

***Wrocławska Wyższa Szkoła***

***Informatyki Stosowanej***

Wydział Informatyki

Krzysztof Chwedziak

Nr albumu: 6379

Aplikacja bazodanowa wspomagająca zarządzanie magazynem

Praca: inżynierska

Kierunek: Informatyka

Specjalność: Programowanie

Praca wykonana pod kierunkiem:

Dr hab. Inż. Krzysztofa Pieczarki

*Wrocław 2023*

Spis treści

[1. Wstęp 3](#_Toc130809207)

[3. Zagadnienia Teoretyczne 4](#_Toc130809208)

[3.1 Wykorzystane technologie 4](#_Toc130809209)

[4. Część projektowa 7](#_Toc130809210)

[4.1 Wymagania funkcjonalne 7](#_Toc130809211)

[4.2 Wymagania niefunkcjonalne 9](#_Toc130809212)

[4.3 Ograniczenia dziedzinowe 11](#_Toc130809213)

[5. Bezpieczeństwo aplikacji 11](#_Toc130809214)

[5.1 Rodzaje ataków na aplikacje 11](#_Toc130809215)

[5.2 Wykorzystane sposoby zabezpieczające 11](#_Toc130809216)

[6. Graficzny interfejs użytkownika 11](#_Toc130809217)

[Bibliografia 19](#_Toc130809218)

# 1. Wstęp

Aplikacje bazodanowe są jednym z wielu narzędzi informatycznych, które umożliwiają zarządzanie bazami danych w sposób przyjazny dla użytkownika, najczęściej są tworzone z odpowiednimi interfejsami graficznymi ułatwiającymi pracę użytkownika. W dzisiejszych czasach, kiedy dane stały się jednym z najważniejszych zasobów przedsiębiorstw, to aplikacje te stały się wręcz niezbędne dla większości z branż, między innymi, w firmach produkcyjnych, posiadających techników, a co za tym idzie, posiadających magazyny utrzymania ruchu.

Magazyny te są miejscami, w których przechowywane są między innymi, części zamienne, materiały eksploatacyjne, ale i narzędzia służące do wykonywania napraw i konserwacji maszyn przemysłowych na terenie zakładu produkcyjnego. Zarządzanie tymi zasobami wymaga precyzyjnego planowania, kontroli stanu magazynu oraz podejmowania szybkich i skutecznych decyzji w przypadku wystąpienia awarii.

W pracy inżynierskiej została udokumentowany proces stworzenia aplikacji bazodanowej, której zadaniem jest pomoc w zarządzaniu magazynem utrzymania ruchu. Przedstawiona aplikacja umożliwia wykonywanie niezbędnych operacji związanych z utrzymaniem magazynu, między innymi kontrolę i zapewnienie stanu części poprzez kompleksową obsługę zamówień między magazynem a dostawcami utworzonymi w bazie.

# 3. Zagadnienia Teoretyczne

3.1 Wykorzystane technologie

Projekt Aplikacji bazodanowej wspomagającej zarządzanie magazynem wykorzystuje najnowsze technologie używane w procesie tworzenia współczesnych aplikacji internetowych, mając na uwadze cel, jak i wymagania postawione do tego projektu, można pogrupować technologie te na trzy główne kategorie: Bazy danych, Front-end, Back-end.

Bazy danych: MySQL to relacyjna baza danych, która jest jednym z najpopularniejszych systemów zarządzania bazami danych na świecie. MySQL jest darmową i otwarto-źródłową bazą danych, a jej popularność wynika z prostoty w obsłudze, a także wysokiej wydajności, oraz dużych możliwości skalowania.

MySQL posiada wiele zalet, które czynią ją dobrym wyborem dla aplikacji biznesowych, dlatego też postawiłem właśnie na tę technologię w swoim projekcie, są to m.in.:

• Wysoka wydajność: MySQL jest zoptymalizowany pod kątem wydajności i szybkości przetwarzania dużych ilości danych. Wykorzystuje zoptymalizowane algorytmy i struktury danych, co pozwala na szybkie wykonywanie zapytań.

• Duża niezawodność: MySQL jest bardzo niezawodnym systemem, który oferuje możliwość replikacji, co zwiększa dostępność bazy danych i minimalizuje ryzyko utraty danych.

• Bezpieczeństwo: MySQL posiada wiele funkcjonalności związanych z bezpieczeństwem, takich jak mechanizmy uwierzytelniania i autoryzacji, szyfrowanie oraz audytowanie działań użytkowników.

• Dobra skalowalność: MySQL oferuje wiele możliwości skalowania, co pozwala na dopasowanie systemu do potrzeb aplikacji i zwiększenie jej wydajności.

• Dostęp do narzędzi: MySQL posiada wiele narzędzi i bibliotek do pracy z bazami danych, co ułatwia pracę programistom i administratorom baz danych.

W swojej aplikacji wykorzystałem MYSQL jako główny system zarządzania bazami danych. Baza danych zawiera tabele z takimi danymi jak: użytkownicy, materiały, dostawcy, zamówienia, klucze bezpieczeństwa i inne, które są wykorzystywane w aplikacji do przetwarzania i wyświetlania informacji dla użytkowników. Schemat bazy danych został stworzony w programie MYSQL Workbench w wersji 8.0.32, co pozwoliło na optymalne projektowanie i wizualizację struktury bazy danych. Dzięki MySQL, moja aplikacja jest szybka, niezawodna i bezpieczna.

Back-end to część aplikacji internetowej, która odpowiedzialna jest za przetwarzanie danych i logikę biznesową. To serwerowa strona aplikacji, która jest niewidoczna dla użytkownika końcowego i działa w tle. Back-end zajmuje się obsługą żądań HTTP, interakcją z bazami danych, autoryzacją i uwierzytelnieniem użytkowników oraz zabezpieczeniem aplikacji przed atakami.

Back-end często jest tworzony w językach programowania takich jak Java, PHP, Python, Ruby, czy JavaScript. W dzisiejszych czasach coraz częściej wykorzystuje się frameworki, które ułatwiają pracę programistom, do najpopularniejszych frameworków należą: Spring, Django, Ruby on Rails, czy Express.js.

Back-end jest bardzo ważnym elementem aplikacji internetowych, ponieważ odpowiada za poprawne funkcjonowanie aplikacji i zabezpieczenia przed atakami.

W swojej aplikacji wykorzystałem framework Spring, jest on stworzony do aplikacji pisanych w języku Java. Został zaprojektowany tak, aby uprościć rozwój oprogramowania poprzez zapewnienie gotowych komponentów do konfiguracji aplikacji. Spring umożliwia wstrzykiwanie zależności, czyli umieszczanie zależności między klasami w plikach konfiguracyjnych, co pozwala na zmniejszenie skomplikowania kodu aplikacji. Ponadto, Spring oferuje wiele narzędzi takich jak Spring Data, Spring Security czy Spring Boot, które umożliwiają znaczne prostsze i optymalne tworzenie aplikacji.

Dodatkowo skorzystałem z narzędzia Maven, które służy do zarządzania projektami w języku Java i pozwala na zarządzanie bibliotekami, budowanie aplikacji testowanie i wdrażanie. Maven wykorzystuje plik pom.xml, który zawiera informacje o projekcie, w tym zależności, plug-iny i konfiguracje.

W projekcie aplikacji mojej pracy zostały wykorzystane następujące biblioteki:

• „spring-boot-starter-data-jpa” – umożliwia korzystanie z JPA do komunikacji z bazą danychw ramach frameworka Spring Boot

• „spring-boot-starter-data-rest” – pozwala na stworzenie interfejsu RESTful API na bazie repozytoriów JPA.

• „spring-boot-starter-mail” – zapewnia narzędzie do wysyłania wiadomości e-mail z aplikacji.

• „spring-boot-starter-security” – umożliwia integrację z frameworkiem Spring Security, zapewniającym zabezpieczenia w aplikacji.

• „spring-boot-starter-thymeleaf” – bibliotek do generowania widoków HTML z wykorzystaniem silnika szablonów Thymyleaf.

• „thymeleaf-extras-springsecurity6” - biblioteka dodatkowych tagów Thymeleaf umożliwiających integrację z Spring Security.

• “mysql-connector-j” - sterownik JDBC (Java Database Connectivity) dla bazy danych MySQL.

• „lombok” - biblioteka umożliwiająca generowanie kodu źródłowego, takiego jak gettery, settery i konstruktory, na podstawie adnotacji.

• „spring-boot-starter-test” - umożliwia pisanie testów jednostkowych w ramach frameworka Spring Boot.

• „spring-security-test” - biblioteka do testowania funkcjonalności związanych z zabezpieczeniami w aplikacji.

• „validation-api” - pozwala na walidację danych wejściowych w aplikacji.

• „mapstruct” - biblioteka umożliwiająca łatwe mapowanie pomiędzy obiektami różnych klas.

• „jjwt-jackson, jjwt-impl, jjwt-api” - biblioteki do tworzenia i weryfikowania tokenów JWT (JSON Web Tokens) w aplikacji.

• „spring-boot-starter-validation” - umożliwia walidację danych wejściowych w aplikacji.

Front-end to część aplikacji, która jest bezpośrednio widoczna i interaktywna dla użytkownika. Odpowiada ona za prezentację i interakcję z danymi pobranymi z Back-endu. W przypadku aplikacji internetowych Front-end to przede wszystkim kod HTML, CSS i JavaScript

Angular to jedna z popularnych technologii Front-endowych wykorzystywanych do tworzenia aplikacji internetowych. Jest to framework, który umożliwia łatwe i efektywne tworzenie aplikacji jednostronicowych, tzw. SPA. W Angularze wykorzystuje się TypeScript - nadzbiór języka JavaScript, co pozwala na bardziej czytelny i bezpieczniejszy kod.

Bootstrap to popularny framework CSS, który umożliwia łatwe i szybkie tworzenie responsywnych stron internetowych. Bootstrap dostarcza gotowe komponenty takie jak przyciski, menu, formularze, itp., co pozwala na oszczędność czasu w procesie tworzenia interfejsu użytkownika. Wykorzystanie Angular z dodatkiem Bootstrap w aplikacji pozwoliło na stworzenie responsywnego i intuicyjnego interfejsu użytkownika, który umożliwia łatwe i intuicyjne korzystanie z funkcjonalności oferowanych przez aplikację.

# 4. Część projektowa

4.1 Wymagania funkcjonalne

Aplikacja internetowa z wykorzystanymi technologiami daje możliwość wykonywania niżej wymienionych funkcjonalności:

Wymagania związane z użytkownikami:

• Utworzenie / zmiana / zablokowanie użytkownika

• Wymagania związane z materiałami:

• Utworzenie/modyfikacja/zablokowanie materiału,

• Utworzenie unikalnego identyfikatora materiału,

• Utworzenie / modyfikacja / usunięcie numeru wewnętrznego materiału,

• Utworzenie / modyfikacja nazwy materiału,

• Utworzenie / zmiana stanu magazynowego materiału,

• Utworzenie / zmiana ceny materiału,

• Utworzenie / zmiana jednostki wymiarowej materiału,

• Utworzenie / zmiana / usunięcie kodu EAN materiału,

• Utworzenie / modyfikacja / usunięcie wymaganego stanu minimalnego materiału,

• Utworzenie / modyfikacja / usunięcie opisu materiału,

• Utworzenie / modyfikacja / usunięcie lokalizacji składowania materiału,

• Wymagania związane z zamówieniami:

• Utworzenie / anulowanie zamówienia,

• Utworzenie unikalnego identyfikatora zamówienia,

• Utworzenie / zmiana statusu zamówienia,

• Dodanie pozycji z identyfikatorem do zamówienia,

• Dodanie rekordu zawierającego pojedynczy materiał wraz ze wskazaną przez użytkownika ilością jednostkową.

• Wysłanie zamówienia na przypisany w bazie danych adres E-mail do dostawcy,

• Możliwość zmiany statusu zamówienia

• Wymagania związane z dostawcami:

• Możliwość utworzenia / modyfikacji dostawcy,

• Dodanie / zmiana adresu dostawcy (państwo, miasto, ulica, numer domu, numer mieszkania, kod pocztowy)

• Dodanie / zmiana NIP dostawcy,

• Dodanie / zmiana REGON dostawcy,

• Dodanie / zmiana / usunięcie numeru kontaktowego dostawcy,

• Dodanie / zmiana adresu E-mail dostawcy,

• Dodanie / zmiana numeru konta dostawcy,

• Dodanie / zmiana numeru nazwy banku dostawcy,

• Dodanie / zmiana / usunięcie komentarza dostawcy,

Wymagania związane z obsługą aplikacji:

• Mechanizm wyszukiwania materiału po następujących atrybutach materiału: numer wewnętrzny (tabela: material, atrybut: materialId), nazwa materiału (tabela: material, atrybut: materialName), kategoria (tabela: category, atrybut: name), dostawca (tabela: vendor, atrybut: vendorName),

• Mechanizm stronicowania tabeli z materiałami znajdującymi się w bazie danych

• Mechanizm sortowania wyświetlanych materiałów po parametrach: nazwa materiału, identyfikator materiału, numer wewnętrzny materiału, dostawca,

• Interfejs dodawania nowego i modyfikacji materiału,

• Interfejs dodawania nowego i modyfikacji dostawcy,

• Interfejs dodawania nowego i modyfikacji pracownika,

• Interfejs wysłania zamówień do dostawcy,

• Interfejs przyjmowania zamówień,

• Interfejs rezerwacji materiału,

• Interfejs wyświetlania zamówień pracowników do wydań z możliwością ich potwierdzania, anulowania lub zmiany,

• Interfejs wyświetlania indywidualnych zamówień pracownika,

• Wymagania związane z wyposażeniem pracownika:

• Utworzenie / modyfikacja / usunięcie wyposażenia pracownika,

• Utworzenie unikatowego identyfikatora dla wyposażenia pracownika,

• Utworzenie zmiana / numeru inwentarzowego wyposażenia pracownika,

• Utworzenie / zmiana nazwy wyposażenia pracownika,

• Utworzenie producenta wyposażenia pracownika,

• Utworzenie modelu wyposażenia pracownika,

• Utworzenie numeru seryjnego producenta wyposażenia pracownika,

• Wprowadzenie daty zakupu wyposażenia pracownika,

• Wprowadzenie daty przekazania pracownikowi wyposażenia,

• Wprowadzenie daty ważności gwarancji na wyposażenie pracownika,

• Wprowadzenie daty złomowania wyposażenia pracownika,

• Wprowadzenie ceny zakupu wyposażenia pracownika,

• Wprowadzenie waluty zakupu wyposażenia pracownika,

• Zapisanie pliku z dowodem zakupu wyposażenia pracownika

4.2 Wymagania niefunkcjonalne

Wydajność – Aplikacja działa szybko i sprawnie nawet przy dużej liczbie użytkowników, w celu szybkiego i płynnego działania aplikacji zastosowałem różne techniki optymalizacyjne, takie jak cache’owanie[[1]](#footnote-1), redukcja liczby zapytań do bazy danych, optymalizacja kodu i minimalizacja czasu odpowiedzi. Dodatkowo zapewniłem odpowiednie środowisko produkcyjne, takie jak serwer z dużą ilością zasobów, aby zapewnić maksymalną wydajność aplikacji. Dzięki tym rozwiązaniom korzystanie z aplikacji odbywa się w sposób płynny i efektywny, nawet w przypadku dużego obciążenia.

Niezawodność – W celu zapewnienia, że aplikacja działa bezawaryjnie i bez utraty danych zastosowałem różne strategie takie jak backup danych, kontrola wersji kodu, a także testy automatyczne i manualne w procesie tworzenia aplikacji. Dzięki wybranej usługi bazy danych, tj. MySQL w łatwy sposób administratorzy systemu mogą ręcznie dokonać utworzenia kopii zapasowej danych zawartych w bazie, a także za harmonogramować automatyczne tworzenie kopii zapasowej w panelu obsługi danych np. MySQL Workbench. Kontrola wersji jest procesem śledzenia i zarządzania zmianami w kodzie źródłowym aplikacji lub innego typu dokumentów. Dzięki temu procesowi, programiści mogą monitorować i kontrolować historię zmian, porównywać wersje kodu źródłowego oraz wycofywać zmiany, gdy jest to konieczne.

GitHub to platforma internetowa umożliwiająca hosting i zarządzanie projektami z wykorzystaniem systemu kontroli wersji Git. Umożliwia to programistom pracę w grupie, wymianę kodu, współpracę i śledzenie historii zmian. GitHub oferuje wiele narzędzi i funkcji, takie jak możliwość zgłaszania problemów, zarządzenie pull-requestami[[2]](#footnote-2), kontrolę dostępu do repozytorium[[3]](#footnote-3) i wiele innych. Dzięki GitHubowi, programiści mogą łatwo i skutecznie zarządzać kodem źródłowym swoich projektów, co przyczynia się do sprawnego rozwoju aplikacji.

Bezpieczeństwo aplikacji jest niezwykle ważnym aspektem dla utrzymania zaufania użytkowników, oczekują oni bowiem, że ich dane osobowe, jak i dane poufne firmy, będą bezpieczne podczas korzystania z aplikacji. Wszelkie naruszenia bezpieczeństwa mogą wpłynąć na ich postrzeganie aplikacji i firmy w ogóle. Dlatego zabezpieczyłem aplikację przed atakami, co jest kluczowe dla utrzymania użytkowników i zapewnienia ich zadowolenia. Więcej na temat bezpieczeństwa ogólnie, i użytych metod wypowiem się w osobnym rozdziale poświęconemu bezpieczeństwu, gdyż uważam, że jest to niezwykle ważny aspekt w tworzeniu aplikacji internetowych.

Łatwość użytkowania jest również niezbędnym wymaganiem, ponieważ ma bezpośredni wpływ na doświadczenie użytkownika i zadowolenie z aplikacji. Aplikacja, która jest łatwa w obsłudze i intuicyjna pozwala użytkownikom na szybkie i sprawne realizowanie swoich celów. Dzięki temu użytkownicy są bardziej zadowoleni i chętniej korzystają z aplikacji. Z drugiej strony, aplikacja, która jest trudna w obsłudze lub nieintuicyjna, może wprowadzać frustrację i powodować, że użytkownicy przestaną korzystać z aplikacji, bądź będą to robili niechętnie będąc do tego zmuszonym przez pracodawcę.

Aplikacja, którą stworzyłem zawiera cechy aplikacji łatwej w obsłudze, posiada ona kilka kluczowych cech, które wyróżniają ją na tle innych:

* Prosta i intuicyjna nawigacja – użytkownik jest w stanie łatwo się odnaleźć i ma możliwość szybkiego dostępu do poszukiwanych funkcjonalności
* Jasne i przejrzyste komunikaty – aplikacja dostarcza użytkownikowi jasne i zrozumiałe komunikaty, które pozwalają mu na zrozumienie co się dzieje na ekranie.
* Minimalistyczny interfejs – zastosowałem minimalistyczny i przejrzysty interfejs, który pozwala użytkownikom skoncentrować się na najważniejszych elementach aplikacji.
* Spójna infrastruktura – aplikacja ma wspomnianą spójną infrastrukturę i styl aby użytkownicy mogli łatwo poruszać się w niej i nie musieli za każdym razem uczyć się nowych zasad.

Łatwość użytkowania to kluczowa cecha każdej aplikacji, która ma bezpośredni wpływ na doświadczenie użytkownika i zadowolenie z aplikacji. Mając na uwadze, że moja aplikacja jest stworzona głównie dla pracowników magazynu oraz działu technicznego, w którym miałem okazję pracować osobiście. Wiem jak kluczowym jest to aspektem szczególnie dla osób nie obcujących z aplikacjami na porządku dziennym w życiu prywatnym, bądź robiącym to w niewielkim stopniu, na przykład przeglądając platformy społecznościowe, bądź serwisy internetowe z wiadomościami. Bardzo często ci ludzie w momencie wdrożenia aplikacji będą do niej wrogo nastawieni, poniższej przedstawiam krzywą zmiany wg Johna Fishera. Widać na nim jak przebiega proces akceptacji między innymi nowego procesu, i jeżeli na samym początku utrudnimy obsługę aplikacji trudnym interfejsem, czyli czymś na co przeciętny użytkownik zwraca uwagę najbardziej, to narazimy się na wcześniej wymienione ryzyka.

(Fisher, 2017)

Rysunek Krzywa zmiany, John Fisher

4.3 Ograniczenia dziedzinowe

W procesie projektowania i tworzenia diagramu bazy danych do aplikacji przyjęto następujące ograniczenia dziedzinowe.

Nazwy tabel, nazwy atrybutów oraz inne elementy bazy zostały utworzone w języku angielskim mając na uwadze przyszłość aplikacji, oraz przejrzystość kodu, a także osobiste preferencje. Na co dzień w pracy, oraz prywatnie tworząc aplikacje internetowe posługuję się tym językiem i mam wypracowane nawyki i schematy, które są również znane innym programistom, przez co możliwy będzie dalszy rozwój projektu nawet przez osoby trzecie.

# 5. Bezpieczeństwo aplikacji

Bezpieczeństwo aplikacji jak już wcześniej było wspomniane, jest jednym z najważniejszych czynników, które wpływają na jej jakość. W przypadku aplikacji bazodanowej, którą stworzyłem, zapewnienie bezpieczeństwa jest szczególnie ważne, ponieważ zawarte mają być w niej poufne informacje, takie jak dane pracowników, dane dostawców, ustalone ceny materiałów i informacje dotyczące magazynu.

W tym rozdziale omówię, dlaczego bezpieczeństwo aplikacji jest ważne, jakie są rodzaje ataków na aplikacje oraz w jaki sposób moja aplikacja została zabezpieczona.

5.1 Rodzaje ataków na aplikacje

Atak typu brute-force, polega na wielokrotnym próbowaniu różnych kombinacji ciągu znaków aż do odnalezienia poprawnego hasła. Atakujący może wykorzystać automatyczne narzędzia do przeprowadzania ataku brute-force. Analogicznie można przeprowadzać odwrócony atak typu brute-force, czyli ustawienie jednego hasła, najczęściej bardzo prostego i wykonywanie skryptu odgadującego nazwę użytkownika.

Zabezpieczyć się przed tego typu atakami można ograniczając liczbę prób logowania, np. blokując konto po kilku nieudanych próbach logowania lub stosując CAPTCHA[[4]](#footnote-4)

Atak XSS (ang. Cross Site Scripting), polega na wstrzyknięciu kodu JavaScript do strony internetowej w celu wykradnięcia danych użytkownika lub wykonania innych działań w imieniu użytkownika. Atak XSS jest możliwy, gdy aplikacja nie filtruje i waliduje danych wejściowych, które są przesyłane z przeglądarki użytkownika. Aby zabezpieczyć się przed tego typu atakami należy filtrować i walidować dane wejściowe, używając bibliotek lub frameworków, które oferują takie funkcjonalności.

Atak SQL Injection (ang. Injection - z angielskiego wstrzykiwanie), jest to jedna z najczęstszych i najgroźniejszych technik ataków na strony i aplikacje internetowe.

Polega na wykorzystaniu błędów w sposób, w jaki aplikacja tworzy zapytania SQL, umożliwiając potencjalnemu atakującemu wykonanie niepożądanych działań, takich jak wyciąganie poufnych informacji, modyfikacja lub usuwanie danych.

Przykłady ataków SQL Inection:

* Wstrzykiwanie kodu SQL w formularzach logowania – atakujący wprowadza kod SQL zamiast danych logowania, co niesie za sobą ryzyko uzyskania nieautoryzowanego dostępu do bazy danych i wykradnięcie poufnych informacji.  
  Formularze logowania w większości zawierają dwa pola do wprowadzania danych. Tak jak w przypadku mojej aplikacji jest to pole zawierające adres email, będący nazwą użytkownika oraz hasło.

5.2 Wykorzystane sposoby zabezpieczające

# 6. Graficzny interfejs użytkownika

Tabela 4.1 – Tabela material[[5]](#footnote-5)

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: material |  |
| Atrybut | Opis |
| material\_id | Typ danych: BIGINT – klucz główny tabeli |
| material\_number | Typ danych: VARCHAR (255) |
| material\_name | Typ danych: VARCHAR (255) |
| material\_price | Typ danych: DOUBLE |
| material\_quantity | Typ danych: INT |
| unit\_of\_measure | Typ danych: VARCHAR (255) |
| material\_ean | Typ danych: VARCHAR (255) |
| material\_created | Typ danych: DATETIME (6) |
| material\_safety\_stock | Typ danych: INT |
| material\_description | Typ danych: VARCHAR (255) |
| category\_id | Typ danych: BIGINT – klucz obcy tabeli category |
| vendor\_id | Typ danych: BIGINT – klucz obcy tabeli vendor |
| material\_status | Typ danych: BIT (1) |
| material\_manufacturer | Typ danych: VARCHAR (255) |
| material\_location | Typ danych: VARCHAR (255) |

Tabela 4.2 – Tabela category

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: cateogry |  |
| Atrybut | Opis |
| category\_id | Typ danych: BIGINT – klucz główny tabeli |
| name | Typ danych: VARCHAR (255) |

Tabela 4.3 – Tabela vendor

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: vendor |  |
| Atrybut | Opis |
| vendor\_id | Typ danych: BIGINT – klucz główny tabeli |
| vendor\_name | Typ danych: VARCHAR (255) |
| vendor\_address | Typ danych: VARCHAR (255) |
| vendor\_city | Typ danych: VARCHAR (255) |
| vendor\_postal\_code | Typ danych: VARCHAR (255) |
| vendor\_country | Typ danych: VARCHAR (255) |
| vendor\_email | Typ danych: VARCHAR (255) |
| vendor\_phone | Typ danych: VARCHAR (255) |
| vendor\_nip | Typ danych: VARCHAR (255) |
| vendor\_regon | Typ danych: VARCHAR (255) |
| vednor\_krs | Typ danych: VARCHAR (255) |
| vendor\_bank\_account | Typ danych: VARCHAR (255) |
| vendor\_bank\_name | Typ danych: VARCHAR (255) |
| vendor\_created | Typ danych: DATETIME (6) |

Tabela 4.4 – Tabela user

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: user |  |
| Atrybut | Opis |
| user\_id | Typ danych: BIGINT – klucz główny tabeli |
| password | Typ danych: VARCHAR (255) |
| email | Typ danych: VARCHAR (255) |
| first\_name | Typ danych: VARCHAR (255) |
| last\_name | Typ danych: VARCHAR (255) |
| created | Typ danych: DATETIME (6) |
| enabled | Typ danych: BIT (1) |

Tabela 4.5 – Tabela OrderItem

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: order\_item |  |
| Atrybut | Opis |
| order\_item\_id | Typ danych: BIGINT – klucz główny tabeli |
| orders\_id | Typ danych: BIGINT – klucz obcy tabeli orders |
| material\_id | Typ danych: BIGINT – klucz obcy tabeli material |
| quantity | Typ danych: INT |

Tabela 4.6 Status

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: status |  |
| Atrybut | Opis |
| status\_id | Typ danych: BIGINT – klucz główny tabeli |
| name | Typ danych: VARCHAR (255) |

Tabela 4.7 orders

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: orders |  |
| Atrybut | Opis |
| orders\_id | Typ danych: BIGINT – klucz główny tabeli |
| order\_number | Typ danych: VARCHAR (255) |
| order\_date | Typ danych: DATETIME (6) |
| delivery\_date | Typ danych: DATETIME (6) |
| comment | Typ danych: VARCHAR (255) |
| order\_total | Typ danych: DOUBLE |
| user\_id | Typ danych: BIGINT – klucz obcy tabeli user |
| status\_id | Typ danych: BIGINT – klucz obcy tabeli status |

Tabela 4.8 – Tabela InternalOrder

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: internal\_order |  |
| Atrybut | Opis |
| internal\_order\_id | Typ danych: BIGINT – klucz główny tabeli |
| pick\_up\_location | Typ danych: VARCHAR (255) |
| internal\_order\_date | Typ danych: DATETIME (6) |

Tabela 4.9 – Tabela token

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: verification\_token |  |
| Atrybut | Opis |
| id | Typ danych: BIGINT – klucz główny tabeli |
| token | Typ danych: VARCHAR (255) |
| user\_user\_id | Typ danych: BIGINT – klucz obcy tabeli user |
| expiry\_date | Typ danych: DATETIME (6) |

Tabela 4.10 – Tabela RefreshToken

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: refresh\_token |  |
| Atrybut | Opis |
| id | Typ danych: BIGINT – klucz główny tabeli |
| token | Typ danych: VARCHAR (255) |
| created\_date | Typ danych: DATETIME (6) |

Tabela 4.11 – Tabela internal\_order\_item

Źródło: [opracowanie własne]

|  |  |
| --- | --- |
| Tabela: internal\_order\_item |  |
| Atrybut | Opis |
| id | Typ danych: BIGINT – klucz główny tabeli |
| internal\_order\_id | Typ danych: BIGINT – klucz obcy tabeli internal\_order |
| material\_id | Typ danych: BIGINT – klucz obcy tabeli material |
| quantity | Typ danych: INT |

[Rysunek 1 Krzywa zmiany, John Fisher 11](#_Toc130808606)

# Bibliografia

GitHub Inc. (2023, Marzec 26). *GitHub*. Pobrano z lokalizacji GitHub Docs: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests

1. Cache’owanie – wprowadzenie pamięci podręcznej jako narzędzie optymalizacji i wydajności aplikacji [↑](#footnote-ref-1)
2. Pull-request - (GitHub Inc., 2023) [↑](#footnote-ref-2)
3. Repozytorium – centralne miejsce, w którym przechowywany jest kod źródłowy projektu i historia zmian [↑](#footnote-ref-3)
4. CAPTCHA (ang. Completely Automated Public Turing test to tell Computers and Humans Apart) to mechanizm zabezpieczający przed automatycznymi programami [↑](#footnote-ref-4)
5. Material (ang. Material – z angielskiego materiał, w aplikacji przyjęty jako część zamienna) [↑](#footnote-ref-5)