oprogramowania, w którym testy są pisane przed implementacją kodu. Proces TDD składa się z trzech głównych etapów:

Pisanie testu: Na początku deweloper tworzy prosty test, który definiuje nową funkcjonalność lub poprawkę. Ten test na początku się nie powiedzie, ponieważ nie ma jeszcze odpowiedniej implementacji.

Pisanie kodu: Następnie programista pisze minimalny kod, który pozwala przejść napisanemu wcześniej testowi. W tej fazie kod może być bardzo podstawowy, a nawet tymczasowy.

Refaktoryzacja: Po przejściu testu deweloper refaktoryzuje kod, usuwając ewentualne redundancje i poprawiając jego strukturę, przy jednoczesnym zachowaniu pozytywnego wyniku testu.

W ten sposób to podejście zmusza do myślenia o poprawności od samego początku, co prowadzi do bardziej niezawodnych i łatwiejszych do utrzymania systemów.[23]

Zostało również wykorzystane podejście BDD. Behavior-Driven Development (BDD) to podejście, które rozwija TDD, skupiając się na definiowaniu zachowań systemu zamiast jego funkcji. W BDD testy są pisane w formie opisów zachowań systemu w naturalnym języku, co ułatwia komunikację między deweloperami, testerami i interesariuszami. W BDD używa się scenariuszy, które opisują jak system powinien się zachowywać w odpowiedzi na różne działania użytkownika. Przykładowe scenariusze BDD zapisane są często według schematu Given-When-Then, gdzie:

* **Given** – opisuje stan początkowy systemu,
* **When** – określa akcję wykonaną przez użytkownika,
* **Then** – definiuje oczekiwany wynik.

BDD pomaga skupić się na tym, jak aplikacja ma działać z perspektywy użytkownika końcowego, co jest istotne szczególnie w projektach o wysokich wymaganiach biznesowych.[28]

Każda klasa była również sprawdzana i testowana pod kątem Test coverage. Test coverage (pokrycie testami) to miara określająca, jaka część kodu jest testowana przez zestaw testów automatycznych. Wyższe pokrycie testami oznacza, że większa liczba scenariuszy i przypadków użycia jest sprawdzana przez testy, co zwiększa pewność, że aplikacja działa poprawnie. Test coverage można mierzyć na różne sposoby:

* Pokrycie linii kodu – mierzy, ile linii kodu zostało wykonanych podczas testów.
* Pokrycie instrukcji warunkowych – mierzy, ile instrukcji warunkowych (np. if-else) zostało sprawdzonych.
* Pokrycie gałęzi – mierzy, czy każda możliwa ścieżka wykonania w kodzie (np. każda gałąź w instrukcji warunkowej) została przetestowana.[29]

W Javie do mierzenia pokrycia testami często używa się narzędzi takich jak JaCoCo (Java Code Coverage), które generują szczegółowe raporty na temat pokrycia kodu.

Wszystkie te metody wspólnie pomagają w stworzeniu stabilnej i niezawodnej aplikacji, redukując ryzyko wystąpienia błędów w kodzie produkcyjnym.

* 1. **Planowanie testów (Integracyjne oraz Jednostkowe)**

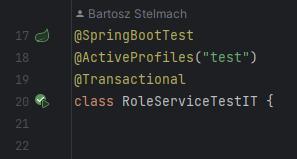
Testy integracyjne to rodzaj testów oprogramowania, których celem jest sprawdzenie, czy poszczególne moduły lub komponenty systemu współpracują ze sobą poprawnie. W odróżnieniu od testów jednostkowych, które testują pojedyncze funkcje lub klasy w izolacji, testy integracyjne skupiają się na weryfikacji współdziałania tych elementów w większym kontekście. Testy integracyjne w Javie są często wspierane przez odpowiednie adnotacje, które pomagają skonfigurować i uruchomić testy w odpowiednim środowisku. Oto kilka kluczowych adnotacji, które są używane przy tworzeniu testów integracyjnych: [24]

**@SpringBootTes**t – najważniejsza adnotacja, która uruchamia pełny kontekst Spring Boot. Umożliwia testowanie integracji aplikacji, uruchamiając całą aplikację lub jej część w pełnym środowisku testowym. Adnotacja ta automatycznie konfiguruje wszystkie niezbędne komponenty, jak np. warstwa dostępu do danych, warstwa serwisów, a także konteksty bazy danych.[30]

**@Transactional -**  zapewnia, że każda zmiana w bazie danych w trakcie testu jest cofana po zakończeniu testu. Dzięki temu każdy test jest izolowany i nie wpływa na inne testy poprzez pozostawienie danych w bazie.

**@ActiveProfiles -** jest używana w Spring Boot do określania profili aktywnych w trakcie uruchamiania aplikacji lub testów. Profile w Spring Boot to mechanizm, który pozwala na zarządzanie różnymi konfiguracjami dla różnych środowisk, takich jak dev, test, prod.

Te wyżej adnotację zostały użyte w projekcie podczas pisania testów integracyjnych.

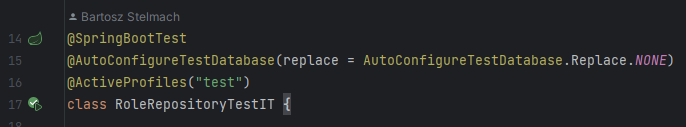


Rysunek 38 Adnotacje w testach integracyjnych [13]

Do testy integracyjnych testujących repozytoria koniecznym jest użyć troszkę innych adnotacji:

**@DataJpaTest -** jest używana do testowania warstwy JPA, czyli operacji związanych z bazą danych. Uruchamia ona jedynie niezbędne komponenty do testowania repozytoriów i ogranicza uruchamianie pozostałych komponentów aplikacji, co skraca czas testowania.

**@AutoConfigureTestDatabase(replace= AutoConfigureTestDatabase.Replace.NONE) -** pozwala na testowanie na rzeczywistej bazie danych (w moim przypadku PostgreSQL).

****

Rysunek 39 Adnotacje w testach integracyjnych w repozytorium [13]

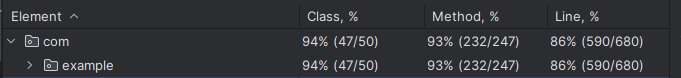
Testy jednostkowe są podstawową techniką testowania w oprogramowaniu, mającą na celu weryfikację poprawności działania poszczególnych fragmentów kodu — zazwyczaj pojedynczych metod lub funkcji. Celem testów jednostkowych jest izolacja najmniejszych fragmentów logiki aplikacji i sprawdzenie, czy działają one zgodnie z oczekiwaniami. W testach jednostkowych testowane jednostki są oddzielone od zależności, które są zamockowane przy pomocy narzędzi takich jak Mockito. Dzięki temu można symulować zachowanie komponentów, z którymi testowana metoda wchodzi w interakcję, co pozwala na przetestowanie samej logiki bez konieczności uruchamiania np. bazy danych czy innych zewnętrznych systemów. Testy jednostkowe są wykonywane automatycznie i często, co sprawia, że są one idealne do ciągłej integracji (CI) i są kluczowe w praktykach takich jak Test Driven Development (TDD).[25]

Testy jednostkowe w Spring Boot wymagają odpowiedniego przygotowania, w tym konfiguracji zależności i adnotacji, które pozwalają na wydajną i izolowaną weryfikację kodu.



Rysunek 40 Mockowanie w testach jednostkowych [13]

W praktyce dobra jakość pokrycia testami zależy od charakteru projektu, ale przyjmuje się, że pokrycie testami na poziomie od 70% do 90% jest uważane za bardzo dobre. W przypadku krytycznych aplikacji, gdzie stabilność i niezawodność są kluczowe (np. w medycynie czy lotnictwie), warto dążyć do wyższych wartości, nawet powyżej 90%.Jednak warto pamiętać, że 100% pokrycie testami nie zawsze oznacza, że kod jest idealny. Nawet jeśli wszystkie linie kodu są przetestowane, jakość testów ma kluczowe znaczenie – testy muszą być dobrze zaprojektowane, aby faktycznie wykrywały potencjalne błędy i anomalie w systemie. Z tego względu istotne jest także zrównoważenie między ilością testów a ich jakością.

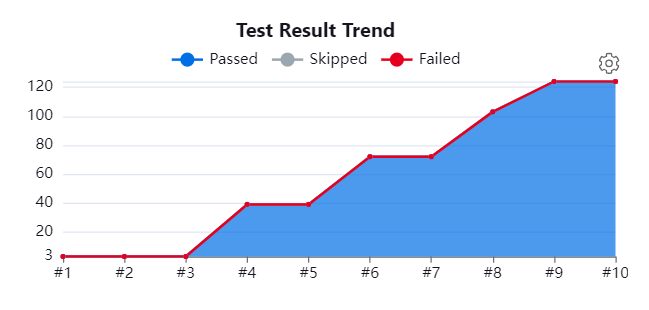


Rysunek 41Test coverage aplikacji [13]

W przypadku danej pracy dyplomowej wskaźniki są bardzo wysokie co może potwierdzać, iż aplikacja jest sprawdzona i przetestowana pod kątem różnych przypadków.

* 1. **Automatyzacja testów z użyciem Jenkinsa**

Automatyzacja testów z użyciem Jenkinsa pozwala na regularne i automatyczne uruchamianie testów po każdej zmianie w kodzie źródłowym. Dzięki temu zespół deweloperski ma pewność, że nowe funkcjonalności nie wprowadzą błędów do istniejącego systemu. Jenkins integruje się z narzędziami do kontroli wersji, takimi jak Git, umożliwiając automatyczne pobieranie zmian w projekcie oraz uruchamianie testów jednostkowych, integracyjnych czy innych. Jenkinsfile definiuje etapy pipeline'u, takie jak pobranie kodu (Checkout), kompilacja (Build), testowanie (Test) i publikacja artefaktów (Publish). Dzięki temu każda zmiana jest dokładnie przetestowana, co pomaga wykrywać błędy na wczesnym etapie. Wyniki testów są prezentowane w formie wykresów.[3]



Rysunek 42 Wykres zautomatyzowanych testów [13]

Taki sposób automatyzacji procesu testowania znacząco zwiększa jakość kodu, pozwala na szybkie wykrywanie regresji oraz sprawia, że proces dostarczania nowych funkcji do projektu staje się bardziej niezawodny i efektywny.

Gdyby zostały wykryte jakiekolwiek błędy w testach, wykres pozwoliłby określić co jest nie tak oraz dałbym o tym znać w Console Output.

**Podsumowanie**

W ramach pracy nad systemem zarządzania projektami omówione zostały kluczowe zagadnienia związane z projektowaniem, implementacją, testowaniem oraz automatyzacją procesów w projekcie. Wprowadzono system oparty na architekturze Spring Boot, wspierany przez mechanizmy takie jak Liquibase do zarządzania bazą danych oraz Maven do zarządzania zależnościami i budowania projektu. Całość kodu została umieszczona w zdalnym repozytorium GitHub, co umożliwia łatwą współpracę i śledzenie zmian w projekcie. Ważnym elementem była również integracja z Jenkins, który odpowiada za automatyczne testowanie i wdrażanie. Dzięki tej integracji możliwe było stworzenie pipeline’u, który po każdej zmianie w kodzie pobiera najnowszą wersję projektu, kompiluje go, uruchamia testy, a w przypadku ich sukcesu, pakuje artefakty i publikuje je. Pozwoliło to na znaczną poprawę wydajności i zwiększenie niezawodności wprowadzanych zmian. Proces testowania odgrywał kluczową rolę w zapewnieniu jakości kodu. Wykorzystano Test Driven Development (TDD) oraz testy jednostkowe i integracyjne, aby pokryć jak największą część kodu i minimalizować ryzyko wprowadzenia błędów. Automatyzacja testów za pomocą Jenkinsa pozwalała na regularne sprawdzanie jakości aplikacji i bieżące monitorowanie wyników testów, co znacząco przyczyniło się do stabilności systemu. Dzięki zastosowaniu odpowiednich metodologii oraz narzędzi, zapewniono wysoki poziom kontroli nad kodem oraz szybką reakcję na ewentualne błędy, co jest kluczowe dla sukcesu projektu w dłuższej perspektywie.

Podczas pracy nad projektem pojawiło się kilka wyzwań, które wymagały szczególnej uwagi i dodatkowego nakładu pracy. Jednym z kluczowych wyzwań było precyzyjne określenie, który fragment kodu powinien być uruchamiany w środowisku deweloperskim, a który w produkcyjnym. Było to szczególnie istotne, aby zapewnić stabilność i spójność aplikacji w obu środowiskach. Kolejnym trudnym aspektem było skonfigurowanie poprawnej integracji GitHuba z Jenkisem przy użyciu tokenów. Proces ten wymagał dodatkowej konfiguracji i rozwiązywania problemów związanych z autoryzacją i zabezpieczeniami, jednak finalnie udało się to z sukcesem zrealizować, co umożliwiło płynną automatyzację testów i wdrożeń.

W celu przezwyciężenia tych trudności korzystałem z oficjalnej dokumentacji narzędzi, takich jak Spring, Jenkins czy GitHub. Szczegółowe analizy dostępnych rozwiązań pomogły mi zrozumieć, jak najlepiej skonfigurować środowiska oraz jak poradzić sobie z wyzwaniami związanymi z autoryzacją. Wykorzystanie dostępnych zasobów online oraz przykładów kodu przyczyniło się do rozwiązania napotkanych problemów oraz do usprawnienia całego procesu integracji i automatyzacji w projekcie.

W kontekście dalszego rozwoju aplikacji jednym z potencjalnych kierunków jest przeniesienie całego systemu do chmury, eliminując potrzebę korzystania z infrastruktury lokalnej (on-prem). Dzięki temu rozwiązaniu możliwe byłoby wykorzystanie takich zalet chmury jak elastyczność, skalowalność oraz lepsza dostępność. Wdrożenie aplikacji w chmurze umożliwia automatyczne rozdzielenie obciążenia między różnymi strefami dostępności (AZ – Availability Zones), co zwiększa odporność systemu na awarie i minimalizuje ryzyko przestojów dzięki lepszym mechanizmom Disaster Recovery (DR). W chmurze łatwo można zarządzać kopiami zapasowymi, autoskalowaniem oraz rozbudową zasobów w miarę wzrostu zapotrzebowania, co sprawia, że aplikacja jest przygotowana na dynamiczne zmiany i rozwój bez potrzeby rozbudowy infrastruktury fizycznej. Dzięki usługom chmurowym możliwe jest także szybkie i bezproblemowe wdrażanie nowych wersji aplikacji oraz zarządzanie nimi z dowolnego miejsca, co dodatkowo ułatwia pracę zespołom deweloperskim.

W ramach dalszego rozwoju aplikacji zarządzania projektami można rozważyć dodanie nowych encji i funkcjonalności, które zwiększą jej wartość użytkową oraz skalowalność. Przykładowym ulepszeniem mogłoby być wprowadzenie encji odpowiedzialnej za zarządzanie załącznikami (plikami), co pozwoliłoby użytkownikom na dodawanie dokumentów do poszczególnych projektów lub zadań. Dzięki temu, aplikacja mogłaby być bardziej uniwersalna i wspierać pełniejsze zarządzanie dokumentacją projektową.

**Bibliografia**

[1] Software Project Management Theory: TheoryW Dostęp: <https://www.hotpmo.com/management-models/software-project-management-theory-theory-w/> (odwiedzona 10.07.2024).

[2] What are CI/CD and the CI/CD pipeline? Dostęp: <https://www.ibm.com/think/topics/ci-cd-pipeline> (odwiedzona 10.07.2024)

[3] Jenkins Dostęp: <https://www.techtarget.com/searchsoftwarequality/definition/Jenkins> (odwiedzona 11.07.2024)

[4] What is CI/CD Dostęp : <https://about.gitlab.com/gitlab-duo/> (odwiedzona 11.07.2024)

[5] What is a CI/CD pipeline? Dostęp: <https://circleci.com/blog/what-is-a-ci-cd-pipeline/> (odwiedzona 11.07.2024)

[6] CI CD: 5 Trends shaping the Future od DevOps, Dostęp: <https://ofituria.com/ci-cd-5-trends-shaping-the-future-of-devops/> (odwiedzona 11.07.2024)

[7] What is Git?, Dostęp: <https://www.atlassian.com/git/tutorials/what-is-git> (odwiedzona 13.07.2024)

[8] Introduction to Maven – A Simple Project Management Tool, Dostęp: <https://geekflare.com/apache-maven-for-beginners/> (odwiedzona 13.07.2024)

[9] Introduction to Spring Boot: Understanding the Framework and Its Architecture, Dostęp: <https://medium.com/@riskiilyas03/exploring-spring-boot-understanding-the-framework-and-its-architecture-7e68205cb555> (odwiedzona 13.07.2024)

[10] Exploring Relational Database Management Systems (RDBMS), Dostęp: <https://codeinstitute.net/global/blog/what-is-a-relational-database-management-system/> (odwiedzona 13.07.2024)

[11] Introduction to Postman for API Development, Dostęp: <https://www.geeksforgeeks.org/introduction-postman-api-development/> , (odwiedzona 13.07.2024)

[12] What are Functional and Non-Functional Requirements and How to Document These, Dostęp: <https://enkonix.com/blog/functional-requirements-vs-non-functional/> (odwiedzona 15.07.2024)

[13] Opracowanie własne

[14] Defining JPA Entities. Dostęp: <https://www.baeldung.com/jpa-entities> (odwiedzona 16.07.2024)

[15] Working with Spring Data Repositories. Dostęp: <https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html> (odwiedzona 16.07.2024)

[16] Building REST services with Spring. Dostęp: <https://spring.io/guides/tutorials/rest> (odwiedzona 16.07.2024)

[17] The DTO Pattern (Data Transfer Object) . Dostęp: <https://www.baeldung.com/java-dto-pattern> (odwiedzona 16.07.2024)

[18] The Spring @Controller and @RestController Annotations. Dostęp: <https://www.baeldung.com/spring-controller-vs-restcontroller> (odwiedzona 16.07.2024)

[19] Spring Boot with H2 Database. Dostęp: <https://www.geeksforgeeks.org/spring-boot-with-h2-database/> . (odwiedzona 16.07.2024)

[20] Externalized Configuration. Dostęp: <https://docs.spring.io/spring-boot/docs/1.1.0.M1/reference/html/boot-features-external-config.html> (odwiedzona 18.07.2024)

[21] An introduction to environment variables and how to use them. Dostęp: <https://medium.com/chingu/an-introduction-to-environment-variables-and-how-to-use-them-f602f66d15fa> (odwiedzona 18.07.2024)

[22] PostreSQL About. Dostęp: <https://www.postgresql.org/about/> (odwiedzona 18.07.2024)

[23] Introduction to testing. Dostęp: <https://java-programming.mooc.fi/part-6/3-introduction-to-testing> (odwiedzona 20.07.2024)

[24] Integration Testing. Dostęp: <https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/integration-testing.html> (odwiedzona 25.07.2024)

[25] Unit Testing with Spring Boot. Dostęp: <https://reflectoring.io/unit-testing-spring-boot/> (odwiedzona 25.07.2024)

[26] Git. Dostęp: <https://www.techtarget.com/searchitoperations/definition/Git> (odwiedzona 13.07.2024)

[27] Introduction to Liquibase. Dostęp: <https://docs.liquibase.com/concepts/introduction-to-liquibase.html> (odwiedzona 16.07.2024)

[28] Know Everything about BDD Testing. Dostęp: <https://www.browserstack.com/guide/what-is-bdd-testing> (odwiedzona 26.07.2024)

[29] Code Covergae. Dostęp: <https://www.baeldung.com/cs/code-coverage> (odwiedzona 26.07.2024)

[30] Testing in Spring Boot. Dostęp: <https://www.baeldung.com/spring-boot-testing> (odwiedzona 28.07.2024)