**20-01-2024**

Politechnika Krakowska im. Tadeusza Kościuszki

Wydział Inżynierii Elektrycznej i Komputerowej

Sylwia Jaworska

Grzegorz Listwan

Krzysztof Pacura

Inżynieria Programowania

Prowadzący przedmiot:

Prof. dr hab. inż. Sergii Telenyk

Projekt tworzenie oprogramowania dla systemu zarządzania zakładem przemysłowym.

**Raport 3**

Spis treści

[Wstęp 4](#_Toc155128143)

[Temat projektu 4](#_Toc155128144)

[Cel projektu 4](#_Toc155128145)

[Opis problemu 4](#_Toc155128146)

[Założenia projektu 6](#_Toc155128147)

[Skład zespołu: 7](#_Toc155128148)

[Technologie: 7](#_Toc155128149)

[Fundusze: 7](#_Toc155128150)

[Terminy: 7](#_Toc155128151)

[Testy jakości: 7](#_Toc155128152)

[Bezpieczeństwo Aplikacji 8](#_Toc155128153)

[Dokumentacja projektu: 8](#_Toc155128154)

[Wsparcie posprzedażowe 8](#_Toc155128155)

[Szkolenia dla użytkowników 8](#_Toc155128156)

[Zakres produktu 8](#_Toc155128157)

[Główne funkcje produktu 8](#_Toc155128158)

[Wymagania 10](#_Toc155128159)

[Wymagania funkcjonalne 10](#_Toc155128160)

[Wymagania niefunkcjonalne 15](#_Toc155128161)

[Cele biznesowe 16](#_Toc155128162)

[Struktura Organizacyjno-Funkcjonalna zakładu przemysłowego 17](#_Toc155128163)

[Produkty i usługi 18](#_Toc155128164)

[Metody pracy i potrzeba zmiany 19](#_Toc155128165)

[Podmioty zainteresowane 22](#_Toc155128166)

[Analiza ograniczeń 23](#_Toc155128167)

[Analiza ryzyka 25](#_Toc155128168)

[Analiza wzorców architektonicznych 27](#_Toc155128169)

[Architektura monolityczna 27](#_Toc155128170)

[Architektura mikroserwisowa 28](#_Toc155128171)

[Architektura oparta na wydarzeniach 29](#_Toc155128172)

[Wybór architektury 30](#_Toc155128173)

[Analiza infrastruktur projektu 30](#_Toc155128174)

[Wybór infrastruktury projektu 31](#_Toc155128175)

[Analiza architektury za pomocą trzech wzorców 32](#_Toc155128176)

[Wzorzec Komponentowo-Łącznikowy: 32](#_Toc155128177)

[Wzorzec Modułowy: 32](#_Toc155128178)

[Wzorzec Alokacyjny dla Architektury Mikroserwisów w SaaS: 32](#_Toc155128179)

[Specyfikacja implementacyjna 35](#_Toc155128180)

[Schemat bazy danych – diagram ERD 35](#_Toc155128181)

[Diagram Klas 36](#_Toc155128182)

[Diagram Przypadków Użycia 37](#_Toc155128183)

[Diagram Przepływu Danych – DFD 38](#_Toc155128184)

# Wstęp

Dokument ten jest dokumentacją projektu z przedmiotu Inżynieria Programowania.

W skład zespołu realizującego ten projekt wchodzą następujące osoby: Sylwia Jaworska, Grzegorz Listwan, Krzysztof Pacura.

Celem niniejszego projektu jest zgłębienie wiedzy na temat tworzenia oprogramowania opartego na trzech poziomach: biznesowym, produkcyjnym i procesowym.

# Temat projektu

„Tworzenie oprogramowania dla systemu zarządzania zakładem przemysłowym.”

# Cel projektu

Celem projektu jest stworzenie aplikacji wspierającej zarządzanie zakładem przemysłowym. Aplikacja ta powinna zawierać moduły takie jak hala produkcyjna, magazyn, utrzymanie ruchu, zakupy, logistyka czy HR tworząc razem kompleksowy system do zarządzania wszystkimi zasobami w zakładzie przemysłowym. Efektem końcowym projektu będzie nowoczesny, skalowalny i elastyczny system, zwiększający efektywność zarządzania zasobami w zakładzie przemysłowym, co pozwoli na zaoszczędzenie czasu i pieniędzy poprzez lepsze wykorzystanie zasobów ludzkich, sprzętowych oraz stanów magazynowych.

# Opis problemu

Zakład przemysłowy powinien być zorganizowany w taki sposób, aby wszystkie zasoby były wykorzystywane w sposób optymalny co oznacza żeby nie było ani nadmiernej eksploatacji ani przestojów. Organizacja takiej pracy stanowi duże wyzwanie dla kadry kierowniczej bez pomocy odpowiedniego oprogramowania. Takimi problemami mogą być np.:

1. ***Optymalizacja zasobów ludzkich -***  oznacza to iż może zdarzyć się sytuacja w której nadmierna liczba pracowników jest przypisana do danego obszaru a w innym obszarze jest ich za mało, co prowadzi do nadmiernego obciążenia lub niewykorzystania pełnego potencjału ludzkiego.
2. ***Zarządzanie sprzętem -***  brak efektywnego i prawidłowego wykorzystania urządzeń może prowadzić do przestojów spowodowanych nadmiernym zużyciem, awarią oraz błędną eksploatacją przez nieprzeszkolonego pracownika.
3. ***Zarządzanie magazynem***  - brak klarownego śledzenia zapasów, co może prowadzić do braków w niektórych materiałach lub nadmiernych nadwyżek innych, co wpływa na koszty i produkcję. Brak systemu umożliwiającego szybkie reagowanie na zmiany w zamówieniach, może ograniczyć zdolność do elastycznego dostosowania produkcji.
4. ***Niezadowolenie pracowników -***  jeśli zasoby ludzkie nie są efektywnie zarządzane, pracownicy mogą odczuwać frustrację z powodu nadmiernego obciążenia lub braku klarowności w zakresie obowiązków.
5. ***Wydajność operacyjna -***  problemy w tym obszarze mogą prowadzić do ogólnie niskiej wydajności operacyjnej, opóźnień w produkcji, zwiększenia kosztów i utraty konkurencyjności.
6. ***Reklama i marketing -***  nieefektywne działania promocyjne lub brak jasnego przekazu może prowadzić do pogorszenia relacji z klientem i gorszego postrzegania marki.
7. ***Księgowość –*** brak efektywnego systemu księgowego może mieć negatywny wpływ na bieżącą analizę danych finansowych co może wpłynąć na skuteczność podejmowanych decyzji biznesowych.
8. ***Ryzyko i statystyka -***  brak spójnego systemu analizy ryzyka może prowadzić do nieodpowiedniego przygotowania na nieprzewidziane zdarzenia. Brak danych lub narzędzi do skutecznej prognozy potrzeb rynkowych i produkcji może prowadzić do błędnych decyzji.
9. ***System inżynieryjny -***  brak standardów inżynieryjnych może prowadzić do różnic w jakości i spójności produkcji. Ponadto brak struktury wspierającej proces innowacyjny może ograniczać zdolność zakładu do wprowadzania nowych rozwiązań technologicznych.
10. ***Kontrola procesu –*** brak skutecznego systemu monitorowania i kontroli procesów produkcyjnych może prowadzić do niemożności śledzenia etapów produkcji i wykrywania problemów w czasie rzeczywistym.
11. ***Automatyzacja -*** niedostateczne zastosowanie systemów automatyzacji może ograniczyć efektywność produkcji i zwiększyć ryzyko błędów ludzkich.
12. ***BHP –*** niepoprawny rozkład maszyn i surowców w hali produkcyjnej może prowadzić do ograniczenia przepływu produkcyjnego i zwiększenia ryzyka wypadków, ponadto niewłaściwe rozmieszczenie stanowisk pracy lub brak ergonomii może prowadzić do spadku wydajności oraz zwiększonej liczby urazów pracowników.

Stworzenie takiego systemu może przynieść wiele korzyści, w tym lepsze wykorzystanie zasobów, optymalizację procesów produkcyjnych, redukcję strat i poprawę efektywności operacyjnej. Taki system mógłby integrować zarządzanie magazyn, utrzymanie ruchu, zakupy, logistyka czy HR tworząc razem kompleksowy system do zarządzania wszystkimi zasobami w zakładzie przemysłowym zapewniając bardziej płynne funkcjonowanie.

# Założenia projektu

System powinien być zintegrowany i obejmować moduły zarządzania zasobami ludzkimi, sprzętowymi i magazynem. Te moduły powinny współpracować, umożliwiając jednoczesne monitorowanie i zarządzanie różnymi aspektami działalności zakładu. Kolejnym ważnym aspektem jest aby produkt ten umożliwiał śledzenie danych dotyczących pracowników, takich jak umiejętności, obecność, dostępność, wydajność oraz harmonogramy pracy. To pozwoli na lepsze dopasowanie pracowników do zadań i optymalne wykorzystanie ich umiejętności, ponadto system powinien zawierać moduł do monitorowania wydajności i stanu sprzętu, powinien śledzić zużycie, okresy eksploatacji, planowane konserwacje i naprawy, aby zoptymalizować wykorzystanie i minimalizować przestoje. Niewątpliwie ważnym aspektem tego oprogramowania powinna być możliwość zarządzanie stanami magazynowymi, śledzenie ruchu towarów, kontrolę zapasów i automatyzację zamówień w celu uniknięcia nadmiernych braków lub nadmiarów. Projekt powinien być elastyczny i skalowalny, umożliwiając dostosowanie do różnych potrzeb i rozmiarów zakładu przemysłowego. Ponadto, interfejs użytkownika powinien być intuicyjny, aby użytkownicy mogli łatwo korzystać z systemu. Zapewnienie bezpieczeństwa danych pracowników, informacji o sprzęcie i magazynie jest kluczowe. System powinien spełniać wysokie standardy bezpieczeństwa, zapewniając poufność, integralność i dostępność danych, powinien oferować funkcje raportowania i analizy danych, umożliwiając zarządzanie na podstawie danych i generowanie raportów dotyczących wydajności, kosztów, efektywności oraz prognozowanie potrzeb zasobów. Wdrożenie systemu powinno być poparte odpowiednim szkoleniem dla personelu oraz zapewnieniem wsparcia technicznego na bieżąco, aby zapewnić skuteczne i ciągłe wykorzystanie systemu.

## Skład zespołu:

* Zespół powinien składać się z osób o różnych umiejętnościach, aby zapewnić kompleksową implementację projektu. W naszym przypadku zespół powinien składać się z trzech osób: jednego programisty Java, jednego specjalisty ds. baz danych i interfejsu użytkownika oraz jednego członka odpowiedzialnego za logikę biznesową

## Technologie:

* W projekcie wykorzystane zostaną następujące technologie:

- implementacja aplikacji w języku Java

- wykorzystanie bazy danych MySQL

- zarządzanie kodem źródłowym przy użyciu kontroli wersji GIT i platformy GitHub

- praca w zintegrowanym środowisku IDE takim jak InteliiJ IDEA

- narzędzia do zarządzania projektem takie jak Jira w celu efektywnego planowania, monitorowania postępów i zarządzania zadaniami w zespole

## Fundusze:

* Budżet projektu obejmuje koszty zatrudnienia, zakupu licencji na narzędzia programistyczne, a także inne niezbędne wydatki, nie przekraczając łącznej kwoty 1 000 000zł.

## Terminy:

* Etap projektowania aplikacji powinien być zakończony do końca listopada 2023
* Etap implementacji oprogramowania do końca grudnia 2023
* Dostarczenie całości projektu do końca stycznia 2024

## Testy jakości:

* Stworzenie planów testów, który uwzględnia różne scenariusze testowe

## Bezpieczeństwo Aplikacji

* Wdrożenie mechanizmów szyfrowania danych, wykonywanie kopi zapasowych, zabezpieczenie przed atakami XSS, CSRF itp.

## Dokumentacja projektu:

* Utworzenie dokumentacji technicznej, instrukcji użytkownika oraz dokumentacji dla deweloperów, co ułatwi dalszy rozwój i utrzymanie aplikacji

## Wsparcie posprzedażowe

* Utworzenie systemu zgłaszania błędów, dostępności pomocy technicznej oraz opracowanie harmonogramu aktualizacji.

## Szkolenia dla użytkowników

* Przeprowadzenie szkoleń oraz warsztatów dla użytkowników końcowych, aby zminimalizować czas potrzebny na dostosowanie się do nowego oprogramowania

# Zakres produktu

Zakres tworzonego produktu obejmuje stworzenie aplikacji desktopowej oraz dedykowanej dla niej bazy danych. Z systemu będzie mogło korzystać wielu użytkowników jednocześnie. Prosty i intuicyjny interfejs ma ułatwić korzystanie z aplikacji przeciętnym użytkownikom, ale też osobą które nie mają na co dzień styczności z takimi systemami.

# Główne funkcje produktu

1. **Zarządzanie Produkcją:**
   * *Planowanie Produkcji:* Elastyczne narzędzie do tworzenia planów produkcyjnych uwzględniających dostępność zasobów, terminy realizacji i preferencje klientów.
   * *Monitorowanie Procesów Produkcyjnych:* Śledzenie postępu produkcji, identyfikacja potencjalnych opóźnień, analiza wydajności i generowanie raportów.
2. **Zarządzanie Magazynem:**
   * *Inwentaryzacja:* Automatyczna inwentaryzacja magazynów przy użyciu technologii RFID lub kodów kreskowych, minimalizująca błędy ludzkie.
   * *Optymalizacja Zasobów:* Narzędzia wspierające optymalizację poziomu zapasów, minimalizujące koszty magazynowania.
3. **Zarządzanie Zamówieniami i Dostawami:**
   * *Obsługa Zamówień:* Pełna integracja z procesem zamówień, od przyjęcia zamówienia do wysyłki produktu, z możliwością śledzenia statusu.
   * *Zarządzanie Dostawami:* Monitorowanie dostaw, śledzenie terminów dostaw, zarządzanie relacjami z dostawcami.
4. **Planowanie Zasobów Firmy:**
   * *Integracja Zasobów:* Skonsolidowane zarządzanie zasobami firmy, w tym finansami, kadrami, oraz infrastrukturą IT.
   * *Raportowanie i Analizy:* Generowanie raportów i analiz dla różnych dziedzin funkcjonowania zakładu, wspomaganie procesów decyzyjnych.
5. **Zarządzanie Jakością:**
   * *Kontrola Jakości:* Monitorowanie jakości produktów na różnych etapach produkcji, z możliwością wprowadzania korekt w czasie rzeczywistym.
   * *Zarządzanie Reklamacjami:* System do obsługi reklamacji klientów, z możliwością analizy przyczyn i wprowadzania usprawnień.
6. **Obsługa BHP i Ergonomii:**
   * *Monitorowanie Warunków Pracy:* Zbieranie danych dotyczących warunków pracy, z możliwością identyfikacji obszarów wymagających poprawy.
   * *Zarządzanie Szkoleniami BHP:* Planowanie i śledzenie szkoleń z zakresu BHP dla pracowników.
7. **Bezpieczeństwo i Audyt:**
   * *Zabezpieczenia:* System zgodny z najnowszymi standardami bezpieczeństwa danych, obejmujący autoryzację dostępu, szyfrowanie i monitorowanie aktywności użytkowników.
   * *Audyt:* Dziennik zdarzeń (log) umożliwiający śledzenie działań użytkowników oraz potencjalnych incydentów.
8. **Wsparcie Techniczne i Szkolenia:**
   * *Helpdesk:* System wsparcia technicznego dla użytkowników, obejmujący zgłaszanie błędów, prośby o pomoc oraz dostęp do dokumentacji.
   * *Szkolenia Użytkowników:* Przygotowanie i dostarczenie szkoleń dla pracowników korzystających z systemu.
9. **Elastyczność i Rozszerzalność:**
   * *Konfigurowalność:* Możliwość dostosowywania systemu do specyficznych potrzeb zakładu przemysłowego.
   * *Integracje Zewnętrzne:* Otwarta architektura umożliwiająca integrację z innymi systemami używanymi w firmie.

# Wymagania

## Wymagania funkcjonalne

1. **Produkcja:**

* Użytkownik może zalogować się do systemu jako Pracownik, Lider, Kierownik lub Administrator
* Użytkownik ma dostęp do funkcji i danych zgodnie z przypisaną rolą
* Administrator może przypisywać role do nowych użytkowników i uaktualniać istniejące
* Pracownik może przeglądać aktualne zlecenia dostępne dla niego, zgodne z jego uprawnieniami
* Pracownik może wybrać zlecenie z dostępnych i zmienić jego status z „dostępne” na „w trakcie realizacji”
* Zmiana statusu z „dostępne” na „w trakcie realizacji” skutkuje wysłaniem zamówienia na przypisane do zlecenia komponenty do magazynu
* Zmiana statusu z „dostępne” na „w trakcie realizacji” rezerwuje dostęp do narzędzi, maszyn i innych wykorzystywanych zasobów
* System aktualizuje dostępne zlecenia uwzględniając zablokowane zasoby
* Pracownik może zmienić status zlecenia, które wykonuje na „wykonane” oraz wprowadzić odpowiednie dane do systemu
* Zmiana statusu zlecenia na „wykonane” zwalnia dostęp do danych narzędzi i innych wykorzystywanych zasobów
* Lider ma dostęp do wszystkich opcji Pracownika rozszerzone o dodatkowe opcje
* Lider może przeglądać wszystkie zlecenia dostępne, wykonywane, zakończone wraz z informacją kto jest przypisany do danego zlecenia, jakie zasoby są wykorzystywane i jaki jest czas wykonywania
* Lider może przeglądać listę pracowników wraz z informacją o ich aktualnych zadaniach
* Lider ma możliwość generowania raportów dotyczących wydajności pracy z uwzględnieniem danego okresu czasu, obszaru czy pracownika
* Lider może przeglądać listy wszystkich dostępnych maszyn i sprzętów wraz z informacją o ich aktualnym wykorzystaniu
* Lider ma możliwość generowania raportów dotyczących czasu wykorzystania danego sprzętu i maszyn
* Kierownik ma dostęp do wszystkich opcji Lidera rozszerzone o dodatkowe opcje
* Kierownik może tworzyć nowe kategorie zadań i przypisywać do nich wymagane uprawnienia, sprzęty, materiały i inne wykorzystywane zasoby
* Kierownik może tworzyć nowe zlecenia z wybranej kategorii zadań
* Kierownik ma możliwość nadania priorytetu zadaniu lub zmianę aktualnego
* Kierownik może przypisywać pracownikom nowe uprawnienia do korzystania z danych maszyn i sprzętu
* Kierownik może dodawać nowe sprzęty i maszyny wraz z informacją o kategorii i wymaganych uprawnieniach
* Kierownik może zmieniać status maszyn i sprzętów na „niedostępne”

1. **Magazyn:**

* System powinien umożliwiać dokładne śledzenie ilości i lokalizacji wszystkich produktów w magazynie.
* Automatyczne aktualizacje stanów magazynowych po każdej transakcji (przyjęciu, wydaniu, przemieszczeniu).
* Możliwość składania zamówień na nowe produkty zintegrowana z systemem zakupowym.
* Realizacja zamówień zgodnie z ustalonymi priorytetami i terminami.
* System powinien umożliwiać optymalne rozmieszczenie produktów w magazynie, uwzględniając ich rotację i dostępność.
* Automatyczne generowanie zamówień surowców na podstawie minimalnych poziomów magazynowych.
* Powiadomienia o konieczności uzupełnienia stanów magazynowych.
* System powinien umożliwiać identyfikację i śledzenie partii produktów, ułatwiając w razie potrzeby wycofanie z rynku.
* Integracja z systemami dostawców, klientów i innych partnerów biznesowych.
* Automatyczne przekazywanie informacji o zamówieniach, dostawach i stanach magazynowych.
* Dokładne monitorowanie ruchu towarów w magazynie, umożliwiające generowanie raportów i analiz.
* Umożliwienie jednoznacznej identyfikacji produktów poprzez zastosowanie odpowiednich kodów kreskowych lub RFID.
* Automatyczne skanowanie i rejestracja przy użyciu technologii identyfikacji.
* Generowanie raportów dotyczących stanów magazynowych, ruchu towarów, i efektywności operacyjnej

1. **Logistyka:**

* Monitorowanie i zarządzanie dostawcami, w tym ocena ich wydajności.
* Śledzenie terminów dostaw i identyfikacja potencjalnych opóźnień.
* Integracja z systemem magazynowym dla skoordynowanego zarządzania zapasami i przepływem towarów.
* Śledzenie w czasie rzeczywistym przesyłek i dostaw.
* Powiadomienia o ewentualnych opóźnieniach, zmianach trasy lub innych istotnych informacjach.
* Łatwa integracja z modułami produkcyjnymi i magazynowymi.
* Automatyczne przekazywanie informacji o zamówieniach i dostawach.
* System umożliwiający automatyczne rezerwacje przestrzeni transportowej i dostawców zgodnie z harmonogramem produkcji.
* Generowanie raportów dotyczących efektywności tras, czasów dostaw, oraz kosztów transportu.
* Narzędzia do analizy efektywności procesów logistycznych i identyfikacji obszarów poprawy.
* Przypisywanie zadań związanych z logistyką pracownikom.
* Harmonogramowanie działań związanych z załadunkiem, rozładunkiem, i transportem.
* Monitorowanie i kontrola zgodności z wymaganiami celnymi i podatkowymi.

1. **HR:**

* System powinien umożliwiać pełne zarządzanie danymi personalnymi pracowników, w tym informacjami kontaktowymi, historią zatrudnienia, oraz kwalifikacjami.
* Planowanie i monitorowanie dostępności pracowników w zależności od potrzeb produkcyjnych.
* Rejestrowanie kompetencji i umiejętności pracowników.
* Systematyczne monitorowanie czasu pracy pracowników.
* Rejestracja nadgodzin, urlopów, i innych form czasu wolnego.
* Rejestracja i zarządzanie procesem szkoleń, w tym śledzenie ukończonych szkoleń przez pracowników.
* Planowanie rozwoju zawodowego pracowników.
* Automatyczne obliczanie wynagrodzeń na podstawie czasu pracy i innych czynników.
* Zarządzanie systemem świadczeń i beneficjami dla pracowników.
* Dokładna ewidencja zatrudnienia pracowników, w tym daty rozpoczęcia i zakończenia pracy.
* Zarządzanie dokumentacją związaną z zatrudnieniem, taką jak umowy i aneksy.
* Platforma do komunikacji wewnętrznej, umożliwiająca przekazywanie informacji, ogłoszeń, i powiadomień pracownikom.
* Elektroniczny system zgłaszania problemów i pytań.
* Okresowe oceny wydajności pracowników.
* Planowanie celów indywidualnych i zbiorowych oraz monitorowanie ich realizacji.

1. **Utrzymanie ruchu:**

* System powinien umożliwiać planowanie regularnych przeglądów technicznych maszyn i urządzeń.
* Monitorowanie terminów przeglądów i generowanie powiadomień o zbliżających się terminach.
* Przypisywanie zadań konserwacyjnych pracownikom lub zespołom.
* Automatyczne generowanie harmonogramu utrzymania ruchu na podstawie historii awarii i planów produkcyjnych.
* Rejestracja incydentów, awarii i zgłoszeń napraw.
* Systematyczne analizowanie danych awarii w celu identyfikacji przyczyn i zapobiegania powtarzaniu się problemów.
* Monitorowanie stanów magazynowych części zamiennych.
* Automatyczne zamawianie części na podstawie minimalnych poziomów zapasów.
* Generowanie raportów dotyczących czasu przestoju maszyn z powodu konserwacji, napraw i przeglądów.
* Analiza danych dotyczących wydajności maszyn.
* Integracja z systemami monitoringu stanu technicznego maszyn.
* Automatyczne przekazywanie danych o parametrach pracy maszyn i urządzeń.
* Planowanie konserwacji.
* Automatyczne dostosowywanie harmonogramu utrzymania ruchu w przypadku awarii.
* Przechowywanie i zarządzanie dokumentacją techniczną maszyn i urządzeń.
* Łatwy dostęp do instrukcji obsługi, schematów i dokumentacji technicznej.

1. **Zakupy:**

* System powinien umożliwiać pełne zarządzanie danymi dostawców, w tym informacjami kontaktowymi, historią dostaw i ocenami ich wydajności.
* Automatyczne powiadomienia o zmianach w relacjach z dostawcami, takich jak zmiana cen czy terminów płatności.
* Ewidencja i aktualizacja katalogu produktów dostępnych u dostawców.
* Monitorowanie zmian cen i warunków zakupu.
* Automatyczne generowanie zamówień na podstawie minimalnych poziomów magazynowych i prognoz zapotrzebowania.
* Możliwość ręcznego modyfikowania automatycznie generowanych zamówień.
* Rejestracja dostaw surowców i materiałów.
* Monitorowanie statusu zamówień od momentu złożenia przez dział zakupów do momentu dostawy.
* Powiadomienia o ewentualnych opóźnieniach czy zmianach w terminach dostaw.
* Zarządzanie negocjacjami cenowymi, warunkami płatności i innymi aspektami umów handlowych.
* Przechowywanie umów handlowych i ich terminów ważności.
* Monitorowanie i kontrola budżetu zakupów.
* Raportowanie dotyczące wydatków, oszczędności i ewentualnych przekroczeń budżetowych.
* Integracja z modułem magazynowym w celu optymalizacji zarządzania zapasami i kontrolowania stanów magazynowych.
* Raportowanie dotyczące wydajności dostawców na podstawie parametrów takich jak terminowość, jakość dostaw, i elastyczność.
* Oceny i recenzje dostawców ze strony działu zakupów.

1. **Jakość:**

* Możliwość przeprowadzania kontroli jakości surowców, półproduktów i gotowych wyrobów.
* Systematyczne rejestrowanie danych dotyczących wyników kontroli jakości.
* Elektroniczne dokumentowanie i przechowywanie raportów z kontroli jakości.
* Integracja z systemami monitorowania procesów produkcyjnych w celu śledzenia parametrów mających wpływ na jakość.
* Generowanie ostrzeżeń w przypadku odchylenia od ustalonych norm.
* Rejestracja i analiza reklamacji od klientów związanych z jakością produktów.
* Planowanie i przeprowadzanie audytów jakościowych w zakładzie produkcyjnym.
* Ocena zgodności z wymaganiami norm jakościowych i procedurami firmy.
* Przechowywanie i aktualizacja norm i standardów jakościowych obowiązujących w branży oraz wewnętrznych procedur jakościowych.
* Automatyczne dostosowywanie procedur w przypadku zmian w normach.
* Generowanie raportów dotyczących wyników kontroli jakości, incydentów, audytów i działań korygujących.
* Narzędzia do analizy trendów i identyfikacji obszarów poprawy.
* Określanie i dokumentowanie procedur postępowania w przypadku odchylenia jakościowego.
* Automatyczne przypisywanie zadań i monitorowanie ich realizacji.
* Planowanie i monitorowanie szkoleń pracowników związanych z procedurami jakościowymi.
* Integracja z modułem magazynowym, produkcji i innymi modułami w celu uzyskania pełnej widoczności i kontroli nad procesami związanymi z jakością.

1. **BHP**

* Pełna ewidencja pracowników zawierająca informacje dotyczące stanu zdrowia, przeszkolenia BHP i innych istotnych danych.
* Automatyczne powiadomienia o zbliżających się terminach badań okresowych.
* Planowanie i śledzenie przeprowadzanych szkoleń z zakresu BHP.
* Elektroniczna rejestracja uczestnictwa w szkoleniach.
* Rejestracja i analiza danych dotyczących warunków pracy.
* Ewidencja i monitorowanie wydawanych pracownikom środków ochrony indywidualnej (np. kaski, odzież ochronna).
* Automatyczne przypominanie o konieczności wymiany lub przeglądu wyposażenia.
* Rejestracja każdego wypadku, incydentu czy choroby zawodowej.
* Planowanie i monitorowanie okresowych przeglądów systemów bezpieczeństwa (np. gaśnice, systemy gaśnicze).
* Automatyczne generowanie zleceń na przeglądy i naprawy.
* Ewidencja i dostęp do aktualnych przepisów, norm i zaleceń BHP obowiązujących w danym zakładzie.
* Powiadamianie pracowników o zmianach w przepisach i procedurach BHP.
* Generowanie raportów dotyczących statystyk wypadków, incydentów, przeprowadzonych szkoleń i innych działań związanych z BHP.

## Wymagania niefunkcjonalne

**1. Wydajność:** Czas odpowiedzi systemu nie dłuższy niż 1 sekunda dla wszystkich interakcji użytkownika.

**2. Dostępność:** Minimalna dostępność systemu na poziomie 99% w dni robocze roku kalendarzowego

**3. Bezpieczeństwo:** Wdrożenie standardów szyfrowania danych w transmisji i przechowywaniu. Regularne audyty bezpieczeństwa systemu co kwartał.

**4. Skalowalność:** Możliwość obsługi wzrostu liczby użytkowników do co najmniej 1000 bez utraty wydajności.Automatyczne skalowanie zasobów systemu w zależności od obciążenia.

**5. Zgodność z regulacjami:** Pełna zgodność systemu z obowiązującymi przepisami branżowymi oraz normami ISO związanych z produkcją.

**6. Interoperacyjność:** Łatwa integracja z istniejącymi systemami w zakładzie przemysłowym, w tym ERP i systemami finansowymi.

**7. Czas Pracy:** Dostępność systemu 24/7, umożliwiająca pracę w trybie ciągłym.Minimalne przerwy serwisowe i wsparcie dla pracy wielozmianowej.

**8. Utrzymanie:** System łatwy w utrzymaniu, z możliwością szybkiego wdrażania poprawek i aktualizacji.Dostarczenie dokumentacji technicznej i instrukcji obsługi.

**9. Elastyczność technologiczna:** System niezależny od konkretnej platformy technologicznej, umożliwiający ewentualną migrację na nowsze technologie.

**10. Dostępność szkoleń:** Dostępność szkoleń dla użytkowników podczas wdrożenia i w trakcie eksploatacji systemu.Wsparcie szkoleniowe dla nowych użytkowników.

**11. Odporność na błędy:** System powinien posiadać mechanizmy zapewniające odporność na awarie i szybkie przywracanie usług w przypadku błędów.

**12. Efektywne zarządzanie pamięcią:** Optymalne zarządzanie pamięcią systemu, minimalizujące jej zużycie i zapewniające stabilność działania.

# Cele biznesowe

- optymalizacja wydajności

- redukcja kosztów

- poprawa jakości i terminowość produkcji

- minimalizacja przestojów i awarii

- poprawa procesów podejmowania decyzji

- zachowanie bezpieczeństwa danych

Oprogramowanie dostosowane do celów zakładu przemysłowego może spełniać kluczowe funkcje w osiąganiu tych celów. Optymalizacja wydajności może być wspierana poprzez systemy monitorowania i analizy danych, umożliwiające identyfikację obszarów do usprawnienia. Redukcję kosztów zapewnią rozwiązania do zarządzania zasobami i procesami, umożliwiając lepszą kontrolę nad wydatkami. Poprawa jakości produkcji może wyniknąć z zastosowania systemów automatyki i kontroli jakości. Minimalizacja przestojów i awarii może być osiągnięta poprzez prognozowanie usterki i systemy utrzymania ruchu. Efektywne procesy podejmowania decyzji uzyskamy dzięki narzędziom BI (Business Intelligence) do analizy danych. Wreszcie, bezpieczeństwo danych będzie chronione przez systemy zarządzania bezpieczeństwem informacji oraz odpowiednie protokoły i zabezpieczenia sieciowe.

# Struktura Organizacyjno-Funkcjonalna zakładu przemysłowego

W organizacji zakładu przemysłowego można wyróżnić kilka kluczowych działów lub służb, związanych z różnymi aspektami funkcjonowania przedsiębiorstwa.

1. **Dział Planowania Produkcji:** Odpowiada za opracowywanie planów produkcyjnych, harmonogramowanie produkcji oraz optymalizację wykorzystania zasobów. Współpracuje z innymi działami w celu uwzględnienia zapotrzebowania rynku, dostępności surowców i terminów dostaw.

2. **Dział Analityki Danych:** Zajmuje się gromadzeniem, analizą i interpretacją danych związanych z procesami produkcyjnymi. Działa na rzecz optymalizacji procesów, identyfikacji obszarów do poprawy i dostarczania informacji wspierających podejmowanie decyzji.

3. **Dział Zakupów i Dostaw:** Odpowiada za zarządzanie łańcuchem dostaw, negocjacje z dostawcami, kontrolę jakości dostarczanych surowców oraz składowanie materiałów. Współpracuje z działem planowania produkcji w celu zapewnienia płynności dostaw.

4. **Dział Utrzymania Ruchu:** Zapewnia bieżącą konserwację maszyn i urządzeń, prowadzi działania prewencyjne w celu minimalizacji przestojów związanych z awariami technicznymi.

5. **Dział Kontroli Jakości:** Monitoruje i kontroluje jakość produktów w procesie produkcji, opracowuje standardy jakości, a także wprowadza działania korygujące w przypadku wykrycia niezgodności.

6. **Dział Inżynierii Produkcji:** Odpowiada za rozwój i doskonalenie procesów produkcyjnych, wprowadzanie nowych technologii, a także projektowanie układów produkcyjnych w celu zwiększenia efektywności.

7. **Dział Bezpieczeństwa Informatycznego:** Zajmuje się ochroną systemów informatycznych przed atakami oraz zarządzaniem bezpieczeństwem danych. Zapewnia, że informacje produkcyjne są chronione przed nieuprawnionym dostępem.

8. **Dział Zarządzania Projektem:** Wprowadza nowe inicjatywy i inwestycje, zarządza projektami mającymi na celu wprowadzenie nowych technologii lub usprawnień procesów.

Taka struktura organizacyjno-funkcjonalna pozwala na skoordynowane działania różnych służb, co przyczynia się do efektywnego zarządzania zakładem przemysłowym i osiągania zdefiniowanych celów.

# Produkty i usługi

Zakład przemysłowy oferuje różnorodne produkty i usługi, z których korzystają klienci na rynku. Dotychczasowe produkty mogą obejmować elementy skomplikowanych maszyn, komponenty elektroniczne, czy gotowe produkty konsumenckie. Usługi mogą obejmować obszar serwisu posprzedażowego, dostosowywania produktów do indywidualnych potrzeb klienta oraz szkolenia z obsługi sprzętu.

Nowe oprogramowanie wprowadzone do zakładu przemysłowego ma kluczowe znaczenie dla produkcji i świadczenia usług. Oto kilka aspektów, jakie mogą wpłynąć na ofertę zakładu:

1. **Automatyzacja Produkcji:** Oprogramowanie umożliwiające automatyzację procesów produkcyjnych może przyspieszyć tempo produkcji, eliminując błędy ludzkie i zapewniając spójność jakości produktów.

2. **Zarządzanie Łańcuchem Dostaw:** Systemy wspomagające zarządzanie dostawami i surowcami mogą skrócić czasy oczekiwania na komponenty, eliminując opóźnienia w produkcji.

3. **Personalizacja Produktów:** Oprogramowanie umożliwiające dostosowywanie produktów do indywidualnych potrzeb klienta pozwala na rozszerzenie oferty o spersonalizowane rozwiązania, co może zwiększyć atrakcyjność oferty.

4. **Monitorowanie Jakości:** Systemy monitorujące jakość w czasie rzeczywistym mogą przyczynić się do identyfikacji potencjalnych problemów produkcyjnych, co prowadzi do szybszych działań korygujących i poprawy jakości już w trakcie procesu produkcyjnego.

5. **Usługi Dodatkowe:** Nowe oprogramowanie może umożliwić świadczenie dodatkowych usług, takich jak zdalne wsparcie techniczne, analiza danych w czasie rzeczywistym czy monitorowanie parametrów pracy produktów.

W rezultacie, dzięki nowym możliwościom, jakość już produkowanych towarów może ulec poprawie poprzez lepszą kontrolę procesów, skrócenie czasów reakcji na ewentualne problemy oraz dostosowanie produktów do oczekiwań rynkowych. Wprowadzenie nowego oprogramowania nie tylko usprawni produkcję istniejących produktów, ale także otworzy drzwi do innowacyjnych rozwiązań i nowych obszarów biznesowych.

# Metody pracy i potrzeba zmiany

Metody pracy w zakładzie przemysłowym obejmują szereg procesów, których celem jest efektywna produkcja, utrzymanie wysokiej jakości produktów oraz zapewnienie płynności operacyjnej. Oto kilka kluczowych metod pracy w tego typu środowisku:

1. **Planowanie Produkcji:** Opracowywanie planów produkcyjnych uwzględniających ilość, terminy oraz rodzaje produktów. Planowanie odbywa się na podstawie analizy popytu rynkowego, dostępności surowców i zasobów.

2. **Zarządzanie Łańcuchem Dostaw:** Skoordynowane zarządzanie dostawami surowców i komponentów, aby zapewnić płynność procesu produkcyjnego. Obejmuje to monitorowanie dostaw, negocjacje z dostawcami i zarządzanie zapasami.

3. **Automatyzacja Procesów:** Wykorzystanie technologii automatyzacji w celu zwiększenia efektywności i precyzji produkcji. Automatyzacja może obejmować montaż, pakowanie, a także kontrolę jakości.

4. **Kontrola Jakości:** Stosowanie systemów kontroli jakości w różnych etapach produkcji, aby zapewnić zgodność z normami i oczekiwaniami klientów. Kontrola obejmuje testowanie produktów pod kątem jakości i wydajności.

5. **Utrzymanie Ruchu:** Planowanie i przeprowadzanie regularnych prac konserwacyjnych, naprawczych i prewencyjnych, aby utrzymać sprzęt w dobrej kondycji i zminimalizować przestoje.

6. **Szkolenia Pracowników:** Regularne szkolenia pracowników w zakresie obsługi maszyn, nowych technologii, zasad bezpieczeństwa oraz standardów jakości.

7. **Zastosowanie Lean Manufacturing:** Wdrażanie zasad Lean Manufacturing w celu eliminacji marnotrawstwa, optymalizacji procesów i skrócenia czasu cyklu produkcyjnego.

8. **Technologie IoT:** Wykorzystanie Internetu Rzeczy (IoT) do monitorowania parametrów produkcji w czasie rzeczywistym, zbierania danych oraz identyfikacji obszarów do optymalizacji.

9. **Zarządzanie Projektami:** Efektywne zarządzanie projektami w celu wprowadzenia nowych produktów, technologii czy procesów. Wymaga to planowania, monitorowania postępów i dostosowywania działań w trakcie realizacji projektu.

10. **Szerokie Wykorzystanie Technologii:** Zastosowanie zaawansowanych technologii, takich jak sztuczna inteligencja, analiza danych czy cyfrowa integracja, aby podnosić efektywność, prognozować awarie i doskonalić procesy.

11. **Zarządzanie Odpadami:** Optymalizacja gospodarki odpadami poprzez zastosowanie środków mających na celu redukcję ilości generowanych odpadów i ich efektywne przetwarzanie.

Zastosowanie tych metod pracy ma na celu nie tylko zwiększenie efektywności produkcji, ale również poprawę jakości produktów, bezpieczeństwa pracy oraz zrównoważony rozwój zakładu przemysłowego. Ciągła adaptacja do zmieniających się warunków rynkowych oraz nowych technologii jest kluczowa dla konkurencyjności i sukcesu w branży przemysłowej.

Potrzeby zmiany, udoskonalenia i zastosowania nowych metod pracy w zakładzie przemysłowym wynikają z różnych czynników mających wpływ na efektywność, jakość, zrównoważony rozwój oraz konkurencyjność przedsiębiorstwa. Oto kilka kluczowych aspektów, które uzasadniają potrzebę wprowadzenia zmian:

1. **Efektywność Produkcji:** Zmiany są konieczne w celu zwiększenia efektywności procesów produkcyjnych, skrócenia czasu cyklu i eliminacji marnotrawstwa. Wykorzystanie nowoczesnych technologii i automatyzacji może poprawić wydajność.

2. **Jakość Produktów:** Wzrost oczekiwań klientów pod względem jakości produktów wymaga ciągłego doskonalenia procesów kontroli jakości oraz wdrażania innowacyjnych metod produkcyjnych zapewniających lepszą jakość.

3. **Zrównoważony Rozwój:** Konieczność dostosowania metod pracy do standardów zrównoważonego rozwoju, co obejmuje ograniczenie zużycia surowców, minimalizację odpadów oraz wprowadzenie bardziej ekologicznych praktyk produkcyjnych.

4. **Adaptacja do Nowych Technologii:** Wprowadzenie nowych technologii, takich jak sztuczna inteligencja, analiza danych czy Internet Rzeczy, aby utrzymać konkurencyjność, prognozować awarie maszyn i dostosowywać produkcję do zmieniających się warunków.

5. **Elastyczność Produkcji:** Zmiana metod pracy w kierunku większej elastyczności, aby dostosować produkcję do zmieniających się trendów rynkowych, popytu klientów oraz dynamicznych warunków gospodarczych.

6. **Zarządzanie Łańcuchem Dostaw:** Potrzeba stałego doskonalenia zarządzania łańcuchem dostaw, aby skutecznie reagować na zmienne warunki rynkowe, uniknąć opóźnień w dostawach i utrzymać płynność produkcji.

7. **Szkolenia Pracowników:** Zapewnienie pracownikom regularnych szkoleń, aby dostosowywać ich umiejętności do nowych technologii i procedur, co przyczynia się do zwiększenia efektywności i bezpieczeństwa pracy.

8. **Zarządzanie Projektami:** Wprowadzenie nowych metod zarządzania projektami, które umożliwiają szybszą adaptację do zmian, monitorowanie postępów i skuteczne zarządzanie zasobami.

9. **Optymalizacja Kosztów:** Konieczność ciągłej optymalizacji kosztów produkcji poprzez identyfikację obszarów oszczędności, efektywne zarządzanie zasobami oraz minimalizację strat.

10. **Dostosowanie do Trendów Rynkowych:** Zmiany są konieczne, aby dostosowywać produkcję do nowych trendów rynkowych, zrozumienia potrzeb klientów oraz szybkiej adaptacji do konkurencyjnych rozwiązań.

Wprowadzenie zmian, udoskonaleń i zastosowanie nowych metod pracy powinno być oparte na ciągłym monitorowaniu środowiska biznesowego, innowacyjności i gotowości do dostosowywania się do dynamicznych zmian w branży przemysłowej.

# Podmioty zainteresowane

1. **Klienci:**

***Cele:*** Klienci dążą do zakupu produktów o wysokiej jakości, zgodnych z ich oczekiwaniami i spełniających określone wymagania. Często poszukują również konkurencyjnych cen, terminowości dostaw oraz efektywnego wsparcia posprzedażowego.

2. **Menadżerowie:**

***Cele:*** Menadżerowie zakładu przemysłowego skupiają się na efektywnym zarządzaniu produkcją, utrzymaniem płynności operacyjnej, zminimalizowaniu kosztów i osiągnięciu określonych wskaźników wydajności. Dążą również do wprowadzania innowacji i doskonalenia procesów.

3. **Pracownicy:**

***Cele:*** Pracownicy zakładu przemysłowego mają różnorodne cele, ale w ogólnym sensie dążą do pewności zatrudnienia, możliwości rozwoju zawodowego, uczestnictwa w bezpiecznych i zdrowych warunkach pracy oraz uczucia satysfakcji z wykonywanej pracy.

4. **Inwestorzy:**

***Cele:*** Inwestorzy zazwyczaj dążą do osiągnięcia zwrotu z zainwestowanych środków, maksymalizacji wartości akcji, zrównoważonego wzrostu oraz minimalizacji ryzyka finansowego. Interesuje ich rentowność przedsiębiorstwa.

5. **Właściciele:**

***Cele:*** Właściciele zakładu przemysłowego często mają cele związane z długoterminowym sukcesem przedsiębiorstwa, zwiększaniem jego wartości oraz utrzymaniem silnej pozycji na rynku. Mogą również dążyć do rozwijania firmy w kierunku innowacji i zrównoważonego rozwoju.

6. **Społeczność lokalna:**

***Cele:*** Społeczność lokalna może dążyć do współpracy z zakładem przemysłowym w celu tworzenia miejsc pracy, zrównoważonego rozwoju, minimalizacji wpływu na środowisko, a także poprawy relacji społecznych i edukacji.

7. **Rząd i Organizacje Regulacyjne:**

***Cele:*** Rząd i organizacje regulacyjne mają cele związane z przestrzeganiem prawa, bezpieczeństwem pracowników, ochroną środowiska, kontrolą jakości produktów oraz wspieraniem gospodarki.

8. **Dostawcy:**

***Cele:*** Dostawcy dążą do utrzymania długoterminowych relacji z zakładem przemysłowym, zapewnienia terminowych dostaw surowców i komponentów, a także uczestnictwa w rozwijających się projektach.

Zrozumienie celów różnych interesariuszy jest kluczowe dla skutecznego zarządzania przedsiębiorstwem, ponieważ pozwala na dostosowanie strategii i działań do różnorodnych oczekiwań i potrzeb każdej grupy.

# Analiza ograniczeń

1. ***Ograniczenie technologiczne –***  Istniejące systemy lub infrastruktura IT w zakładzie mogą być niekompatybilne z nowym systemem zarządzania zasobami. Może to wymagać modernizacji lub integracji z istniejącymi rozwiązaniami, co może być kosztowne i czasochłonne.

1. ***Ograniczenie finansowe –*** Budżet przeznaczony na ten projekt może być ograniczony, co może wpływać na zakres i możliwości implementacyjne systemu.
2. ***Ograniczenie czasowe –*** Wdrożenie kompleksowego systemu zarządzania może zająć więcej czasu niż początkowo zakładano, szczególnie jeśli są potrzebne dostosowania do specyficznych potrzeb zakładu.
3. ***Opór ze strony pracowników –*** Wprowadzenie nowego systemu może spotkać się z oporem ze strony pracowników, którzy mogą mieć trudności w adaptacji lub obawiać się utraty pracy w związku z automatyzacją procesów.
4. ***Ograniczenie w integracji danych –*** Dane dotyczące zasobów ludzkich, sprzętowych i magazynowych mogą być przechowywane w różnych systemach lub być w różnych formatach, co może utrudnić ich integrację i współpracę.
5. ***Problemy z jakością danych –*** Istniejące dane mogą być niekompletne, nieaktualne lub niespójne, co może utrudnić ich efektywne wykorzystanie w nowym systemie.
6. ***Zmiany regulacyjne –*** Rygorystyczne przepisy lub zmiany w regulacjach branżowych mogą wpłynąć na projekt, wymagając spełnienia nowych standardów bezpieczeństwa danych lub raportowania.
7. ***Ryzyko wdrożenia -*** Istnieje ryzyko, że nowy system może nie spełnić oczekiwań lub może pojawić się potrzeba nieprzewidzianych dostosowań lub zmian w trakcie wdrażania.

|  |  |  |
| --- | --- | --- |
| Ograniczenie | Dotkliwość ograniczenia | Możliwe rozwiązania problemu |
| Ograniczenie technologiczne | wysoka | * dokładna analiza istniejącej infrastruktury i zidentyfikowanie niezbędnych aktualizacji * modernizacja istniejącej infrastruktury * wprowadzenie stopniowej modernizacji * dostosowanie systemu do aktualnych rozwiązań |
| Ograniczenie finansowe | wysoka | * podział całości na moduły wdrażane w różnym okresie czasu * wybranie krytycznego obszaru do realizacji * ustalenie priorytetów i redukcja wymagań |
| Ograniczenie czasowe | średnia | * opracowanie harmonogramu projektu, uwzględniając ewentualne opóźnienia * zwiększenie zespołu do pracy nad projektem * outsourcing pracowniczy |
| Opór ze strony pracowników | niska | * analiza potrzeb pracowników * intuicyjny łatwy w obsłudze system * odpowiednie szkolenia dla pracowników w celu zminimalizowania obaw związanych z nowym systemem * wsparcie powdrożeniowe |
| Ograniczenie w integracji danych | średnia | * uwzględnienie w projektowaniu system różnych źródeł danych * zastosowanie narzędzi do integracji danych, umożliwiające konwersję różnych formatów i synchronizację danych * wcześniejsze przygotowanie i ujednolicenie danych |
| Problemy z jakością danych | średnia | * ocena jakości danych i procesy oczyszczania danych przed migracją. * system monitorowania jakości danych po wdrożeniu |
| Zmiany regulacyjne | średnia | * analiza przepisów prawnych i regulacji pod kątem projektowania systemu * współpraca z ekspertami prawnymi, aby upewnić się, że system jest zgodny z obowiązującymi przepisami * uwzględnienie przyszłych aktualizacji systemu pod względem zamian w przepisach |
| Ryzyko wdrożenia | średnia | * dokładna analiza potrzeb przyszłego użytkownika * dokładne testy przed wdrożeniem, aby zidentyfikować i rozwiązać potencjalne problemy * skorzystanie z metodologii Agile, umożliwiającej dostosowanie się do zmian w trakcie projektu * ciągła komunikacja ze zleceniodawcą, aby szybko reagować na zmiany i problemy |

# Analiza ryzyka

Baza danych

- zaburzenie integralności

- wyciek danych

- błąd zapisu/odczytu

- nieautoryzowany dostęp

Serwer

- wirusy i ataki hackerskie

- niespodziewane zatrzymanie systemu

- awaria elektryczności

- awaria sprzętu

- błędy ludzkie

- fizyczne naruszenie integralności struktury serwera

Użytkownicy

- błędne użytkowanie systemu

- nieautoryzowany dostęp do systemu

Aplikacja

- brak kompatybilności wstecznej

- nieaktualne oprogramowanie

- nieprawidłowe użytkowanie

- błędy w kodzie

Projektowanie systemu

- przekroczenie budżetu

- niedotrzymanie terminów

- niedostateczna analiza wymagań

- brak jasności w architekturze

- błędne zrozumienie specyfikacji

- brak dostatecznej wiedzy na temat projektowania takiego systemu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ryzyko | Prawdopodo  bieństwo | Dotkliwość ryzyka | Możliwe rozwiązania problemu |
| Baza danych | zaburzenie integralności | Średnie | Wysoka | Regularne sprawdzanie spójności danych, zastosowanie mechanizmów kontroli dostępu |
| wyciek danych | Średnie | Wysoka | Zastosowanie silnych mechanizmów szyfrowania, regularne audyty bezpieczeństwa, monitorowanie ruchu sieciowego |
| błąd zapisu/odczytu | Niskie | Średnia | Regularne testy funkcjonalne, tworzenie kopii zapasowych danych, implementacja mechanizmów sprawdzających poprawność operacji na danych |
| nieautoryzowany dostęp | Średnie | Wysoka | Wdrożenie silnego systemu uwierzytelniania, monitorowanie logów dostępu, regularna analiza dostępów |
| Serwer | wirusy i ataki hackerskie | Wysokie | Wysoka | Aktualizacja regularna systemu i oprogramowania, stosowanie firewalli, regularne skanowanie antywirusowe |
| niespodziewane zatrzymanie systemu | Średnie | Wysoka | Użycie systemów z podwójnym źródłem zasilania, backupy systemu, monitoring wydajności |
| awaria elektryczności | Niskie | Wysoka | Użycie zasilaczy awaryjnych (UPS), planowanie regularnych testów awaryjnych |
| awaria sprzętu | Średnie | Średnia | Regularna konserwacja sprzętu, utrzymanie zasobów zapasowych, umowa na serwis |
| błędy ludzkie | Średnie | Średnia | Szkolenia pracowników, implementacja procedur bezpieczeństwa, podwójna weryfikacja kluczowych operacji |
| fizyczne naruszenie integralności struktury serwera | Niskie | Wysoka | Fizyczne zabezpieczenia serwerowni, monitorowanie dostępu do pomieszczenia serwerowego |
| Użytkownicy | błędne użytkowanie systemu | Średnie | Średnia | Szkolenia dla użytkowników, tworzenie i udostępnianie dokumentacji dotyczącej poprawnego użytkowania systemu |
| nieautoryzowany dostęp do systemu | średnie | Wysoka | Wdrożenie silnych mechanizmów uwierzytelniania, monitorowanie aktywności użytkowników, restrykcyjne uprawnienia |
| Aplikacja | brak kompatybilności wstecznej | Niskie | Średnia | Planowanie regularnych aktualizacji, utrzymanie dokumentacji zmian w aplikacji |
| nieaktualne oprogramowanie | Niskie | Niska | Automatyczne powiadomienia o dostępnych aktualizacjach, procedury aktualizacji oprogramowania |
| nieprawidłowe użytkowanie | Średnie | Średnia | Szkolenia dla użytkowników, interfejs użytkownika zaprojektowany w sposób intuicyjny |
| błędy w kodzie | Niska | Wysokie | Regularne przeglądy kodu, stosowanie testów jednostkowych, wdrażanie procedur debugowania |
| Projektowanie systemu | przekroczenie budżetu | Średnie | Wysoka | Staranne planowanie budżetu, monitorowanie kosztów na bieżąco, rezerwowanie środków na nieprzewidziane wydatki |
| niedotrzymanie terminów | Średnie | Wysoka | Realistyczne planowanie czasu, stosowanie metodyk zarządzania projektem, regularne monitorowanie postępów |
| niedostateczna analiza wymagań | Średnie | Wysoka | Staranne zbieranie i analizowanie wymagań, regularne spotkania z klientem, stosowanie prototypowania |
| brak jasności w architekturze | Średnie | Średnia | Dokładna dokumentacja architektoniczna, regularne przeglądy projektu, stosowanie zrozumiałych wzorców projektowych |
| błędne zrozumienie specyfikacji | Średnie | Wysoka | Aktywne pytania i klarowanie wątpliwości w trakcie analizy specyfikacji, regularne spotkania z klientem |
| brak dostatecznej wiedzy na temat projektowania takiego systemu | Niskie | Wysoka | Szkolenia dla zespołu, współpraca z doświadczonymi ekspertami, analiza benchmarków branżowych |

# Analiza wzorców architektonicznych

## Architektura monolityczna

Architektura monolityczna to tradycyjny model projektowania oprogramowania, w którym cała aplikacja jest rozwijana jako jeden spójny blok, zazwyczaj jako pojedyncza aplikacja. Charakteryzuje się ona tym, że cały kod, logika biznesowa oraz interfejs użytkownika są zintegrowane w jednym monolicie.

**Główne cechy architektury monolitycznej:**

* Jeden duży blok: Cała aplikacja jest rozwijana jako jeden, spójny blok kodu. Wszystkie komponenty, takie jak interfejs użytkownika, logika biznesowa i warstwa dostępu do danych, są połączone w jednym miejscu.
* Łatwość w rozwoju i wdrażaniu: Ze względu na to, że cały system znajduje się w jednym miejscu, proces rozwijania i wdrażania aplikacji może być stosunkowo prosty i przejrzysty.
* Bezpośrednie wywołania funkcji: Komunikacja między różnymi komponentami aplikacji odbywa się bezpośrednio poprzez wywołania funkcji lub procedur w ramach jednego systemu.
* Prosta architektura: Zwykle architektura monolityczna nie jest skomplikowana, co ułatwia zrozumienie i utrzymanie systemu.
* Problemy skalowalności: Trudności związane ze skalowalnością, ponieważ aplikacja jako całość musi być skalowana, nawet jeśli tylko część systemu jest obciążona.
* Ryzyko jednego punktu awarii: W przypadku awarii lub błędu w jednej części aplikacji, cały system może być niedostępny, co ma potencjał znacznego wpływu na działanie całego zakładu przemysłowego.
* Ograniczona elastyczność: Modyfikacje w jednym obszarze aplikacji mogą potencjalnie wpływać na inne, co może prowadzić do trudności w adaptacji i rozbudowie systemu.

Architektura monolityczna jest łatwa do zrozumienia i rozwijania, szczególnie w mniejszych projektach. Jednak w przypadku większych i bardziej złożonych systemów, może ograniczać elastyczność i skalowalność, co może być istotne w przypadku dużego zakładu przemysłowego, gdzie różnorodność procesów i potrzeb może być znacząca.

## Architektura mikroserwisowa

Architektura mikroserwisowa to podejście do budowy oprogramowania, w którym aplikacja jest rozbijana na mniejsze, niezależne serwisy (mikroserwisy), z których każdy obsługuje jedną lub kilka powiązanych funkcji biznesowych. Każdy mikroserwis jest rozwijany, wdrażany, skalowany i zarządzany niezależnie od pozostałych, co daje większą elastyczność i możliwość modularyzacji systemu.

**Główne cechy architektury mikroserwisowej:**

* Modularyzacja aplikacji: Aplikacja jest podzielona na wiele mikroserwisów, z których każdy odpowiada za określone zadania lub funkcjonalności. Każdy mikroserwis działa jako osobna jednostka biznesowa.
* Niezależne wdrożenie i skalowanie: Każdy mikroserwis może być rozwijany, testowany, wdrażany i skalowany niezależnie. To oznacza, że można zwiększać zasoby tylko tam, gdzie są potrzebne, co poprawia wydajność i elastyczność systemu.
* Komunikacja przez interfejsy: Mikroserwisy komunikują się ze sobą przez interfejsy, często za pomocą protokołów komunikacyjnych, takich jak RESTful API lub komunikacja asynchroniczna. Łatwiejsze utrzymanie: Ze względu na modularyzację, mikroserwisy są łatwiejsze do zarządzania, utrzymania i aktualizacji. Wprowadzanie zmian w jednym mikroserwisie nie wpływa na całość systemu.
* Złożoność infrastruktury: Wymaga bardziej zaawansowanej infrastruktury, ponieważ zarządzanie wieloma serwisami wymaga odpowiednich narzędzi do monitorowania, skalowania i zarządzania.
* Zarządzanie komunikacją między serwisami: Istotne jest odpowiednie zarządzanie komunikacją między serwisami, aby zapewnić spójność i integralność danych, a także uniknąć opóźnień w komunikacji.
* Elastyczność i skalowalność: Architektura mikroserwisowa umożliwia łatwiejsze dostosowywanie aplikacji do zmieniających się potrzeb i wymagań biznesowych. Poszczególne mikroserwisy mogą być skalowane niezależnie.

Architektura mikroserwisowa jest szczególnie korzystna w dużych systemach, gdzie różne części aplikacji mają różne wymagania i muszą być rozwijane, skalowane i utrzymywane niezależnie. W zakładzie przemysłowym może to być korzystne ze względu na zróżnicowane procesy i obszary funkcjonalne, co wymaga elastyczności i niezależności poszczególnych elementów systemu.

## Architektura oparta na wydarzeniach

Architektura oparta na wydarzeniach (ang. Event-Driven Architecture - EDA) to podejście projektowe, w którym system jest zbudowany wokół generowania, przechwytywania, przetwarzania i reagowania na zdarzenia, które mają miejsce w systemie lub w otaczającym środowisku. W EDA zdarzenia są kluczowymi punktami komunikacji i wyzwalaczami akcji w systemie.

**Główne cechy architektury opartej na wydarzeniach:**

* Zdarzenia jako fundament komunikacji: System skupia się na przekazywaniu zdarzeń (eventów), które reprezentują stan, działanie lub sytuację w systemie. Te zdarzenia są przesyłane między komponentami w systemie.
* Asynchroniczność: Komunikacja między komponentami odbywa się asynchronicznie poprzez zdarzenia. Komponenty reagują na zdarzenia, które otrzymują, niezależnie od innych elementów systemu.
* Luźne powiązania: Komponenty są luźno powiązane, co oznacza, że wysyłanie zdarzeń nie wymaga bezpośredniej komunikacji między komponentami. Komponenty nie muszą wiedzieć, kto odbiera zdarzenie ani co się z nim dzieje.
* Skalowalność i elastyczność: Architektura oparta na wydarzeniach jest elastyczna i skalowalna, ponieważ komponenty mogą reagować na zdarzenia niezależnie. Można dodawać nowe komponenty lub funkcjonalności bez wpływu na całość systemu.
* Real-time processing: Jest przydatna w aplikacjach, które wymagają szybkiej reakcji na zdarzenia w czasie rzeczywistym, takich jak systemy monitorowania, analizy danych w czasie rzeczywistym itp.
* Zarządzanie kolejnością i integralnością danych: Istotne jest zapewnienie odpowiedniego zarządzania kolejnością i integralnością danych, aby uniknąć utraty zdarzeń lub ich nieprawidłowej interpretacji.

Architektura oparta na wydarzeniach jest często stosowana w systemach monitorowania, analizy danych w czasie rzeczywistym, systemach zarządzania zdarzeniowego, gdzie istotne jest natychmiastowe reagowanie na zmiany w systemie lub otoczeniu. W przypadku zakładu przemysłowego może być użyteczna w monitorowaniu produkcji, reagowaniu na awarie, dynamicznej optymalizacji procesów lub przewidywaniu potrzeb produkcyjnych.

## Wybór architektury

Dla projektu obejmującego zarządzanie pracą produkcyjną oraz różnymi modułami, takimi jak magazyn, utrzymanie ruchu, zakupy, logistyka i HR, mogłaby być korzystna architektura oparta na mikroserwisach.

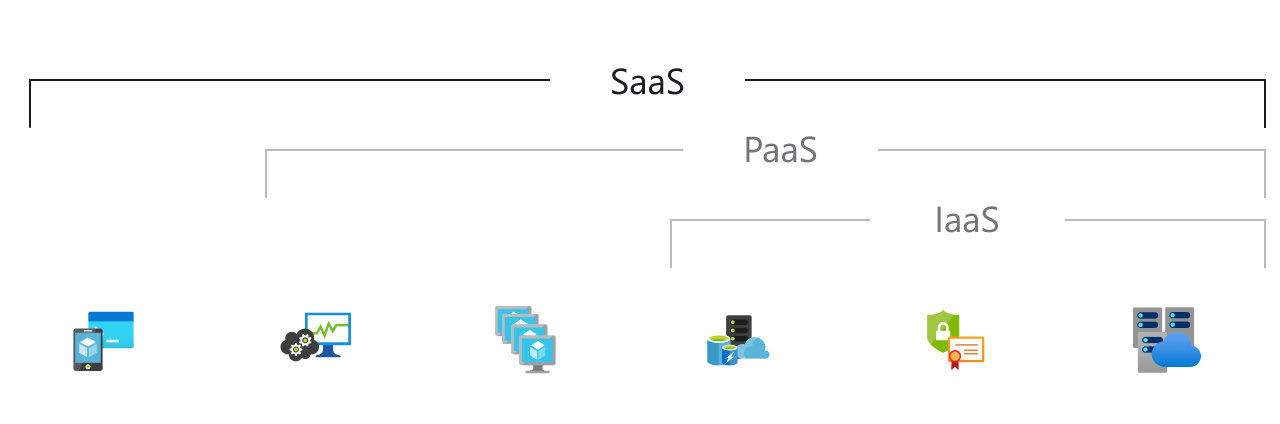
* Mikroserwisy pozwalają na tworzenie niezależnych modułów, co jest zgodne z potrzebą tworzenia różnych modułów dla różnych obszarów funkcjonalnych w zakładzie przemysłowym. Każda mikrousługa może obsługiwać inny aspekt zarządzania zasobami.
* Każda mikrousługa może być wdrażana, aktualizowana i skalowana niezależnie. To pozwala na szybkie dostarczanie nowych funkcji i łatwe zarządzanie cyklem życia oprogramowania dla różnych obszarów.
* Niezależność mikrousług ułatwia testowanie i utrzymanie. Każda usługa może być testowana oddzielnie, a awaria jednej usługi nie wpływa na pozostałe.
* Mikroserwisy pozwalają na łatwe integrowanie się z istniejącymi systemami, co jest istotne w przypadku kompleksowych systemów zarządzania zakładem przemysłowym.

## Analiza infrastruktur projektu

**IaaS** **-** to typ usługi przetwarzania w chmurze, która oferuje podstawowe zasoby obliczeniowe, magazynowe i sieciowe na żądanie z płatnością zgodnie z rzeczywistym użyciem. IaaS to jeden z czterech typów usług w chmurze, wraz z oprogramowaniem jako usługą i przetwarzaniem bez serwerowym.

**PaaS** **-** to kompletne środowisko deweloperskie i wdrożeniowe w chmurze obejmujące zasoby umożliwiające dostarczanie dowolnego rozwiązania, od prostych aplikacji opartych na chmurze po złożone aplikacje dla przedsiębiorstw korzystające z chmury. Rozwiązanie PaaS pomaga uniknąć wydatków i pracy związanych z zakupem licencji na oprogramowanie, podstawowej infrastruktury aplikacji i oprogramowania pośredniczącego, orkiestratorów kontenerów, takich jak Kubernetes, lub narzędzi deweloperskich i innych zasobów oraz zarządzaniem nimi. Pozwala na zarządzanie opracowanymi aplikacjami i usługami, natomiast reszta obszarów jest objęta zakresem działań dostawcy usług w chmurze.

**SaaS -** to kompletne rozwiązanie programowe kupowane w modelu płatności zgodnie z rzeczywistym użyciem od dostawcy usług w chmurze. Płatność jest dokonywana za możliwość używania aplikacji w organizacji, a użytkownicy łączą się z nią przez Internet, zazwyczaj przy użyciu przeglądarki sieci Web. Cała podstawowa infrastruktura, oprogramowanie pośredniczące oraz oprogramowanie i dane aplikacji znajdują się w centrum danych dostawcy usług. Dostawca usług zarządza sprzętem i oprogramowaniem, a po zawarciu odpowiedniej umowy o świadczenie usług zapewni także dostępność i bezpieczeństwo aplikacji oraz Twoich danych. Rozwiązanie SaaS pozwala organizacjom szybko zacząć korzystać z aplikacji przy minimalnym koszcie wstępnym.



## Wybór infrastruktury projektu

W naszym konkretnym przypadku najlepiej sprawdzi się infrastruktura SaaS. Zakład otrzyma dostęp do funkcjonalności którą konkretnie potrzebuje. Model ten pozwoli na sprzedaż oprogramowania większej ilości klientów, gdyż jest on bardziej atrakcyjny poprzez brak konieczności posiadania własnej infrastruktury IT. W tym modelu klient ma dostęp do potrzebnych funkcjonalności na żądanie. Klient będzie jedynie płacił za dostęp do aplikacji co zwalnia go z nabywania licencji. Infrastruktura zostanie zapewniona przez zewnętrznego dostawcę. Na tej infrastrukturze zostanie uruchomiona część serwerowa aplikacji oraz baza danych. Klient po otrzymaniu aplikacji klienckiej będzie tylko musiał połączyć się z serwerem dzięki czemu uzyska dostęp do wykupionych funkcjonalności. Zewnętrzny dostawca zapewni też bezpieczeństwo i niezawodność systemu poprzez systematyczne tworzenie kopi zapasowych, aktualizację oprogramowania części serwerowej oraz monitoring systemu.

## Analiza architektury za pomocą trzech wzorców

### Wzorzec Komponentowo-Łącznikowy:

1. ***Komponenty Mikroserwisów:*** Każdy mikroserwis jest niezależnym komponentem, oferującym jedną, dobrze zdefiniowaną funkcjonalność.

2. ***Łączniki:*** Komunikacja między mikroserwisami odbywa się poprzez interfejsy, często wykorzystując protokoły takie jak HTTP/REST lub message brokery.

3. ***Niezależność Rozwoju i Wdrożeń:*** Każdy mikroserwis może być rozwijany, testowany i wdrażany niezależnie, co zwiększa elastyczność i przyśpiesza cykl życia oprogramowania.

### Wzorzec Modułowy:

1. ***Moduły Mikroserwisów:*** Mikroserwisy są rozbudowane jako oddzielne moduły, które można rozwijać, aktualizować i wymieniać niezależnie od reszty systemu.

2. ***Interfejsy Modułowe:*** Każdy mikroserwis posiada zdefiniowane interfejsy, precyzyjnie opisujące, jakie usługi oferuje innym komponentom systemu.

3. ***Podział Odpowiedzialności:*** Modułowość pozwala na przypisanie jednoznacznych obszarów odpowiedzialności dla każdego mikroserwisu, ułatwiając zarządzanie systemem.

### Wzorzec Alokacyjny dla Architektury Mikroserwisów w SaaS:

1. ***Rodzaj Produktu:*** SaaS (Software as a Service): Dostarczanie oprogramowania jako usługi, bez konieczności zarządzania infrastrukturą przez klienta.

2. ***Rodzaj Infrastruktury:*** Infrastruktura Chmurowa (Chmura Obliczeniowa): Wybór chmury obliczeniowej jako platformy do dostarczania i hostowania mikroserwisów.

**Opis Wzorca Alokacyjnego:**

1. ***Chmura Obliczeniowa:*** Korzystanie z chmury obliczeniowej, na przykład AWS, Azure lub Google Cloud, umożliwia dynamiczną alokację zasobów w zależności od aktualnego obciążenia. To pozwala na skalowanie mikroserwisów w sposób elastyczny, zgodnie z bieżącymi potrzebami.

2. ***Usługi Zarządzania Kontenerami:*** Wykorzystanie narzędzi do zarządzania kontenerami, takich jak Kubernetes, w celu efektywnego zarządzania cyklem życia mikroserwisów, wdrażania i skalowania.

3. ***Automatyzacja Procesów:*** Wykorzystanie narzędzi do automatycznego wdrażania (CI/CD), takich jak Jenkins czy GitLab CI, w celu automatyzacji procesu aktualizacji i wdrożeń mikroserwisów.

4. ***Monitoring i Analiza w Chmurze:*** Implementacja narzędzi monitorujących, takich jak AWS CloudWatch lub Azure Monitor, w celu śledzenia wydajności, dostępności i błędów mikroserwisów.

5. ***Bezpieczeństwo Wbudowane w Usługi Chmurowe:*** Skorzystanie z wbudowanych funkcji bezpieczeństwa oferowanych przez dostawców chmury, takich jak autoryzacja, szyfrowanie i zarządzanie dostępem.

6. ***Zarządzanie Konfiguracją:*** Użycie usług do zarządzania konfiguracją, co pozwala na jednolitą konfigurację i aktualizację mikroserwisów.

Wzorzec alokacyjny w kontekście SaaS i chmury obliczeniowej pozwala na efektywne zarządzanie zasobami, bez konieczności bezpośredniego zaangażowania klienta w obszarze infrastruktury. Dostawca usługi (SaaS) jest odpowiedzialny za zapewnienie skalowalności, bezpieczeństwa i dostępności mikroserwisów, co pozwala klientom skoncentrować się na korzystaniu z oprogramowania bez obaw o infrastrukturę.

Wzorzec mikroserwisowy oparty na tych trzech wzorcach pozwala na elastyczną, modularną i niezależną od dostawcy infrastrukturę, co jest kluczowe dla dynamicznych i skalowalnych systemów. Każdy mikroserwis jest samodzielnym komponentem, posiadającym jasno określone obszary odpowiedzialności i interfejsy komunikacyjne, co ułatwia rozwój, utrzymanie i skalowanie systemu. Alokacyjna elastyczność w chmurze obliczeniowej pozwala dostosować zasoby do bieżących potrzeb, co wpisuje się w ducha architektury mikroserwisowej.

# Specyfikacja implementacyjna

Zaimplementowano projekt z wykorzystaniem wielowątkowości w oparciu o wątki Runnable, wykorzystano połączenie TCP między serwerem a klientami oraz zastosowano bazę danych MariaDB a do utworzenia interfejsu użytkownika wykorzystano javaFX. Poniżej zostanie opisany sposób implementacji i oraz wyniki działania aplikacji.

## Schemat bazy danych – diagram ERD

Obraz zawierający tekst, pismo odręczne, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

## Implementacja bazy danych

Implementacja bazy danych z wykorzystaniem XAMPP i MariaDB zgonie z przedstawionym wyżej diagramem ERD

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

Baza danych została zaimplementowana tak aby możliwa była realizacja wszystkich niezbędnych elementów projektu. W bazie tej zastosowano klucze główne do unikatowej identyfikacji rekordów, klucze obce do powiązania relacji w bazie danych między tabelami. Wykorzystano również różne typy relacji: jeden-do-jeden, jeden-do-wielu oraz wiele-do-wielu co spowodowało konieczność stworzenia tabel łączących tabele w tej ostatniej relacji. Następnie w bazie danych wprowadzono niezbędne dane początkowe bez których aplikacja by nie działała.

## Diagram Klas

Obraz zawierający tekst, zrzut ekranu, Czcionka, Równolegle

Opis wygenerowany automatycznie

## Diagram Przypadków Użycia

Obraz zawierający tekst, pismo odręczne, diagram, Czcionka

Opis wygenerowany automatycznie

## Diagram Przepływu Danych – DFD

**Poziom 0**

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

**Poziom 1**

Obraz zawierający tekst, diagram, Plan, zrzut ekranu

Opis wygenerowany automatycznie

**Poziom 2**

**Obraz zawierający tekst, zrzut ekranu, diagram, Plan

Opis wygenerowany automatycznie**

Obraz zawierający diagram, tekst, Plan, zrzut ekranu

Opis wygenerowany automatycznie

**Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie**

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

**Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie**

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

**Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie**

Obraz zawierający tekst, zrzut ekranu, diagram, Plan

Opis wygenerowany automatycznie

## Implementacja aplikacji

Aplikacja została podzielona na 3 główne pakiety.

- **Client** – pakiet w którym znajduje się implementacja części klienckiej w której możemy wyróżnić dodatkowy pakiet ***controler***  do przechowywania klas odpowiedzialnych za kontrolery widoków. Struktura tego pakietu przedstawiona jest poniżej:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

- **Server** – pakiet w którym znajduje się implementacja części serwerowej aplikacji. Struktura poniżej:

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

- **View** – pakiet w którym znajdują się pliki fxml odpowiedzialne za widoki klienckie aplikacji. Struktura i zawartość poniżej:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

# Opis implementacji

## Klient

### Klasa PasswordEncryptor

package client;  
  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
  
public class PasswordEncryptor {  
  
 public static String encryptPassword(String password) {  
 try {  
 // Tworzymy instancję obiektu MessageDigest z algorytmem SHA-256  
 MessageDigest digest = MessageDigest.*getInstance*("SHA-256");  
  
 // Przekazujemy hasło do obiektu MessageDigest z użyciem UTF-8  
 byte[] hash = digest.digest(password.getBytes(StandardCharsets.*UTF\_8*));  
  
 // Konwertujemy wynikowy bajtowy hash do postaci szesnastkowej (hex)  
 StringBuilder hexString = new StringBuilder();  
 for (byte b : hash) {  
 String hex = Integer.*toHexString*(0xff & b);  
 if (hex.length() == 1) {  
 hexString.append('0');  
 }  
 hexString.append(hex);  
 }  
  
 // Zwracamy zaszyfrowane hasło  
 return hexString.toString();  
  
 } catch (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
 // Tutaj możesz obsłużyć wyjątek odpowiednio do twojej aplikacji  
 return null;  
 }  
 }  
}

Klasa ta zawiera metodę ***encryptPassword,*** która służy do szyfrowania haseł za pomocą algorytmu SHA-256.

### Klasa SceneManager

package client;  
  
import client.controller.\*;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import server.Employee;  
import server.Equipment;  
import server.Order;  
import server.Task;  
  
  
import java.io.IOException;  
import java.util.List;  
  
public class SceneManager {  
 private Stage primaryStage;  
 public SceneManager(Stage primaryStage){  
 this.primaryStage = primaryStage;  
 }  
 public void showLoginScene(TCPClientFX tcpClientFX) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/LoginScene.fxml"));  
 LoginController loginController = new LoginController(tcpClientFX);  
 loader.setController(loginController);  
 Parent root = loader.load();  
 primaryStage.setScene(new Scene(root));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void showAdminScene(TCPClientFX tcpClientFX, Employee employee, List<Employee> employees) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/AdminScene.fxml"));  
 AdminController adminController = new AdminController(tcpClientFX,employee, employees);  
 loader.setController(adminController);  
 Parent root = loader.load();  
 primaryStage.setScene(new Scene(root));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void showEmployeeScene(TCPClientFX tcpClientFX, Employee employee, List<Task> tasks) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/EmployeeScene.fxml"));  
 EmployeeController employeeController = new EmployeeController(tcpClientFX,employee, tasks);  
 loader.setController(employeeController);  
 Parent root = loader.load();  
 primaryStage.setScene(new Scene(root));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void showLeaderScene(TCPClientFX tcpClientFX, Employee employee) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/LeaderScene.fxml"));  
 LeaderController leaderController = new LeaderController(tcpClientFX,employee);  
 loader.setController(leaderController);  
 Parent root = loader.load();  
 primaryStage.setScene(new Scene(root));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void showManagerScene(TCPClientFX tcpClientFX, Employee employee, List<Order> orders, List<Task> tasks, List<Employee> employees, List<Equipment> equipmentList) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/ManagerScene.fxml"));  
 ManagerController managerController = new ManagerController(tcpClientFX,employee,orders, tasks,employees,equipmentList);  
 loader.setController(managerController);  
 Parent root = loader.load();  
 primaryStage.setScene(new Scene(root));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 public void showErrorScene(TCPClientFX tcpClientFX) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/ErrorScene.fxml"));  
 ErrorController errorController = new ErrorController(tcpClientFX);  
 loader.setController(errorController);  
 Parent root = loader.load();  
 primaryStage.setScene(new Scene(root));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 public void closeApp(){  
 primaryStage.close();  
 }  
}

SceneManager to klasa odpowiedzialna za zarządzanie widokami w interfejsie użytkownika. Przyjmuje obiekt ***Stage*** w konstruktorze, reprezentujący główne okno aplikacji. Metody tej klasy, takie jak showLoginScene, showAdminScene czy showManagerScene, pozwalają na przejście między różnymi widokami aplikacji w zależności od uprawnień użytkownika. Każda z tych metod tworzy obiekt FXMLLoader do ładowania plików FXML reprezentujących poszczególne sceny, przypisuje do nich odpowiednich kontrolerów, a następnie ustawia nowy widok w głównym oknie. Ta struktura umożliwia dynamiczne przełączanie się między różnymi interfejsami w trakcie działania aplikacji.

### Klasa TCPClientFX

package client;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import client.controller.LoginController;  
import server.Employee;  
import server.Equipment;  
import server.Order;  
import server.Task;  
  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class TCPClientFX extends Application {  
 private String username;  
 private String password;  
 private SceneManager sceneManager;  
 private Socket socket;  
 private Scanner in;  
 private PrintWriter out;  
 private ObjectInputStream objectInputStream;  
 private ObjectOutputStream objectOutputStream;  
  
  
  
 @Override  
 public void start(Stage primaryStage) throws InterruptedException {  
 try {  
  
  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/LoginScene.fxml"));  
 LoginController loginController = new LoginController(this);  
 loader.setController(loginController);  
  
 Parent root = loader.load();  
 Scene scene = new Scene(root);  
 scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());  
 primaryStage.setTitle("Welcome");  
 primaryStage.setScene(scene);  
 primaryStage.show();  
  
 sceneManager = new SceneManager(primaryStage);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void setLoginData(String username, String password) {  
  
 this.username = username;  
 this.password = password;  
 try {  
 socket = new Socket("localhost", 12345);  
 in = new Scanner(socket.getInputStream());  
 out = new PrintWriter(socket.getOutputStream(), true);  
 objectOutputStream = new ObjectOutputStream(socket.getOutputStream());  
 objectInputStream =new ObjectInputStream(socket.getInputStream());  
 objectOutputStream.writeObject(username);  
 objectOutputStream.writeObject(password);  
  
  
 Employee employee = (Employee) objectInputStream.readObject();  
 if (employee != null) {  
 switch (employee.getRole()) {  
 case "Production Employee":  
 List<Task> employeeTasks = (List<Task>) objectInputStream.readObject();  
 employeeTasks.forEach(task -> System.*out*.println(task.getName()));  
 sceneManager.showEmployeeScene(this, employee,employeeTasks);  
 break;  
 case "Admin": {  
 List<Employee> employees = (List<Employee>) objectInputStream.readObject();  
 sceneManager.showAdminScene(this, employee, employees);  
 break;  
 }  
 case "Leader":  
 sceneManager.showLeaderScene(this, employee);  
 break;  
 case "Manager": {  
 List<Order> orders = (List<Order>) objectInputStream.readObject();  
 List<Task> tasks = (List<Task>) objectInputStream.readObject();  
 List<Employee> employees = (List<Employee>) objectInputStream.readObject();  
 List<Equipment> equipmentList = (List<Equipment>) objectInputStream.readObject();  
 sceneManager.showManagerScene(this, employee, orders, tasks,employees,equipmentList);  
 break;  
 }  
 }  
 } else {  
 sceneManager.showErrorScene(this);  
 }  
  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void logOut() {  
  
 try {  
 objectOutputStream.writeObject("Close");  
 // Zamknij gniazdo i strumienie wejścia/wyjścia  
 if (socket != null && !socket.isClosed()) {  
 socket.close();  
 }  
 if (in != null) {  
 in.close();  
 }  
 if (out != null) {  
 out.close();  
 }  
 if (objectInputStream != null) {  
 objectInputStream.close();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 sceneManager.showLoginScene(this);  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 public List<String> getRoles() {  
 try {  
 return (List<String>) objectInputStream.readObject();  
 }catch (IOException | ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return null;  
 }  
 public void updateEmployee(Employee employee) {  
  
 try {  
 objectOutputStream.writeObject("Update");  
 objectOutputStream.writeObject(employee);  
 }catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
  
 public void showLogin() {  
 sceneManager.showLoginScene(this);  
 }  
 public List<String> getZones() {  
 try {  
 return (List<String>) objectInputStream.readObject();  
 }catch (IOException | ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 public void cancel() {  
 try {  
  
  
 objectOutputStream.writeObject("Cancel");  
 }catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
  
 public List<Object> getOrderInfo(Order order) {  
 List<Object> objects=new ArrayList<>();  
 try {  
 objectOutputStream.writeObject("orders");  
 objectOutputStream.writeObject(order);  
 objects = (List<Object>) objectInputStream.readObject();  
  
 }catch (IOException | ClassNotFoundException e){  
 e.printStackTrace();  
 }  
  
 return objects ;  
 }  
  
 public void addTask(Task task) {  
 try {  
 objectOutputStream.writeObject("addTask");  
 objectOutputStream.writeObject(task);  
 }catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
  
 public List<List<String>> getUseEquipment(int id) {  
 try {  
 objectOutputStream.writeObject("getUseEquipment");  
 objectOutputStream.writeObject(id);  
 List<List<String>> useEquipment = (List<List<String>>) objectInputStream.readObject();  
 return useEquipment;  
 }catch (IOException | ClassNotFoundException e){  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 public void updateEquipment(Equipment equipment) {  
 try{  
 objectOutputStream.writeObject("updateEquipment");  
 objectOutputStream.writeObject(equipment);  
 }catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
}

TCPClientFX to klasa, która pełni rolę klienta w architekturze klient-serwer w systemie opartym na technologii JavaFX. Aplikacja kliencka ta jest odpowiedzialna za nawiązywanie połączenia z serwerem, wymianę danych oraz zarządzanie interfejsem użytkownika.

TCPClientFX jest kluczowym elementem interfejsu użytkownika w systemie zarządzania zasobami, umożliwiającym użytkownikowi komunikację z serwerem i interakcję z systemem w zależności od przypisanej mu roli.

### Klasa AdminController

package client.controller;  
  
import client.TCPClientFX;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.collections.FXCollections;  
import javafx.fxml.FXML;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
import server.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class AdminController {  
 private TCPClientFX tcpClientFX;  
 private Employee employee;  
 private List<Employee> employees;  
 private List<Task> tasks;  
 private List<Order> orders;  
 private List<Equipment> equipment;  
 private List<Component> components;  
 List<String> roles = new ArrayList<>();  
 List<String> zones = new ArrayList<>();  
  
 @FXML  
 private Label name;  
 @FXML  
 private Label lastName;  
 @FXML  
 private Label role;  
 @FXML  
 private Label zone;  
 @FXML  
 private TabPane tabPane;  
  
 public AdminController(TCPClientFX tcpClientFX, Employee employee, List<Employee> employees, List<Task> tasks, List<Order> orders, List<Equipment> equipment, List<Component> components) {  
 this.tcpClientFX = tcpClientFX;  
 this.employee = employee;  
 this.employees = employees;  
 this.tasks = tasks;  
 this.orders = orders;  
 this.equipment = equipment;  
 this.components = components;  
 }  
  
 public AdminController(TCPClientFX tcpClientFX, Employee employee, List<Employee> employees) {  
 this.tcpClientFX = tcpClientFX;  
 this.employee = employee;  
 this.employees = employees;  
  
 }  
  
 @FXML  
 private void initialize() {  
 name.setText("Name: " + employee.getName());  
 lastName.setText("Last name: " + employee.getLastName());  
 role.setText("Role: " + employee.getRole());  
 zone.setText("Zone: " + employee.getZone());  
 roles = tcpClientFX.getRoles();  
 zones = tcpClientFX.getZones();  
 // Dodajemy zakładki i ich zawartość  
 addEmployeesTab();  
   
 }  
  
 private void addEmployeesTab() {  
 Tab employeesTab = new Tab("Employees");  
  
 // Tworzymy TableView dla pracowników  
 TableView<Employee> employeesTable = new TableView<>();  
 employeesTab.setContent(employeesTable);  
  
 // Tworzymy kolumny tabeli  
 TableColumn<Employee, Integer> idColumn = new TableColumn<>("ID");  
 TableColumn<Employee, String> nameColumn = new TableColumn<>("Name");  
 TableColumn<Employee, String> lastNameColumn = new TableColumn<>("Last Name");  
 TableColumn<Employee, String> roleColumn = new TableColumn<>("Role");  
 TableColumn<Employee, String> zoneColumn = new TableColumn<>("Zone");  
 TableColumn<Employee, Void> editColumn = new TableColumn<>("Edit");  
  
 // Ustawiamy, jakie wartości mają być wyświetlane w kolumnach  
 idColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getId()).asObject());  
 nameColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getName()));  
 lastNameColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getLastName()));  
 roleColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getRole()));  
 zoneColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getZone()));  
  
 // Ustawiamy przyciski w kolumnie "Edit"  
 editColumn.setCellFactory(col -> {  
 TableCell<Employee, Void> cell = new TableCell<>() {  
 private final Button editButton = new Button("Edit");  
  
 {  
 editButton.setOnAction(event -> {  
 Employee employee = getTableView().getItems().get(getIndex());  
 int employeeId = employee.getId();  
  
 // Tworzymy nowe okno  
 Stage editStage = new Stage();  
 editStage.setTitle("Edit Employee");  
  
 // Tworzymy VBox, aby umieścić ComboBoxy, etykiety z danymi pracownika oraz przyciski  
 VBox vbox = new VBox();  
 vbox.setSpacing(10);  
  
 // ComboBox dla roli  
 ComboBox<String> roleComboBox = new ComboBox<>(FXCollections.*observableArrayList*(roles));  
 roleComboBox.setValue(employee.getRole()); // Ustawiamy początkową wartość na rolę pracownika  
  
 // ComboBox dla strefy  
 ComboBox<String> zoneComboBox = new ComboBox<>(FXCollections.*observableArrayList*(zones));  
 zoneComboBox.setValue(employee.getZone()); // Ustawiamy początkową wartość na strefę pracownika  
  
 // Etykiety z danymi pracownika  
 vbox.getChildren().add(new Label("Name: " + employee.getName()));  
 vbox.getChildren().add(new Label("Last name: " + employee.getLastName()));  
  
 // Etykieta i ComboBox dla roli  
 HBox roleBox = new HBox(new Label("Role: "), roleComboBox);  
 vbox.getChildren().add(roleBox);  
  
 // Etykieta i ComboBox dla strefy  
 HBox zoneBox = new HBox(new Label("Zone: "), zoneComboBox);  
 vbox.getChildren().add(zoneBox);  
  
 // Przycisk "Apply"  
 Button applyButton = new Button("Apply");  
 applyButton.setOnAction(applyEvent -> {  
 // Pobierz wybrane wartości z ComboBoxów  
 String newRole = roleComboBox.getValue();  
 String newZone = zoneComboBox.getValue();  
  
 // Zaktualizuj dane pracownika w TableView  
 employee.setRole(newRole);  
 employee.setZone(newZone);  
 tcpClientFX.updateEmployee(employee);  
 TableView<Employee> tableView = getTableView();  
 tableView.refresh();  
  
 // Zamknij okno po zastosowaniu zmian  
 editStage.close();  
 });  
  
 // Przycisk "Cancel"  
 Button cancelButton = new Button("Cancel");  
 cancelButton.setOnAction(cancelEvent -> {  
 tcpClientFX.cancel();  
 editStage.close(); // Zamknij okno bez zapisywania zmian  
 });  
  
 // Dodajemy przyciski do VBox  
 vbox.getChildren().addAll(applyButton, cancelButton);  
  
 // Dodajemy VBox do sceny  
 Scene editScene = new Scene(vbox, 300, 200);  
  
 // Ustawiamy scenę w nowym oknie  
 editStage.setScene(editScene);  
  
 // Pokazujemy nowe okno  
 editStage.show();  
  
 roles.forEach(System.*out*::println);  
 zones.forEach(System.*out*::println);  
 System.*out*.println("Edit button clicked for employee with ID: " + employeeId);  
 });  
  
  
 }  
  
 @Override  
 protected void updateItem(Void item, boolean empty) {  
 super.updateItem(item, empty);  
 setGraphic(empty ? null : editButton);  
 }  
 };  
 return cell;  
 });  
  
 // Dodajemy kolumny do tabeli  
 employeesTable.getColumns().addAll(idColumn, nameColumn, lastNameColumn, roleColumn, zoneColumn, editColumn);  
  
 // Dodajemy dane do tabeli  
 employeesTable.getItems().addAll(employees);  
  
 // Dodajemy zakładkę do TabPane  
 tabPane.getTabs().add(employeesTab);  
 }  
  
  
 @FXML  
 private void adminButtonAction() {  
 tcpClientFX.logOut();  
 }  
}

AdminController to kontroler obsługujący interfejs graficzny dla administratora w systemie zarządzania zasobami. Kontroler ten działa w środowisku JavaFX i współpracuje z obiektem TCPClientFX do komunikacji z serwerem.

AdminController zapewnia intuicyjny interfejs graficzny dla administratora, umożliwiając mu zarządzanie danymi pracowników i korzystanie z funkcji systemu zarządzania zasobami.

### Klasa EmployeeController

package client.controller;  
  
import client.TCPClientFX;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import javafx.scene.control.Tab;  
import javafx.scene.control.TabPane;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.PropertyValueFactory;  
import server.Employee;  
import server.Task;  
  
import java.util.List;  
  
public class EmployeeController {  
  
 private TCPClientFX tcpClientFX;  
 private Employee employee;  
 private List<Task> tasks;  
  
 @FXML  
 private Label name;  
 @FXML  
 private Label lastName;  
 @FXML  
 private Label role;  
 @FXML  
 private Label zone;  
  
 @FXML  
 private TabPane tabPane;  
  
 public EmployeeController(TCPClientFX tcpClientFX, Employee employee, List<Task> tasks) {  
 this.tcpClientFX = tcpClientFX;  
 this.employee = employee;  
 this.tasks = tasks;  
 }  
  
 @FXML  
 private void initialize() {  
 name.setText("Name: " + employee.getName());  
 lastName.setText("Last name: " + employee.getLastName());  
 role.setText("Role: " + employee.getRole());  
 zone.setText("Zone: " + employee.getZone());  
  
  
 }  
 @FXML  
 private void employeeButtonAction() {  
 tcpClientFX.logOut();  
 }  
}

EmployeeController jest kontrolerem interfejsu graficznego przeznaczonym dla pracownika. Skupia się na prezentacji informacji o pracowniku oraz jego przypisanych zadaniach. W rezultacie EmployeeController zapewnia intuicyjny interfejs graficzny dla pracownika, wyświetlając podstawowe informacje o nim i umożliwiając łatwe wylogowanie się z systemu.

### Klasa ErrorController

package client.controller;  
import client.SceneManager;  
import client.TCPClientFX;  
import javafx.fxml.FXML;  
  
public class ErrorController {  
 private TCPClientFX tcpClientFX;  
  
 public ErrorController(TCPClientFX tcpClientFX) {  
 this.tcpClientFX = tcpClientFX;  
 }  
@FXML  
 private void errorButtonAction(){  
 tcpClientFX.showLogin();  
 }  
}

ErrorController obsługuje błąd logowania w interfejsie graficznym klienta, umożliwiając powrót do ekranu logowania po wystąpieniu problemu.

### Klasa LeaderController

package client.controller;  
  
import client.TCPClientFX;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import server.Employee;  
  
  
public class LeaderController {  
private TCPClientFX tcpClientFX;  
private Employee employee;  
 @FXML  
 private Label name;  
 @FXML  
 private Label lastName;  
 @FXML  
 private Label role;  
 @FXML  
 private Label zone;  
  
 public LeaderController(TCPClientFX tcpClientFX, Employee employee) {  
 this.tcpClientFX = tcpClientFX;  
 this.employee = employee;  
 }  
  
 @FXML  
 private void initialize() {  
 name.setText("Name: "+employee.getName());  
 lastName.setText("Last name: "+employee.getLastName());  
 role.setText("Role: "+employee.getRole());  
 zone.setText("Zone: "+employee.getZone());  
 }  
  
 @FXML  
 private void leaderButtonAction() {  
 tcpClientFX.logOut();  
 }  
}

LeaderController jest kontrolerem dla interfejsu lidera w systemie. Po pierwsze, inicjalizuje dane lidera, takie jak imię, nazwisko, rola i strefa, aby zostały wyświetlone na ekranie. Po drugie, obsługuje akcję przycisku, umożliwiając liderowi wylogowanie się z systemu.

### Klasa LoginController

package client.controller;  
  
import client.PasswordEncryptor;  
import client.TCPClientFX;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
  
public class LoginController {  
 private String username;  
 private String password;  
 @FXML  
 private TextField nameField;  
 @FXML  
 private PasswordField passField;  
 private Button loginButton;  
 private TCPClientFX tcpClientFX;  
  
 public LoginController(TCPClientFX tcpClientFX){  
 this.tcpClientFX = tcpClientFX;  
 }  
  
 @FXML  
 private void loginButtonAction(){  
 this.username = nameField.getText();  
 this.password = PasswordEncryptor.*encryptPassword*(passField.getText());  
 tcpClientFX.setLoginData(username,password);  
 }  
}

LoginController jest kontrolerem dla interfejsu logowania w systemie. Jego główne zadania to obsługa przycisku logowania.

### Klasa ManagerController

package client.controller;  
  
import client.TCPClientFX;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.fxml.FXML;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
import server.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
import java.util.StringJoiner;  
  
public class ManagerController {  
 private TCPClientFX tcpClientFX;  
 private Employee employee;  
 private List<Order> orders;  
 private List<Task> tasks;  
 private List<Employee> employees;  
private List<Equipment> equipmentList;  
 @FXML  
 private Label name;  
 @FXML  
 private Label lastName;  
 @FXML  
 private Label role;  
 @FXML  
 private Label zone;  
  
 @FXML  
 private TabPane tabPane;  
  
 public ManagerController(TCPClientFX tcpClientFX, Employee employee, List<Order> orders, List<Task> tasks, List<Employee> employees, List<Equipment> equipmentList) {  
 this.tcpClientFX = tcpClientFX;  
 this.employee = employee;  
 this.orders = orders;  
 this.tasks = tasks;  
 this.employees = employees;  
 this.equipmentList=equipmentList;  
 }  
  
 @FXML  
 private void initialize() {  
 name.setText("Name: " + employee.getName());  
 lastName.setText("Last name: " + employee.getLastName());  
 role.setText("Role: " + employee.getRole());  
 zone.setText("Zone: " + employee.getZone());  
  
 // Dodajemy zakładki i ich zawartość  
 addOrdersTab();  
 addTasksTab(tasks);  
 addEmployeesTab(employees);  
 addEquipmentsTab(equipmentList);  
  
 }  
 private void addOrdersTab() {  
 Tab ordersTab = new Tab("Orders");  
  
 // Tworzymy TableView dla zamówień  
 TableView<Order> ordersTable = new TableView<>();  
 ordersTab.setContent(ordersTable);  
  
 // Tworzymy kolumny tabeli  
 TableColumn<Order, Integer> idColumn = new TableColumn<>("ID");  
 TableColumn<Order, String> productColumn = new TableColumn<>("Product");  
 TableColumn<Order, Integer> quantityOrderedColumn = new TableColumn<>("Quantity Ordered");  
 TableColumn<Order, Integer> quantityInProductionColumn = new TableColumn<>("Quantity In Production");  
 TableColumn<Order, Integer> quantityFinishedColumn = new TableColumn<>("Quantity Finished");  
 TableColumn<Order, String> statusColumn = new TableColumn<>("Status");  
 TableColumn<Order, Void> startColumn = new TableColumn<>("Start");  
  
 // Ustawiamy, jakie wartości mają być wyświetlane w kolumnach  
 idColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getId()).asObject());  
 productColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getProduct().getName()));  
 quantityOrderedColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getProduct().getQuantityOrdered()).asObject());  
 quantityInProductionColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getProduct().getQuantityInProduction()).asObject());  
 quantityFinishedColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getProduct().getQuantityFinished()).asObject());  
 statusColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getStatus()));  
  
 // Ustawiamy przyciski w kolumnie "Start"  
 startColumn.setCellFactory(col -> {  
 TableCell<Order, Void> cell = new TableCell<>() {  
 private final Button startButton = new Button("Start");  
  
 {  
 startButton.setOnAction(event -> {  
 Order order = getTableView().getItems().get(getIndex());  
 int orderId = order.getId();  
 // Tutaj dodaj logikę do obsługi przycisku Start dla danego zamówienia (orderId)  
 List<Object> objects = tcpClientFX.getOrderInfo(order);  
 List<Equipment> equipment = (List<Equipment>) objects.get(0);  
 List<Component> components = (List<Component>) objects.get(1);  
 // Otwórz nowe okno "New Task"  
 openNewTaskWindow(order,equipment,components, ordersTable);  
  
 });  
 }  
  
 @Override  
 protected void updateItem(Void item, boolean empty) {  
 super.updateItem(item, empty);  
  
 if (empty || getTableRow().getItem() == null) {  
 setGraphic(null);  
 } else {  
 Order order = (Order) getTableRow().getItem();  
 String status = order.getStatus();  
  
 // Wyświetl przycisk tylko dla zamówień z statusami "accepted" lub "inprogress"  
 if ("accepted".equalsIgnoreCase(status) || "progress".equalsIgnoreCase(status)) {  
 setGraphic(startButton);  
 } else {  
 setGraphic(null);  
 }  
 }  
 }  
 };  
 return cell;  
 });  
  
 // Dodajemy kolumny do tabeli  
 ordersTable.getColumns().addAll(idColumn, productColumn, quantityOrderedColumn, quantityInProductionColumn, quantityFinishedColumn, statusColumn, startColumn);  
  
 // Dodajemy wszystkie zamówienia do tabeli  
 ordersTable.getItems().addAll(orders);  
  
 // Dodajemy zakładkę do TabPane  
 tabPane.getTabs().add(ordersTab);  
 }  
  
 private void openNewTaskWindow(Order selectedOrder, List<Equipment> equipmentList, List<Component> components, TableView<Order> ordersTable) {  
 Stage newTaskStage = new Stage();  
 newTaskStage.setTitle("New Task");  
  
 // Tworzymy elementy interfejsu użytkownika  
 Label nameLabel = new Label("Name:");  
 TextField nameTextField = new TextField();  
  
 Label priorityLabel = new Label("Priority:");  
 ComboBox<String> priorityComboBox = new ComboBox<>();  
 priorityComboBox.getItems().addAll("High", "Normal", "Low");  
  
 Label descriptionLabel = new Label("Description:");  
 TextArea descriptionTextArea = new TextArea();  
  
 Label normLabel = new Label("Norm:");  
 TextField normTextField = new TextField();  
  
 Label componentLabel = new Label("Components:");  
 VBox componentsVBox = new VBox();  
  
 List<CheckBox> componentCheckboxes = new ArrayList<>();  
 components.forEach(component -> componentCheckboxes.add(new CheckBox(component.getName())));  
 componentsVBox.getChildren().addAll(componentCheckboxes);  
  
 Label equipmentLabel = new Label("Equipment:");  
 ComboBox<String> equipmentComboBox = new ComboBox<>();  
 equipmentList.forEach(equipment -> equipmentComboBox.getItems().add(equipment.getName()));  
  
  
 // Dodajemy pole Spinner dla ilości (Quantity)  
 Label quantityLabel = new Label("Quantity:");  
 Spinner<Integer> quantitySpinner = new Spinner<>(0, calculateMaxQuantity(selectedOrder), 0, 1);  
  
 // Tworzymy przyciski "Apply" i "Cancel"  
 Button applyButton = new Button("Apply");  
 Button cancelButton = new Button("Cancel");  
  
 // Ustawiamy akcję dla przycisku "Apply"  
 applyButton.setOnAction(event -> {  
 // Tutaj dodaj logikę do zastosowania wprowadzonych danych  
 List<Component> selectedComponents = new ArrayList<>();  
  
 for (CheckBox checkBox : componentCheckboxes) {  
 if (checkBox.isSelected()) {  
 String componentName = checkBox.getText();  
  
 // Szukaj obiektu Component po nazwie w liście components  
 Optional<Component> foundComponent = components.stream()  
 .filter(component -> component.getName().equals(componentName))  
 .findFirst();  
  
 foundComponent.ifPresent(selectedComponents::add);  
 }  
 }  
  
 // Pobierz nazwę wybranego sprzętu  
 String selectedEquipmentName = equipmentComboBox.getValue();  
  
 // Szukaj obiektu Equipment o danej nazwie w equipmentList  
 Equipment selectedEquipment = equipmentList.stream()  
 .filter(equipment -> equipment.getName().equals(selectedEquipmentName))  
 .findFirst()  
 .orElse(null);  
  
 assert selectedEquipment != null;  
 Task task = new Task(1, nameTextField.getText(), priorityComboBox.getValue(), descriptionTextArea.getText(), Integer.*parseInt*(normTextField.getText()), selectedComponents, selectedEquipment, selectedEquipment.getZone(), quantitySpinner.getValue(),selectedOrder.getProduct().getId(), selectedOrder.getId());  
 tcpClientFX.addTask(task);  
 selectedOrder.getProduct().setQuantityInProduction(selectedOrder.getProduct().getQuantityInProduction()+quantitySpinner.getValue());  
 selectedOrder.setStatus("progress");  
 // Aktualizuj dane w tabeli  
 ordersTable.getItems().clear(); // Wyczyść aktualne dane  
 ordersTable.getItems().addAll(orders); // Dodaj nowe dane  
  
 newTaskStage.close();  
 });  
  
  
 // Ustawiamy akcję dla przycisku "Cancel"  
 cancelButton.setOnAction(event -> {  
 // Tutaj dodaj logikę do anulowania wprowadzonych zmian  
 newTaskStage.close();  
 });  
  
 // Ustawiamy layout za pomocą GridPane  
 GridPane gridPane = new GridPane();  
 gridPane.setVgap(10);  
 gridPane.setHgap(10);  
 gridPane.setAlignment(Pos.*CENTER*);  
  
 // Dodajemy etykiety i pola do wprowadzania danych do GridPane  
 gridPane.add(nameLabel, 0, 0);  
 gridPane.add(nameTextField, 1, 0);  
  
 gridPane.add(priorityLabel, 0, 1);  
 gridPane.add(priorityComboBox, 1, 1);  
  
 gridPane.add(descriptionLabel, 0, 2);  
 gridPane.add(descriptionTextArea, 1, 2);  
  
 gridPane.add(normLabel, 0, 3);  
 gridPane.add(normTextField, 1, 3);  
  
 gridPane.add(componentLabel, 0, 4);  
 gridPane.add(componentsVBox, 1, 4);  
  
 gridPane.add(equipmentLabel, 0, 5);  
 gridPane.add(equipmentComboBox, 1, 5);  
  
  
 gridPane.add(quantityLabel, 0, 7);  
 gridPane.add(quantitySpinner, 1, 7);  
  
 gridPane.add(new HBox(10, applyButton, cancelButton), 1, 8);  
  
 // Ustawiamy scenę  
 Scene scene = new Scene(gridPane, 400, 600); // Zwiększam wysokość okna  
 newTaskStage.setScene(scene);  
  
 // Pokazujemy nowe okno  
 newTaskStage.show();  
 }  
  
 private void addTasksTab(List<Task> tasks) {  
 Tab tasksTab = new Tab("Tasks");  
  
 // Tworzymy TableView dla zadań  
 TableView<Task> tasksTable = new TableView<>();  
 tasksTab.setContent(tasksTable);  
  
 // Tworzymy kolumny tabeli  
 TableColumn<Task, Integer> idColumn = new TableColumn<>("ID");  
 TableColumn<Task, String> nameColumn = new TableColumn<>("Name");  
 TableColumn<Task, String> priorityColumn = new TableColumn<>("Priority");  
 TableColumn<Task, String> descriptionColumn = new TableColumn<>("Description");  
 TableColumn<Task, Integer> normColumn = new TableColumn<>("Norm");  
 TableColumn<Task, String> zoneColumn = new TableColumn<>("Zone");  
 TableColumn<Task, Integer> quantityColumn = new TableColumn<>("Quantity");  
 TableColumn<Task, Void> viewColumn = new TableColumn<>("View");  
  
 // Ustawiamy, jakie wartości mają być wyświetlane w kolumnach  
 idColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getId()).asObject());  
 nameColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getName()));  
 priorityColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getPriority()));  
 descriptionColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getDescription()));  
 normColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getNorm()).asObject());  
 zoneColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getZone()));  
 quantityColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getQuantity()).asObject());  
  
 // Ustawiamy przyciski w kolumnie "View"  
 viewColumn.setCellFactory(col -> {  
 TableCell<Task, Void> cell = new TableCell<>() {  
 private final Button viewButton = new Button("View");  
  
 {  
 viewButton.setOnAction(event -> {  
 Task task = getTableView().getItems().get(getIndex());  
 int taskId = task.getId();  
 // Tutaj dodaj logikę do obsługi przycisku View dla danego zadania (taskId)  
 System.*out*.println("View button clicked for task with ID: " + taskId);  
  
 // Otwórz nowe okno "View Task"  
 //openViewTaskWindow(task);  
 });  
 }  
  
 @Override  
 protected void updateItem(Void item, boolean empty) {  
 super.updateItem(item, empty);  
  
 if (empty || getTableRow().getItem() == null) {  
 setGraphic(null);  
 } else {  
 setGraphic(viewButton);  
 }  
 }  
 };  
 return cell;  
 });  
  
 // Dodajemy kolumny do tabeli  
 tasksTable.getColumns().addAll(idColumn, nameColumn, priorityColumn, descriptionColumn, normColumn, zoneColumn, quantityColumn, viewColumn);  
  
 // Dodajemy wszystkie zadania do tabeli  
 tasksTable.getItems().addAll(tasks);  
  
 // Dodajemy zakładkę do TabPane  
 tabPane.getTabs().add(tasksTab);  
 }  
  
  
  
 private int calculateMaxQuantity(Order selectedOrder) {  
 if (selectedOrder != null) {  
 int maxQuantity = selectedOrder.getProduct().getQuantityOrdered()  
 - selectedOrder.getProduct().getQuantityInProduction()  
 - selectedOrder.getProduct().getQuantityFinished();  
  
 return Math.*max*(maxQuantity, 0); // Nie pozwalamy na ujemne wartości  
 }  
 return 0;  
 }  
  
 private void addEmployeesTab(List<Employee> employees) {  
 Tab employeesTab = new Tab("Employees");  
  
 // Tworzymy TableView dla pracowników  
 TableView<Employee> employeesTable = new TableView<>();  
 employeesTab.setContent(employeesTable);  
  
 // Tworzymy kolumny tabeli  
 TableColumn<Employee, Integer> idColumn = new TableColumn<>("ID");  
 TableColumn<Employee, String> nameColumn = new TableColumn<>("Name");  
 TableColumn<Employee, String> lastNameColumn = new TableColumn<>("Last Name");  
 TableColumn<Employee, String> roleColumn = new TableColumn<>("Role");  
 TableColumn<Employee, String> zoneColumn = new TableColumn<>("Zone");  
 TableColumn<Employee, Void> viewColumn = new TableColumn<>("View");  
  
 // Ustawiamy, jakie wartości mają być wyświetlane w kolumnach  
 idColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getId()).asObject());  
 nameColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getName()));  
 lastNameColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getLastName()));  
 roleColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getRole()));  
 zoneColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getZone()));  
  
 // Ustawiamy przyciski w kolumnie "View"  
 viewColumn.setCellFactory(col -> {  
 TableCell<Employee, Void> cell = new TableCell<>() {  
 private final Button viewButton = new Button("View");  
  
 {  
 viewButton.setOnAction(event -> {  
 Employee employee = getTableView().getItems().get(getIndex());  
 int employeeId = employee.getId();  
 // Tutaj dodaj logikę do obsługi przycisku View dla danego pracownika (employeeId)  
 System.*out*.println("View button clicked for employee with ID: " + employeeId);  
  
 // Otwórz nowe okno "View Employee"  
 // openViewEmployeeWindow(employee);  
 });  
 }  
  
 @Override  
 protected void updateItem(Void item, boolean empty) {  
 super.updateItem(item, empty);  
  
 if (empty || getTableRow().getItem() == null) {  
 setGraphic(null);  
 } else {  
 setGraphic(viewButton);  
 }  
 }  
 };  
 return cell;  
 });  
  
 // Dodajemy kolumny do tabeli  
 employeesTable.getColumns().addAll(idColumn, nameColumn, lastNameColumn, roleColumn, zoneColumn, viewColumn);  
  
 // Dodajemy wszystkich pracowników do tabeli  
 employeesTable.getItems().addAll(employees);  
  
 // Dodajemy zakładkę do TabPane  
 tabPane.getTabs().add(employeesTab);  
 }  
 private void addEquipmentsTab(List<Equipment> equipments) {  
 Tab equipmentsTab = new Tab("Equipments");  
  
 // Tworzymy TableView dla sprzętu  
 TableView<Equipment> equipmentsTable = new TableView<>();  
 equipmentsTab.setContent(equipmentsTable);  
  
 // Tworzymy kolumny tabeli  
 TableColumn<Equipment, Integer> idColumn = new TableColumn<>("ID");  
 TableColumn<Equipment, String> nameColumn = new TableColumn<>("Name");  
 TableColumn<Equipment, String> statusColumn = new TableColumn<>("Status");  
 TableColumn<Equipment, String> zoneColumn = new TableColumn<>("Zone");  
 TableColumn<Equipment, Void> viewColumn = new TableColumn<>("View");  
  
 // Ustawiamy, jakie wartości mają być wyświetlane w kolumnach  
 idColumn.setCellValueFactory(cellData -> new SimpleIntegerProperty(cellData.getValue().getId()).asObject());  
 nameColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getName()));  
 statusColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getStatus()));  
 zoneColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getZone()));  
  
 // Ustawiamy przyciski w kolumnie "View"  
 viewColumn.setCellFactory(col -> {  
 TableCell<Equipment, Void> cell = new TableCell<>() {  
 private final Button viewButton = new Button("View");  
  
 {  
 viewButton.setOnAction(event -> {  
 Equipment equipment = getTableView().getItems().get(getIndex());  
 List<List<String>> equipmentUse = tcpClientFX.getUseEquipment(equipment.getId());  
 openViewEquipmentWindow(equipment,equipmentUse);  
 });  
 }  
  
 @Override  
 protected void updateItem(Void item, boolean empty) {  
 super.updateItem(item, empty);  
  
 if (empty || getTableRow().getItem() == null) {  
 setGraphic(null);  
 } else {  
 setGraphic(viewButton);  
 }  
 }  
 };  
 return cell;  
 });  
  
 // Dodajemy kolumny do tabeli  
 equipmentsTable.getColumns().addAll(idColumn, nameColumn, statusColumn, zoneColumn, viewColumn);  
  
 // Dodajemy wszystkie sprzęty do tabeli  
 equipmentsTable.getItems().addAll(equipments);  
  
 // Dodajemy zakładkę do TabPane  
 tabPane.getTabs().add(equipmentsTab);  
 }  
 private void openViewEquipmentWindow(Equipment equipment, List<List<String>> equipmentUse) {  
 // Tworzymy nowe okno "View Equipment"  
 Stage viewEquipmentStage = new Stage();  
 viewEquipmentStage.setTitle("View Equipment");  
  
 // Tworzymy kontener VBox dla układu okna  
 VBox vbox = new VBox(10);  
  
 // Dodajemy etykiety i pola tekstowe do VBox  
 Label nameLabel = new Label("Name: " + equipment.getName());  
  
 // Lista rozwijana (ComboBox) dla statusu  
 Label statusLable = new Label("Status: ");  
 ComboBox<String> statusComboBox = new ComboBox<>();  
 statusComboBox.getItems().addAll("available", "in use", "out of use");  
 statusComboBox.setValue(equipment.getStatus()); // Ustawienie domyślnego statusu  
  
 Label zoneLabel = new Label("Zone: " + equipment.getZone());  
  
 // Dodajemy elementy do VBox  
 vbox.getChildren().addAll(nameLabel, statusLable, statusComboBox, zoneLabel);  
  
 // Dodaj informacje z equipmentUse  
 for (List<String> innerList : equipmentUse) {  
 StringJoiner stringJoiner = new StringJoiner(" ");  
 for (String value : innerList) {  
 stringJoiner.add(value);  
  
 }  
 Label label = new Label(stringJoiner.toString());  
 vbox.getChildren().add(label);  
  
 }  
  
 // Tworzymy HBox dla przycisków Apply i Cancel  
 HBox buttonBox = new HBox(10);  
 Button applyButton = new Button("Apply");  
 Button cancelButton = new Button("Cancel");  
  
 // Obsługa przycisku Apply  
 applyButton.setOnAction(event -> {  
 equipment.setStatus(statusComboBox.getValue());  
 tcpClientFX.updateEquipment(equipment);  
  
 // Zamknij okno po zastosowaniu zmian  
 viewEquipmentStage.close();  
 });  
  
 // Obsługa przycisku Cancel  
 cancelButton.setOnAction(event -> {  
 // Zamknij okno bez zastosowywania zmian  
 viewEquipmentStage.close();  
 });  
  
 // Dodaj przyciski do HBox  
 buttonBox.getChildren().addAll(applyButton, cancelButton);  
  
 // Dodaj HBox do VBox  
 vbox.getChildren().add(buttonBox);  
  
 // Ustawiamy VBox jako scenę  
 Scene scene = new Scene(vbox, 400, 300); // Zwiększyłem szerokość okna, aby pomieścić więcej informacji  
 viewEquipmentStage.setScene(scene);  
  
 // Pokazujemy nowe okno  
 viewEquipmentStage.show();  
 }  
  
 @FXML  
 private void managerButtonAction() {  
 tcpClientFX.logOut();  
 }  
}

Kod w klasie ManagerController odpowiada za obsługę interfejsu graficznego dla managera w systemie. W kodzie tym można wyróżnić:

1. Inicjalizacja Interfejsu

2. Zakładka Zamówień: Tworzenie zakładki "Orders"

3. Zakładka Zadań: Tworzenie zakładki "Tasks"

4. Zakładka Pracowników: Tworzenie zakładki "Employees"

5. Zakładka Sprzętu: Tworzenie zakładki "Equipments"

6. Obsługa Nowego Zadania

7. Obsługa Nowego Sprzętu

W pakiecie tym znajduje się też plik z stylami css.

. root{  
 -fx-min-width: 800px;  
 -fx-min-height: 600px;  
 -fx-background-color: #fffaaa;  
}  
  
/\* Styl dla GridPane \*/  
#gridPane {  
 -fx-min-width: 800;  
 -fx-min-height: 600;  
 -fx-padding: 0 80 5 80;  
  
}  
  
  
/\* Styl dla Label i TextField \*/  
.label, .text-field {  
 -fx-padding: 5;  
}  
  
/\* Styl dla Button \*/  
#loginButton {  
 -fx-padding: 5 0 5 0;  
 -fx-background-color: #3f919a;  
 -fx-font-size: 12;  
 -fx-font-family: Arial;  
 -fx-background-radius: 12;  
  
  
  
}  
  
/\* Styl dla kolumny z logo \*/  
#logoColumn {  
 -fx-pref-width: 400; /\* Szerokość kolumny z logo \*/  
}  
  
/\* Styl dla kolumny z polami logowania \*/  
#loginColumn {  
 -fx-alignment: CENTER;  
  
 -fx-pref-width: 400; /\* Szerokość kolumny z polami logowania \*/  
}

Plik do odpowiedzialny za wygląd aplikacji. Jest to wstęp do stworzenia interfejsu graficznego według pierwotnych założeń. Plik ten może być dalej aktualizowany dostosowując grafikę aplikacji.

### Plik AdminScene.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.geometry.Insets?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.Tab?>  
<?import javafx.scene.control.TabPane?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
  
<GridPane alignment="center" hgap="5" maxHeight="600.0" maxWidth="600.0" minHeight="400.0" minWidth="400.0" vgap="5" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1">  
 <padding>  
 <Insets bottom="10" left="10" right="10" top="10" />  
 </padding>  
  
 <Button onAction="#adminButtonAction" text="Wyloguj" GridPane.columnIndex="4" />  
  
 <!-- Dodane Label'e -->  
 <Label fx:id="name" />  
 <Label fx:id="lastName" GridPane.columnIndex="1" />  
 <Label fx:id="role" GridPane.columnIndex="2" />  
 <Label fx:id="zone" GridPane.columnIndex="3" />  
  
 <TabPane fx:id="tabPane" GridPane.columnIndex="0" GridPane.columnSpan="4" GridPane.rowIndex="1">  
 <tabs>  
  
 </tabs>  
 </TabPane>  
  
 <columnConstraints>  
 <ColumnConstraints maxWidth="108.5" minWidth="13.5" prefWidth="78.5" />  
 <ColumnConstraints maxWidth="314.0" minWidth="0.0" prefWidth="117.0" />  
 <ColumnConstraints maxWidth="342.0" minWidth="0.0" prefWidth="163.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="415.0" minWidth="56.0" prefWidth="138.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="366.0" minWidth="56.0" prefWidth="75.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints maxHeight="191.0" minHeight="10.0" prefHeight="33.0" vgrow="ALWAYS" />  
 <RowConstraints maxHeight="327.0" minHeight="169.0" prefHeight="327.0" vgrow="ALWAYS" />  
 <RowConstraints />  
 <RowConstraints />  
 <RowConstraints />  
 </rowConstraints>  
  
</GridPane>

### Plik EmployeeScene.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.geometry.Insets?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
  
<GridPane hgap="5" maxHeight="600.0" maxWidth="600.0" minHeight="400.0" minWidth="400.0" vgap="5" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1">  
  
 <padding>  
 <Insets bottom="10" left="10" right="10" top="10" />  
 </padding>  
  
 <Button onAction="#employeeButtonAction" text="Wyloguj" GridPane.columnIndex="4" />  
  
 <!-- Dodane Label'e -->  
 <Label fx:id="name"/>  
 <Label fx:id="lastName" GridPane.columnIndex="1" />  
 <Label fx:id="role" GridPane.columnIndex="2" />  
 <Label fx:id="zone" GridPane.columnIndex="3" />  
  
 <columnConstraints>  
 <ColumnConstraints maxWidth="108.5" minWidth="13.5" prefWidth="78.5" />  
 <ColumnConstraints maxWidth="314.0" minWidth="0.0" prefWidth="117.0" />  
 <ColumnConstraints maxWidth="342.0" minWidth="0.0" prefWidth="163.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="415.0" minWidth="56.0" prefWidth="138.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="366.0" minWidth="56.0" prefWidth="75.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints maxHeight="191.0" minHeight="10.0" prefHeight="33.0" vgrow="ALWAYS" />  
 <RowConstraints maxHeight="327.0" minHeight="169.0" prefHeight="327.0" vgrow="ALWAYS" />  
 <RowConstraints />  
 <RowConstraints />  
 <RowConstraints />  
 </rowConstraints>  
  
</GridPane>

### Plik ErrorScene.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.text.Font?>  
  
<AnchorPane prefHeight="145.0" prefWidth="155.0" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1">  
 <children>  
 <Button layoutX="59.0" layoutY="118.0" mnemonicParsing="false" onAction="#errorButtonAction" text="Close" />  
 <Label layoutX="26.0" layoutY="73.0" prefHeight="44.0" prefWidth="141.0" text="Login error !!!" textFill="#e10000">  
 <font>  
 <Font size="24.0" />  
 </font>  
 </Label>  
 </children>  
</AnchorPane>

### Plik LeaderScene.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
  
<?import javafx.geometry.Insets?>  
<GridPane xmlns="http://javafx.com/javafx"  
 xmlns:fx="http://javafx.com/fxml"  
 maxHeight="600.0" maxWidth="600.0" minHeight="400.0" minWidth="400.0"  
 alignment="center"  
 hgap="5"  
 vgap="5">  
 <padding>  
 <Insets bottom="10" left="10" right="10" top="10" />  
 </padding>  
  
 <Button onAction="#leaderButtonAction" text="Wyloguj" GridPane.columnIndex="4" />  
  
 <!-- Dodane Label'e -->  
 <Label fx:id="name"/>  
 <Label fx:id="lastName" GridPane.columnIndex="1" />  
 <Label fx:id="role" GridPane.columnIndex="2" />  
 <Label fx:id="zone" GridPane.columnIndex="3" />  
  
 <columnConstraints>  
 <ColumnConstraints maxWidth="108.5" minWidth="13.5" prefWidth="78.5" />  
 <ColumnConstraints maxWidth="314.0" minWidth="0.0" prefWidth="117.0" />  
 <ColumnConstraints maxWidth="342.0" minWidth="0.0" prefWidth="163.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="415.0" minWidth="56.0" prefWidth="138.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="366.0" minWidth="56.0" prefWidth="75.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints maxHeight="191.0" minHeight="10.0" prefHeight="33.0" vgrow="ALWAYS" />  
 <RowConstraints maxHeight="327.0" minHeight="169.0" prefHeight="327.0" vgrow="ALWAYS" />  
 <RowConstraints />  
 <RowConstraints />  
 <RowConstraints />  
 </rowConstraints>  
</GridPane>

### Plik LoginScene.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.geometry.Insets?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.PasswordField?>  
<?import javafx.scene.control.TextField?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
<?import javafx.scene.image.Image?>  
<AnchorPane style="-fx-background-color: #fffaaa" xmlns="http://javafx.com/javafx"  
 xmlns:fx="http://javafx.com/fxml"  
 >  
<GridPane id="gridPane" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1" >  
  
 <padding>  
 <Insets bottom="10" left="10" right="10" top="10" />  
 </padding>  
  
 <columnConstraints>  
 <ColumnConstraints fx:id="logoColumn"/>  
 <ColumnConstraints fx:id="loginColumn"/>  
 </columnConstraints>  
  
  
 <ImageView id="logoImage" GridPane.columnIndex="0" GridPane.rowIndex="0" GridPane.columnSpan="6" >  
 <image >  
 <Image url="@logo.png" />  
 </image>  
 </ImageView>  
  
 <Label id="loginLabel" text="Login:" GridPane.columnIndex="1" GridPane.rowIndex="1" minWidth="100" alignment="CENTER\_RIGHT"/>  
 <TextField fx:id="nameField" id="loginField" GridPane.columnIndex="2" GridPane.rowIndex="1" minWidth="100"/>  
  
 <Label id="passwordLabel" text="Password:" GridPane.columnIndex="3" GridPane.rowIndex="1" minWidth="100" alignment="CENTER\_RIGHT"/>  
 <PasswordField fx:id="passField" id="passwordField" GridPane.columnIndex="4" GridPane.rowIndex="1" minWidth="100"/>  
  
 <Button id="loginButton" onAction="#loginButtonAction" text="Login" GridPane.columnIndex="5" GridPane.rowIndex="1" minWidth="100" />  
  
</GridPane>  
</AnchorPane>

### Plik ManagerScene.fxml

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.geometry.Insets?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.control.Label?>  
<?import javafx.scene.control.Tab?>  
<?import javafx.scene.control.TabPane?>  
<?import javafx.scene.layout.ColumnConstraints?>  
<?import javafx.scene.layout.GridPane?>  
<?import javafx.scene.layout.RowConstraints?>  
  
<?import javafx.scene.layout.VBox?>  
<GridPane style="-fx-background-color: #fffaaa; -fx-min-width: 800;-fx-min-height: 600" alignment="center" hgap="5" maxHeight="600.0" maxWidth="600.0" minHeight="400.0" minWidth="400.0" vgap="5" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1">  
  
 <padding>  
 <Insets bottom="10" left="10" right="10" top="10" />  
 </padding>  
  
 <Button onAction="#managerButtonAction" text="Wyloguj" GridPane.columnIndex="4" />  
  
 <!-- Dodane Label'e -->  
 <Label fx:id="name" />  
 <Label fx:id="lastName" GridPane.columnIndex="1" />  
 <Label fx:id="role" GridPane.columnIndex="2" />  
 <Label fx:id="zone" GridPane.columnIndex="3" />  
  
 <TabPane fx:id="tabPane" GridPane.columnIndex="0" GridPane.columnSpan="5" GridPane.rowIndex="1">  
 <tabs>  
  
 </tabs>  
 </TabPane>  
  
 <columnConstraints>  
 <ColumnConstraints maxWidth="108.5" minWidth="13.5" prefWidth="78.5" />  
 <ColumnConstraints maxWidth="314.0" minWidth="0.0" prefWidth="117.0" />  
 <ColumnConstraints maxWidth="342.0" minWidth="0.0" prefWidth="163.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="415.0" minWidth="56.0" prefWidth="138.0" />  
 <ColumnConstraints hgrow="ALWAYS" maxWidth="366.0" minWidth="56.0" prefWidth="75.0" />  
 </columnConstraints>  
 <rowConstraints>  
 <RowConstraints maxHeight="191.0" minHeight="10.0" prefHeight="33.0" vgrow="ALWAYS" />  
 <RowConstraints maxHeight="327.0" minHeight="169.0" prefHeight="327.0" vgrow="ALWAYS" />  
 <RowConstraints />  
 <RowConstraints />  
 <RowConstraints />  
 </rowConstraints>  
  
</GridPane>

## Serwer

### Klasa Admin

package server;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Admin extends Employee{  
 public Admin(int id, String name, String lastName, String role, String zone, String login, String password) {  
 super(id, name, lastName, role, zone, login, password);  
 }  
  
 public Admin(int id, String name, String lastName, String role, String zone, String login, String password, Task task) {  
 super(id, name, lastName, role, zone, login, password, task);  
 }  
 public void addRole(Employee employee, String role){  
  
 }  
 public void changeRole(Employee updateEmployee){  
 MySQLDatabaseConnector mySQLDatabaseConnector = new MySQLDatabaseConnector();  
 mySQLDatabaseConnector.updateEmployee(updateEmployee.getId(), updateEmployee.getRole(), updateEmployee.getZone());  
 }  
  
  
  
}

W klasie tej zaimplementowano konstruktory dla admina, uwzględniające różne scenariusze przekazywania danych. Dodatkowo, w klasie znajdują się metody ***addRole*** oraz ***changeRole***, z których pierwsza ma na celu dodanie nowej roli dla pracownika, a druga aktualizację roli pracownika w bazie danych poprzez wykorzystanie klasy ***MySQLDatabaseConnector.***

### Klasa ClientHandler

package server;  
  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.List;  
  
public class ClientHandler implements Runnable {  
 private Socket clientSocket;  
 private Employee employee;  
  
 public ClientHandler(Socket socket) {  
 this.clientSocket = socket;  
 }  
  
 @Override  
 public void run() {  
 try {  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(clientSocket.getOutputStream());  
 ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());  
  
 String login = (String) objectInputStream.readObject();  
 String password = (String) objectInputStream.readObject();  
 Login log = new Login();  
 MySQLDatabaseConnector connector = new MySQLDatabaseConnector();  
  
 if (log.check(login, password)) {  
 employee = connector.getUserInfo(login, password);  
 log.startLogin(employee);  
 objectOutputStream.writeObject(employee);  
  
 if (employee instanceof Manager) {  
 List<Order> orders = employee.getListOfOrder();  
 List<Task> tasks = employee.getListOfTask();  
 List<Employee> employees = employee.getListOfEmployees();  
 List<Equipment> equipments = employee.getListOfEquipment();  
 objectOutputStream.writeObject(orders);  
 objectOutputStream.writeObject(tasks);  
 objectOutputStream.writeObject(employees);  
 objectOutputStream.writeObject(equipments);  
  
 String answer;  
 do {  
 answer = (String) objectInputStream.readObject();  
 if (answer.equals("orders")) {  
 Order order = (Order) objectInputStream.readObject();  
 String nameProduct = order.getProduct().getName();  
 List<Equipment> equipmentListOfTask = employee.getListOfEquipmentToTask(nameProduct);  
 List<Component> componentListOfTask = employee.getListOfComponentToTask(nameProduct);  
 List<Object> objects = new ArrayList<>();  
 objects.add(equipmentListOfTask);  
 objects.add(componentListOfTask);  
 objectOutputStream.writeObject(objects);  
 }  
 if (answer.equals("addTask")) {  
 Task task = (Task) objectInputStream.readObject();  
 ((Manager) employee).addNewTask(task, employee);  
 }  
 if (answer.equals("getUseEquipment")) {  
 int equipmentID = (int) objectInputStream.readObject();  
 List<List<String>> useEquipment = ((Manager) employee).getEquipmentTimeOfUseReport(equipmentID);  
 objectOutputStream.writeObject(useEquipment);  
 }  
 if (answer.equals("updateEquipment")) {  
 Equipment equipment = (Equipment) objectInputStream.readObject();  
 ((Manager) employee).changeEquipmentStatus(equipment);  
 }  
 if(answer.equals("Close")){  
 log.endLogin(employee);  
 }  
 } while (!answer.equals("Close"));  
 }  
  
 if (employee instanceof Admin) {  
 List<Employee> employees = ((Admin) employee).getListOfEmployees();  
 objectOutputStream.writeObject(employees);  
 objectOutputStream.writeObject(connector.getRolesList());  
 objectOutputStream.writeObject(connector.getZonesList());  
  
 String answer;  
 do {  
 answer = (String) objectInputStream.readObject();  
 if (answer.equals("Update")) {  
 Employee updateEmployee = (Employee) objectInputStream.readObject();  
 ((Admin) employee).changeRole(updateEmployee);  
 }  
 if(answer.equals("Close")){  
 log.endLogin(employee);  
 }  
 } while (!answer.equals("Close"));  
 }  
  
 if (employee instanceof ProductionEmployee) {  
 Task myTask = ((ProductionEmployee) employee).getMyTask(employee);  
 if (myTask == null) {  
 List<Task> tasksOfEmployee = ((ProductionEmployee) employee).getListOfTask(employee.getId());  
 objectOutputStream.writeObject(tasksOfEmployee);  
 } else {  
 List<Task> tasks = new ArrayList<>();  
 tasks.add(myTask);  
 objectOutputStream.writeObject(tasks);  
 }  
  
 }  
 } else {  
 objectOutputStream.writeObject(null);  
 }  
  
 clientSocket.close();  
 connector.closeConnection();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}

Klasa ClientHandler pełni rolę obsługi klienta po stronie serwera w systemie opartym na gniazdach w języku Java. Głównym zadaniem tej klasy jest przyjęcie połączenia od klienta, obsługa przesyłanych danych oraz koordynacja interakcji między klientem a serwerem

### Klasa Component

package server;  
  
public class Component extends Inventory{  
 public int quantity;  
 public Component(int id, String name, int quantity) {  
 super(id, name);  
 this.quantity = quantity;  
 }  
  
 @Override  
 public void reserve(Task task) {  
  
 }  
  
 @Override  
 public void cancelReservation(Task task) {  
  
 }  
 public void updateQuantity(int quantity){  
  
 }  
}

Klasa ta to „przepis” na to jak ma wyglądać każdy komponent. Znajdują się tu też metody przygotowane do wykorzystania w dalszym rozwoju aplikacji.

### Klasa Employee

package server;  
  
import java.io.Serializable;  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Employee implements Serializable {  
 private int id;  
 private String name;  
 private String lastName;  
 private String role;  
 private String zone;  
 private String login;  
 private String password;  
 private Task task;  
 private static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/system";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
 private Connection connection;  
 private String startTime;  
  
 public String getStartTime() {  
 return startTime;  
 }  
  
 public void setStartTime(String startTime) {  
 this.startTime = startTime;  
 }  
  
 public Employee(int id, String name, String lastName, String role, String zone, String login, String password) {  
 this.id = id;  
 this.name = name;  
 this.lastName = lastName;  
 this.role = role;  
 this.zone = zone;  
 this.login = login;  
 this.password = password;  
  
 }  
 public Employee(int id, String name, String lastName, String role, String zone) {  
 this.id = id;  
 this.name = name;  
 this.lastName = lastName;  
 this.role = role;  
 this.zone = zone;  
 }  
 public Employee(int id, String name, String lastName, String role, String zone, String login, String password, Task task) {  
 this.id = id;  
 this.name = name;  
 this.lastName = lastName;  
 this.role = role;  
 this.zone = zone;  
 this.login = login;  
 this.password = password;  
 this.task = task;  
 }  
 public void getTask(Task task){  
  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getRole() {  
 return role;  
 }  
  
 public String getZone() {  
 return zone;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void acceptTask(Task task){  
  
 }  
 public void endTask(Task task){  
  
 }  
 public void addTaskResult(Task task, Result result){  
  
 }  
 public List<Order> getListOfOrder(){  
 List<Order> orders = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT \n" +  
 " oq.orderQuantityID,\n" +  
 " p.productID,\n" +  
 " p.name AS productName,\n" +  
 " o.Status AS orderStatus,\n" +  
 " oq.quantityOrdered,\n" +  
 " oq.QuantityInProduction,\n" +  
 " oq.OuantityFinished\n" +  
 "FROM \n" +  
 " orderQuantity oq\n" +  
 "JOIN \n" +  
 " product p ON oq.productID = p.productID\n" +  
 "JOIN \n" +  
 " orders o ON oq.orderID = o.OrderID;";  
 try {  
 PreparedStatement preparedStatement= connection.prepareStatement(query);  
 ResultSet result = preparedStatement.executeQuery();  
 while (result.next()){  
 int id = result.getInt("orderQuantityID");  
 int productID = result.getInt("productID");  
 String productName = result.getString("productName");  
 String orderStatus = result.getString("orderStatus");  
 int quantityOrdered = result.getInt("quantityOrdered");  
 int quantityInProduction = result.getInt("QuantityInProduction");  
 int quantityFinished = result.getInt("OuantityFinished");  
 orders.add(new Order(id, orderStatus,new Product(productID,productName,quantityOrdered,quantityInProduction,quantityFinished)));  
 }  
  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 return orders;  
 }  
 public List<Employee> getListOfEmployees(){  
 List<Employee> employees = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT e.employeeID, e.name, e.lastName, r.roleName AS role, z.name AS zone\n" +  
 "FROM Employee e\n" +  
 "JOIN role r ON e.roleID = r.roleID\n" +  
 "JOIN zone z ON e.zoneID = z.zoneID;";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 ResultSet result = preparedStatement.executeQuery();  
 while (result.next()) {  
 int id = result.getInt("employeeID");  
 String name = result.getString("name");  
 String lastName = result.getString("lastName");  
 String role = result.getString("role");  
 String zone = result.getString("zone");  
 employees.add(new Employee(id, name,lastName,role,zone));  
 }  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 return employees;  
 }  
 public List<Task> getListOfTask(){  
 List<Task> tasks = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT \n" +  
 " t.taskID AS taskID,\n" +  
 " t.name AS name,\n" +  
 " t.priority,\n" +  
 " t.description,\n" +  
 " t.norm,\n" +  
 " z.name AS zone\_name,\n" +  
 " t.quantity\n" +  
 "FROM \n" +  
 " Task t\n" +  
 "JOIN \n" +  
 " Zone z ON t.zoneID = z.zoneID;\n";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 ResultSet result = preparedStatement.executeQuery();  
 while (result.next()) {  
 int id = result.getInt("taskID");  
 String name = result.getString("name");  
 String priority = result.getString("priority");  
 String description = result.getString("description");  
 int norm = result.getInt("norm");  
 String zoneName = result.getString("zone\_name");  
 int quantity = result.getInt("quantity");  
 tasks.add(new Task(id, name,priority,description,norm,zoneName,quantity));  
 }  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 return tasks;  
 }  
 public List<Equipment> getListOfEquipment(){  
 List<Equipment> equipments = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
  
 String query = "SELECT \n" +  
 " e.equipmentID AS equipmentID,\n" +  
 " e.name AS equipmentName,\n" +  
 " e.status,\n" +  
 " z.name AS zoneName\n" +  
 "FROM \n" +  
 " equipment e\n" +  
 "JOIN \n" +  
 " zone z ON e.zoneID = z.zoneID;";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 ResultSet result = preparedStatement.executeQuery();  
  
 while (result.next()) {  
 int id = result.getInt("equipmentID");  
 String name = result.getString("equipmentName");  
 String status = result.getString("status");  
 String zoneName = result.getString("zoneName");  
 equipments.add(new Equipment(id, name,status,zoneName));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 return equipments;  
 }  
 public void setRole(String role) {  
 this.role = role;  
 }  
  
 public void setZone(String zone) {  
 this.zone = zone;  
 }  
  
 public List<Equipment> getListOfEquipmentToTask(String nameProduct) {  
 List<Equipment> equipmentList = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query="SELECT equipment.equipmentID,equipment.name, equipmentCategory.name AS equipment\_category, equipment.status, zone.name AS equipment\_zone\n" +  
 "FROM taskCategory\n" +  
 "JOIN taskEquipmentCategory ON taskCategory.taskCategoryID = taskEquipmentCategory.taskCategoryId\n" +  
 "JOIN equipment ON taskEquipmentCategory.equipmentCategoryId = equipment.equipmentCategoryId\n" +  
 "JOIN equipmentCategory ON equipment.equipmentCategoryId = equipmentCategory.equipmentCategoryID\n" +  
 "JOIN zone ON equipment.zoneId = zone.zoneID\n" +  
 "WHERE taskCategory.name =?;";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1,nameProduct);  
 ResultSet result = preparedStatement.executeQuery();  
  
 while (result.next()) {  
 int id = result.getInt("equipmentID");  
 String nameComponent = result.getString("name");  
 String status = result.getString("status");  
 String zone = result.getString("equipment\_zone");  
 equipmentList.add(new Equipment(id, nameComponent, status, zone));  
  
  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return equipmentList;  
  
 }  
  
 public List<Component> getListOfComponentToTask(String nameProduct) {  
 List<Component> components = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query="SELECT component.componentID, Component.name, component.quantity \n" +  
 "FROM TaskCategory\n" +  
 "JOIN TaskCategoryComponent ON TaskCategory.taskCategoryID = TaskCategoryComponent.taskCategoryID\n" +  
 "JOIN Component ON Component.componentID = TaskCategoryComponent.componentID\n" +  
 "WHERE TaskCategory.name =?;";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1,nameProduct);  
 ResultSet result = preparedStatement.executeQuery();  
  
 while (result.next()) {  
 int id = result.getInt("componentID");  
 String nameComponent = result.getString("name");  
 int quantity = result.getInt("quantity");  
 components.add(new Component(id,nameComponent,quantity));  
  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return components;  
 }  
}

Klasa Employee reprezentuje ogólnego pracownika w systemie i zawiera funkcje do zarządzania zadaniami, zamówieniami, pracownikami, wyposażeniem, itp.

Klasa ta stanowi bazę dla bardziej wyspecjalizowanych klas pracowników, takich jak Manager, Admin, i ProductionEmployee. Dziedziczenie pozwala na rozszerzenie funkcjonalności związanej z różnymi rolami pracowników.

W metodach tej klasy wykorzystano następujące zapytania do bazy danych:

"SELECT \n" +  
 " oq.orderQuantityID,\n" +  
 " p.productID,\n" +  
 " p.name AS productName,\n" +  
 " o.Status AS orderStatus,\n" +  
 " oq.quantityOrdered,\n" +  
 " oq.QuantityInProduction,\n" +  
 " oq.OuantityFinished\n" +  
 "FROM \n" +  
 " orderQuantity oq\n" +  
 "JOIN \n" +  
 " product p ON oq.productID = p.productID\n" +  
 "JOIN \n" +  
 " orders o ON oq.orderID = o.OrderID;";

To zapytanie SQL jest używane do pobierania informacji o ilości zamówionych produktów, ich statusie i ilości dostępnych w produkcji i ukończonych.

"SELECT e.employeeID, e.name, e.lastName, r.roleName AS role, z.name AS zone\n" +  
 "FROM Employee e\n" +  
 "JOIN role r ON e.roleID = r.roleID\n" +  
 "JOIN zone z ON e.zoneID = z.zoneID;";

Zapytanie zwraca informacje o pracownikach wraz z ich identyfikatorem, imieniem, nazwiskiem, nazwą roli i nazwą strefy, łącząc dane z tabel Employee, role i zone.

"SELECT \n" +  
 " t.taskID AS taskID,\n" +  
 " t.name AS name,\n" +  
 " t.priority,\n" +  
 " t.description,\n" +  
 " t.norm,\n" +  
 " z.name AS zone\_name,\n" +  
 " t.quantity\n" +  
 "FROM \n" +  
 " Task t\n" +  
 "JOIN \n" +  
 " Zone z ON t.zoneID = z.zoneID;\n";

Zwraca informacje o zadaniach, zawierając identyfikator zadania taskID, nazwę zadania, priorytet, opis, normę, nazwę strefy i ilość. Zadania są pobierane z tabeli Task, a informacje o strefie są pobierane z tabeli Zone poprzez połączenie na podstawie identyfikatora strefy zoneID.

"SELECT \n" +  
 " e.equipmentID AS equipmentID,\n" +  
 " e.name AS equipmentName,\n" +  
 " e.status,\n" +  
 " z.name AS zoneName\n" +  
 "FROM \n" +  
 " equipment e\n" +  
 "JOIN \n" +  
 " zone z ON e.zoneID = z.zoneID;";

To zapytanie SQL zwraca informacje o sprzęcie, obejmujące identyfikator sprzęt, nazwę sprzętu, status oraz nazwę strefy.

"SELECT equipment.equipmentID,equipment.name, equipmentCategory.name AS equipment\_category, equipment.status, zone.name AS equipment\_zone\n" +  
 "FROM taskCategory\n" +  
 "JOIN taskEquipmentCategory ON taskCategory.taskCategoryID = taskEquipmentCategory.taskCategoryId\n" +  
 "JOIN equipment ON taskEquipmentCategory.equipmentCategoryId = equipment.equipmentCategoryId\n" +  
 "JOIN equipmentCategory ON equipment.equipmentCategoryId = equipmentCategory.equipmentCategoryID\n" +  
 "JOIN zone ON equipment.zoneId = zone.zoneID\n" +  
 "WHERE taskCategory.name =?;";

Zwraca informacje o sprzęcie, które jest powiązane z kategorią zadania o określonej nazwie. Wybierane są kolumny: equipmentID, name, equipment\_category, status oraz equipment\_zone. Zapytanie to korzysta z wielu połączeń między różnymi tabelami: taskCategory, taskEquipmentCategory, equipment, equipmentCategory i zone. Warunek WHERE ogranicza wyniki do kategorii zadań o określonej nazwie (taskCategory.name = ?).

"SELECT component.componentID, Component.name, component.quantity \n" +  
 "FROM TaskCategory\n" +  
 "JOIN TaskCategoryComponent ON TaskCategory.taskCategoryID = TaskCategoryComponent.taskCategoryID\n" +  
 "JOIN Component ON Component.componentID = TaskCategoryComponent.componentID\n" +  
 "WHERE TaskCategory.name =?;";

Zwraca informacje o komponentach związanych z kategorią zadania o określonej nazwie.

### Klasa Equipment

package server;  
  
public class Equipment extends Inventory{  
 private String status;  
 private String zone;  
 public Equipment(int id, String name, String status, String zone) {  
 super(id, name);  
 this.status = status;  
 this.zone = zone;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public String getZone() {  
 return zone;  
 }  
  
 @Override  
 public void reserve(Task task) {  
  
 }  
  
 @Override  
 public void cancelReservation(Task task) {  
  
 }  
}

Podobnie jak w przypadku klasy Component jest to „przepis” na sprzęt zawierający niezbędne pola, konstruktor i metody takie jak gettery i setery.

Klasa ta dziedziczy po klasie Inventory.

### Klasa Inventory

package server;  
  
import java.io.Serializable;  
  
public abstract class Inventory implements Serializable {  
 private int id;  
 private String name;  
  
 public Inventory(int id, String name) {  
 this.id = id;  
 this.name = name;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public abstract void reserve(Task task);  
 public abstract void cancelReservation(Task task);  
}

Klasa abstrakcyjna Inventory służy jako podstawa dla różnych rodzajów inwentarzy umożliwiając rozszerzanie i implementowanie specyficznej logiki dla konkretnych typów inwentarzy.

### Klasa Leader

package server;  
  
import java.util.List;  
  
public class Leader extends Employee {  
 public Leader(int id, String name, String lastName, String role, String zone, String login, String password) {  
 super(id, name, lastName, role, zone, login, password);  
 }  
  
 public Leader(int id, String name, String lastName, String role, String zone, String login, String password, Task task) {  
 super(id, name, lastName, role, zone, login, password, task);  
 }  
  
  
  
 public List<Employee> getListOfEmployeeAndTask() {  
return null;  
 }  
  
 public void getTimeReport() {  
  
 }  
  
 public void getZoneReport() {  
  
 }  
public void getEmployeeReport(){  
  
}  
public void getEquipmentStatusReport(){  
  
}  
public void getEquipmentTimeOfUseReport(){  
  
}  
  
}

Klasa ta jest podstawą do wdrożenia operacji które może w naszym systemie wykonywać Leader. Klasa ta będzie rozwijana w dalszej części implementacji aplikacji.

### Klasa License

package server;  
  
public class License {  
 private String id;  
 private String name;  
 private String description;  
  
 public License(String id, String name, String description) {  
 this.id = id;  
 this.name = name;  
 this.description = description;  
 }  
 public boolean checkLicense(Employee employee, Task task){  
 return true;  
 }  
}

Klasa ta to również “przepis” jak ma wyglądać i zachowywać się Licencja zawiera pola, konstruktor i metody niezbędne do zaimplementowania aplikacji.

### Klasa Login

package server;  
import java.util.Date;  
  
import java.sql.\*;  
import java.text.SimpleDateFormat;  
  
public class Login {  
 private static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/system";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
 private Connection connection;  
public Login(){  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
}  
 public boolean check(String login, String password){  
  
 String sqlQuery = "SELECT \* FROM employee WHERE login = ? AND password = ?";  
 boolean flag = false;  
 try (PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery)) {  
 preparedStatement.setString(1, login);  
 preparedStatement.setString(2, password);  
 ResultSet result = preparedStatement.executeQuery();  
 if( result.next())  
 flag=true;  
  
 }catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return flag;  
 }  
 public void startLogin(Employee employee){  
 int id = employee.getId();  
 Date data = new Date();  
 SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
 String dataFormat = format.format(data);  
 employee.setStartTime(dataFormat);  
  
 String sqlQuery = "insert into loginHistory (employeeID,startTime) values (?,?)";  
 try {  
 PreparedStatement statement =connection.prepareStatement(sqlQuery);  
 statement.setInt(1,id);  
 statement.setString(2,dataFormat);  
 statement.executeUpdate();  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 }  
 public void endLogin(Employee employee){  
 int id = employee.getId();  
 Date data = new Date();  
 SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
 String dataFormat = format.format(data);  
 String startTime = employee.getStartTime();  
  
 String sqlQuery = "update loginHistory SET startTime=?,endTime=? where employeeID=?";  
 try {  
 PreparedStatement statement =connection.prepareStatement(sqlQuery);  
 statement.setString(1,startTime);  
 statement.setString(2,dataFormat);  
 statement.setInt(3,id);  
 statement.executeUpdate();  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 }  
}

Klasa Login obsługuje proces logowania pracowników do systemu. Wykorzystuje połączenie z bazą danych MySQL do weryfikacji loginu i hasła pracownika. Dodatkowo, rejestruje datę i czas rozpoczęcia oraz zakończenia sesji logowania w tabeli loginHistory.

### Klasa Manager

package server;  
  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class Manager extends Leader{  
 private static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/system";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
 private Connection connection;  
 public Manager(int id, String name, String lastName, String role, String zone, String login, String password) {  
 super(id, name, lastName, role, zone, login, password);  
 }  
  
 public Manager(int id, String name, String lastName, String role, String zone, String login, String password, Task task) {  
 super(id, name, lastName, role, zone, login, password, task);  
 }  
 public void addTaskCategory(String category){  
  
 }  
 public void addNewTask(Task task, Employee employee){  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT taskCategory.taskCategoryID\n" +  
 "FROM taskCategory\n" +  
 "WHERE taskCategory.name = (SELECT product.name FROM product WHERE product.productID = ?); ";  
 int productID = task.getProductID();  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1,productID);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()) {  
 int taskCategoryID = resultSet.getInt("taskCategoryID");  
 query = "select zoneID from zone where name=?";  
 preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1, task.getEquipment().getZone());  
 resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()){  
 int zoneID = resultSet.getInt("zoneID");  
 query = "insert into task (name,priority,description, taskCategory,norm,productID,quantity,orderID,zoneID) values (?,?,?,?,?,?,?,?,?);";  
 preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1, task.getName());  
 preparedStatement.setString(2, task.getPriority());  
 preparedStatement.setString(3, task.getDescription());  
 preparedStatement.setInt(4, taskCategoryID);  
 preparedStatement.setInt(5, task.getNorm());  
 preparedStatement.setInt(6, task.getProductID());  
 preparedStatement.setInt(7, task.getQuantity());  
 preparedStatement.setInt(8, task.getOrderID());  
 preparedStatement.setInt(9,zoneID);  
 preparedStatement.executeUpdate();  
 query = "SELECT taskID\n" +  
 "FROM task\n" +  
 "ORDER BY taskID DESC\n" +  
 "LIMIT 1;\n";  
 preparedStatement = connection.prepareStatement(query);  
 resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()) {  
 int id = resultSet.getInt("taskID");  
 query = "UPDATE orderQuantity SET QuantityInProduction = QuantityInProduction + ?, orderID=2 WHERE orderQuantityID = ?;";  
 preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1, task.getQuantity());  
 preparedStatement.setInt(2, task.getOrderID());  
 preparedStatement.executeUpdate();  
 query = "insert into taskEquipment(taskID, equipmentID) values (?,?)";  
 preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1, id);  
 preparedStatement.setInt(2, task.getEquipment().getId());  
 preparedStatement.executeUpdate();  
 for (int i = 0; i < task.getComponent().size(); i++) {  
 query = "insert into taskComponent(taskID, componentID,quantity)values (?,?,?)";  
 preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1, id);  
 preparedStatement.setInt(2, task.getComponent().get(i).getId());  
 preparedStatement.setInt(3, task.getQuantity());  
 preparedStatement.executeUpdate();  
 }  
 query = "insert into result (resultID, quantityOK, quantityNOK) values (?,0,0)";  
 preparedStatement=connection.prepareStatement(query);  
 preparedStatement.setInt(1,id);  
 preparedStatement.executeUpdate();  
 query="update task set resultID=? where taskID=?";  
 preparedStatement=connection.prepareStatement(query);  
 preparedStatement.setInt(1,id);  
 preparedStatement.setInt(2,id);  
 preparedStatement.executeUpdate();  
 int employeeID = employee.getId();  
 query="insert into taskStatus(taskID,employeeID,stepName,startStep) values (?,?,'available',CURRENT\_TIMESTAMP)";  
 preparedStatement=connection.prepareStatement(query);  
 preparedStatement.setInt(1,id);  
 preparedStatement.setInt(2,employeeID);  
 preparedStatement.executeUpdate();  
  
 }  
 } }  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 }  
 public List<List<String>> getEquipmentTimeOfUseReport(int id){  
 List<List<String>> useEquipment = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT\n" +  
 " employee.name,\n" +  
 " employee.lastName,\n" +  
 " taskStatus.startStep,\n" +  
 " taskStatus.endStep,\n" +  
 " task.name as taskName,\n" +  
 " TIMESTAMPDIFF(HOUR, taskStatus.startStep, taskStatus.endStep) AS hoursInUse\n" +  
 "FROM\n" +  
 " equipment\n" +  
 "JOIN taskEquipment ON taskEquipment.equipmentID = equipment.equipmentID\n" +  
 "JOIN task ON task.taskID = taskEquipment.taskID\n" +  
 "JOIN taskStatus ON taskStatus.taskID = task.taskID\n" +  
 "JOIN employee ON employee.employeeID = taskStatus.employeeID\n" +  
 "WHERE\n" +  
 " equipment.equipmentID = ?;\n";  
  
try{  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1,id);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()){  
 String name = resultSet.getString("name");  
 String lastName = resultSet.getString("lastName");  
 String startDate = resultSet.getString("startStep");  
 String endDate = resultSet.getString("endStep");  
 String taskName = resultSet.getString("taskName");  
 int use = resultSet.getInt("hoursInUse");  
 useEquipment.add(Arrays.*asList*(name,lastName,startDate,endDate,taskName,use+""));  
 }  
 for (List<String> innerList : useEquipment) {  
 for (String value : innerList) {  
 System.*out*.print(value + " ");  
 }  
 System.*out*.println();  
 }  
}catch (SQLException e){  
 e.printStackTrace();  
}  
return useEquipment;  
 }  
 public void addTaskPriority(Task task, String priority){  
  
 }  
 public void changeTaskPriority(Task task, String priority){  
  
 }  
 public void addLicenseToEmployee(Employee employee, License license){  
  
 }  
 public void addEquipment(Equipment equipment){  
  
 }  
 public void changeEquipmentStatus(Equipment equipment){  
 int id = equipment.getId();  
 String status = equipment.getStatus();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "update equipment set status=? where equipmentID=?";  
 try{  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setString(1,status);  
 preparedStatement.setInt(2,id);  
 preparedStatement.executeUpdate();  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 }  
  
  
}

Klasa Manager rozszerza klasę Leader i reprezentuje menadżera w systemie. Odpowiada za zarządzanie zadaniami, pracownikami, oraz monitorowanie używania sprzętu.

### Klasa MySQLDatabaseConnector

package server;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
public class MySQLDatabaseConnector {  
  
 private static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/system";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
 private Connection connection;  
  
 public MySQLDatabaseConnector() {  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public Employee getUserInfo(String user, String password) {  
 String sqlQuery = "SELECT \* FROM employee WHERE login = ? AND password = ?";  
  
 try (PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery)) {  
 preparedStatement.setString(1, user);  
 preparedStatement.setString(2, password);  
 try (ResultSet result = preparedStatement.executeQuery()) {  
 if (result.next()) {  
 int id = result.getInt("employeeID");  
 String name = result.getString("name");  
 String lastName = result.getString("lastName");  
 int roleID = result.getInt("roleID");  
 int zoneID = result.getInt("zoneID");  
  
 sqlQuery = "Select roleName from role where roleID = ?";  
 try (PreparedStatement roleStatement = connection.prepareStatement(sqlQuery)) {  
 roleStatement.setInt(1, roleID);  
 try (ResultSet roleResult = roleStatement.executeQuery()) {  
 if (roleResult.next()) {  
 String roleName = roleResult.getString("roleName");  
  
 sqlQuery = "Select name from zone where zoneID = ?";  
 try (PreparedStatement zoneStatement = connection.prepareStatement(sqlQuery)) {  
 zoneStatement.setInt(1, zoneID);  
 try (ResultSet zoneResult = zoneStatement.executeQuery()) {  
 if (zoneResult.next()) {  
 String zoneName = zoneResult.getString("name");  
 if(roleName.equals("Production Employee"))  
 return new ProductionEmployee(id, name, lastName,roleName,zoneName,user,password );  
 else if (roleName.equals("Admin"))  
 return new Admin(id, name, lastName,roleName,zoneName,user,password);  
 else if (roleName.equals("Leader"))  
 return new Leader(id, name, lastName,roleName,zoneName,user,password);  
 else if (roleName.equals("Manager")) {  
 return new Manager(id, name, lastName,roleName,zoneName,user,password);  
 }else  
 return new ProductionEmployee(id, name, lastName,roleName,zoneName,user,password);  
  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
 public List<String> getRolesList(){  
 List<String> roles = new ArrayList<>();  
 String sqlQuery = "SELECT roleName from role";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);  
 ResultSet result = preparedStatement.executeQuery();  
 while (result.next()) {  
  
 String name = result.getString("roleName");  
  
 roles.add(name);  
 }  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 return roles;  
 }  
 public List<String> getZonesList(){  
 List<String> zones = new ArrayList<>();  
 String sqlQuery = "SELECT name from zone";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);  
 ResultSet result = preparedStatement.executeQuery();  
 while (result.next()) {  
  
 String name = result.getString("name");  
  
 zones.add(name);  
 }  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 return zones;  
 }  
 public void closeConnection() {  
 try {  
 if (connection != null && !connection.isClosed()) {  
 connection.close();  
 System.*out*.println("Połączenie z bazą danych zostało zamknięte.");  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void updateEmployee(int id, String role, String zone) {  
 String sqlQurey = "SELECT roleID FROM role WHERE roleName=?";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(sqlQurey);  
 preparedStatement.setString(1,role);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 if (resultSet.next()){  
 int idRole = resultSet.getInt("roleID");  
  
 sqlQurey = "SELECT zoneID from zone where name=?";  
 preparedStatement = connection.prepareStatement(sqlQurey);  
 preparedStatement.setString(1,zone);  
 resultSet = preparedStatement.executeQuery();  
 if (resultSet.next()) {  
 int idZone = resultSet.getInt("zoneID");  
  
 sqlQurey = "UPDATE employee set roleID= ?, zoneID= ? where employeeID= ?";  
 preparedStatement = connection.prepareStatement(sqlQurey);  
 preparedStatement.setInt(1, idRole);  
 preparedStatement.setInt(2,idZone);  
 preparedStatement.setInt(3,id);  
 preparedStatement.executeUpdate();  
 }}  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
 }  
}

Klasa MySQLDatabaseConnector służy do komunikacji z bazą danych MySQL w systemie.

### Klasa Order

package server;  
  
import java.io.Serializable;  
  
public class Order implements Serializable {  
 private int id;  
 private String status;  
 private Product product;  
  
 public Order(int id, String status, Product product) {  
 this.id = id;  
 this.status = status;  
 this.product = product;  
 }  
  
 public Product getProduct() {  
 return product;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
}

Klasa Order reprezentuje zamówienie w systemie, zawierając informacje o identyfikatorze zamówienia, statusie oraz przypisanym do niego produkcie, umożliwiając manipulację danymi związanych z zamówieniem.

### Klasa Product

package server;  
  
import java.io.Serializable;  
  
public class Product implements Serializable {  
 private int id;  
 private String name;  
 private int quantityOrdered;  
 private int quantityInProduction;  
 private int quantityFinished;  
  
 public Product( String name, int quantityOrdered) {  
 this.name = name;  
 this.quantityOrdered = quantityOrdered;  
 }  
  
 public Product( int id, String name, int quantityOrdered, int quantityInProduction, int quantityFinished) {  
 this.id = id;  
 this.name = name;  
 this.quantityOrdered = quantityOrdered;  
 this.quantityInProduction = quantityInProduction;  
 this.quantityFinished = quantityFinished;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getQuantityOrdered() {  
 return quantityOrdered;  
 }  
  
 public int getQuantityInProduction() {  
 return quantityInProduction;  
 }  
  
 public int getQuantityFinished() {  
 return quantityFinished;  
 }  
  
 @Override  
 public String toString() {  
 return name +  
 "\t" + quantityOrdered +  
 "\t" + quantityInProduction +  
 "\t" + quantityFinished ;  
 }  
  
 public void setQuantityInProduction(int quantityInProduction) {  
 this.quantityInProduction = quantityInProduction;  
 }  
  
 public int getId() {  
 return id;  
 }  
}

Klasa Product reprezentuje produkt w systemie, przechowując informacje o identyfikatorze, nazwie, ilości zamówionej, ilości w produkcji oraz ilości ukończonej.

### Klasa ProductionEmployee

package server;  
  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class ProductionEmployee extends Employee{  
 private static final String *JDBC\_URL* = "jdbc:mysql://localhost:3306/system";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
 private Connection connection;  
 public ProductionEmployee(int id, String name, String lastName, String role, String zone, String login, String password) {  
 super(id, name, lastName, role, zone, login, password);  
 }  
  
 public ProductionEmployee(int id, String name, String lastName, String role, String zone, String login, String password, Task task) {  
 super(id, name, lastName, role, zone, login, password, task);  
 }  
  
 public Task getMyTask(Employee employee)  
 {  
 Task task = null;  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT \n" +  
 " task.taskID,\n" +  
 " MAX(task.name) as taskName,\n" +  
 " MAX(task.priority) as taskPriority,\n" +  
 " MAX(task.description) as taskDescription,\n" +  
 " MAX(taskCategory.name) as nameCategory,\n" +  
 " MAX(task.norm) as taskNorm,\n" +  
 " MAX(result.quantityOK) as quantityOK,\n" +  
 " MAX(result.quantityNOK) as quantityNOK,\n" +  
 " MAX(taskStatus.stepName) as status,\n" +  
 " MAX(product.name) as productName,\n" +  
 " MAX(task.quantity) as taskQuantity,\n" +  
 " MAX(zone.name) as zoneName,\n" +  
 " MAX(equipment.name) as equipmentName,\n" +  
 " GROUP\_CONCAT(component.name) as componentName \n" +  
 "FROM \n" +  
 " task\n" +  
 "JOIN \n" +  
 " taskStatus ON taskStatus.taskID = task.taskID\n" +  
 "JOIN \n" +  
 " taskCategory ON taskCategory.taskCategoryID = task.taskCategory\n" +  
 "JOIN \n" +  
 " result ON result.resultID = task.resultID\n" +  
 "JOIN \n" +  
 " product ON product.productID = task.productID\n" +  
 "JOIN \n" +  
 " zone ON zone.zoneID = task.zoneID\n" +  
 "JOIN \n" +  
 " taskEquipment ON taskEquipment.taskID = task.taskID\n" +  
 "JOIN \n" +  
 " equipment ON equipment.equipmentID = taskEquipment.equipmentID\n" +  
 "JOIN \n" +  
 " taskComponent ON taskComponent.taskID = task.taskID\n" +  
 "JOIN \n" +  
 " component ON component.componentID = taskComponent.componentID\n" +  
 "WHERE \n" +  
 " taskStatus.employeeID = ?\n" +  
 " AND taskStatus.stepName = 'in progress' \n" +  
 " AND taskStatus.endStep = '0000-00-00 00-00-00'\n" +  
 "GROUP BY \n" +  
 " task.taskID;\n";  
 try{  
 int employeeID = employee.getId();  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1,employeeID);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()){  
 int taskID = resultSet.getInt("taskID");  
 String taskName = resultSet.getString("taskName");  
 String priority = resultSet.getString("taskPriority");  
 String description = resultSet.getString("taskDescription");  
 String category = resultSet.getString("nameCategory");  
 int norm = resultSet.getInt("taskNorm");  
 int quantityOK = resultSet.getInt("quantityOK");  
 int quantityNOK = resultSet.getInt("quantityNOK");  
 String status = resultSet.getString("status");  
 String product = resultSet.getString("productName");  
 int quantity = resultSet.getInt("taskQuantity");  
 String zone = resultSet.getString("zoneName");  
 String equipment = resultSet.getString("equipmentName");  
 String component = resultSet.getString("componentName");  
 task = new Task(taskID,taskName,priority,description,category,norm,quantityOK,quantityNOK,status,product,quantity,zone,equipment,component);  
  
  
 }  
 }catch (SQLException e){  
 e.printStackTrace();  
 }  
return task;  
 }  
 public List<Task> getListOfTask(int employeeID) {  
 List<Task> tasks = new ArrayList<>();  
 try {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(*JDBC\_URL*, *USER*, *PASSWORD*);  
 System.*out*.println("Pomyślnie połączono z bazą danych");  
 } catch (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 String query = "SELECT \n" +  
 " t.taskID,\n" +  
 " t.name,\n" +  
 " t.priority,\n" +  
 " t.description,\n" +  
 " p.name as product,\n" +  
 " t.quantity,\n" +  
 " t.norm,\n" +  
 " ts.stepName as status,\n" +  
 " GROUP\_CONCAT(DISTINCT eq.name) as equipment,\n" +  
 " GROUP\_CONCAT(DISTINCT c.name) as component\n" +  
 "FROM \n" +  
 " task t\n" +  
 "JOIN \n" +  
 " product p ON p.productID = t.productID \n" +  
 "JOIN \n" +  
 " taskstatus ts ON ts.taskID = t.taskID \n" +  
 "JOIN \n" +  
 " taskequipment te ON te.taskID = t.taskID\n" +  
 "JOIN \n" +  
 " equipment eq ON eq.equipmentID = te.equipmentID\n" +  
 "JOIN \n" +  
 " taskcomponent tc ON tc.taskID = t.taskID\n" +  
 "JOIN \n" +  
 " component c ON c.componentID = tc.componentID\n" +  
 "WHERE \n" +  
 " ts.endStep = '0000-00-00 00:00:00' AND ts.stepName = 'available'\n" +  
 " AND t.taskCategory IN (\n" +  
 " SELECT DISTINCT taskcategoryid \n" +  
 " FROM taskcategorylicense \n" +  
 " WHERE licenseID IN (\n" +  
 " SELECT licenseId \n" +  
 " FROM employeelicense \n" +  
 " WHERE employeeId = ? AND expirationDate >= CURRENT\_DATE\n" +  
 " )\n" +  
 " )\n" +  
 " AND eq.equipmentCategoryID IN (\n" +  
 " SELECT equipmentcategoryId \n" +  
 " FROM equipmentcategorylicense \n" +  
 " WHERE licenseID IN (\n" +  
 " SELECT licenseId \n" +  
 " FROM employeelicense \n" +  
 " WHERE employeeId = ? AND expirationDate >= CURRENT\_DATE\n" +  
 " )\n" +  
 " )\n" +  
 " \n" +  
 "GROUP BY \n" +  
 " t.taskID;\n";  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 preparedStatement.setInt(1, employeeID);  
 preparedStatement.setInt(2,employeeID);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()) {  
 int taskID = resultSet.getInt("taskID");  
 String name = resultSet.getString("name");  
 String priority = resultSet.getString("priority");  
 String description = resultSet.getString("description");  
 String productName = resultSet.getString("product");  
 int quantity = resultSet.getInt("quantity");  
 int norm = resultSet.getInt("norm");  
 String status = resultSet.getString("status");  
 String equipment = resultSet.getString("equipment");  
 String component = resultSet.getString("component");  
 tasks.add(new Task(taskID, name, priority, description, productName, quantity, norm, status, equipment, component));  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return tasks;  
 }  
}

Klasa ProductionEmployee rozszerza klasę Employee i odpowiada za funkcjonalność pracownika produkcyjnego w systemie. Metody tej klasy umożliwiają pobranie zadania przypisanego do danego pracownika w trakcie produkcji oraz uzyskanie listy dostępnych zadań, które może on podjąć.

### Klasa Result

package server;  
  
public class Result {  
 private int quantityOK;  
 private int quantityNOK;  
 private String information;  
  
 public Result(int quantityOK, int quantityNOK, String information) {  
 this.quantityOK = quantityOK;  
 this.quantityNOK = quantityNOK;  
 this.information = information;  
 }  
}

Klasa Result reprezentuje wynik produkcji, przechowując informacje o ilości poprawnie i niepoprawnie wykonanych elementów oraz ewentualnych dodatkowych informacjach, umożliwiając zapis i manipulację wynikami produkcyjnymi.

### Klasa ServerTCP

package server;  
  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.List;  
import java.util.Scanner;  
import java.util.concurrent.\*;  
  
public class ServerTCP {  
 private static final int *MAX\_CLIENTS* = 100;  
  
 public static void main(String[] args) {  
 ExecutorService executorService = Executors.*newFixedThreadPool*(*MAX\_CLIENTS*);  
 try (ServerSocket serverSocket = new ServerSocket(12345);) {  
 System.*out*.println("Serwer nasłuchuje na porcie 12345");  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 System.*out*.println("Połączono z " + clientSocket);  
 executorService.execute(new ClientHandler(clientSocket));  
  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 executorService.shutdown();  
 }  
 }  
}

Klasa ServerTCP reprezentuje prosty serwer obsługujący połączenia z wieloma klientami przy użyciu protokołu TCP. Serwer nasłuchuje na porcie 12345 i obsługuje przychodzące połączenia, uruchamiając dla każdego z nich nowy wątekClientHandler

### Klasa Task

package server;  
  
import java.io.Serializable;  
import java.sql.Time;  
import java.util.List;  
  
public class Task implements Serializable {  
 private String componentName;  
 private String equipmentName;  
 private String product;  
 private int quantityNOK;  
 private int quantityOK;  
 private int taskID;  
 private String name;  
 private String priority;  
 private String description;  
 private int norm;  
 private List<Component> component;  
 private Equipment equipment;  
 private String status;  
 private Time timeInStep;  
 private Employee employee;  
 private String zone;  
 private int quantity;  
 private int productID;  
 private int orderID;  
 private String category;  
  
  
 public Task(int taskID, String name, String priority, String description, int norm, String zone, int quantity) {  
 this.taskID = taskID;  
 this.name = name;  
 this.priority = priority;  
 this.description = description;  
 this.norm = norm;  
 this.zone=zone;  
 this.quantity=quantity;  
 }  
 public Task(int taskID, String name, String priority, String description, int norm, List<Component> component, Equipment equipment, String zone, int quantity, int productID, int orderID) {  
 this.taskID = taskID;  
 this.name = name;  
 this.priority = priority;  
 this.description = description;  
 this.norm = norm;  
 this.component = component;  
 this.equipment=equipment;  
 this.zone=zone;  
 this.quantity=quantity;  
 this.productID=productID;  
 this.orderID=orderID;  
 }  
  
 public Task(int taskID, String name, String priority, String description, int norm, List<Component> component, Equipment equipment, String status, Time timeInStep, Employee employee) {  
 this.taskID = taskID;  
 this.name = name;  
 this.priority = priority;  
 this.description = description;  
 this.norm = norm;  
 this.component = component;  
 this.equipment = equipment;  
 this.status = status;  
 this.timeInStep = timeInStep;  
 this.employee = employee;  
 }  
  
 public Task(int taskID, String taskName, String priority, String description, String category, int norm, int quantityOK,  
 int quantityNOK, String status,String product, int quantity, String zone, String equipment, String component) {  
 this.taskID=taskID;  
 this.name=taskName;  
 this.priority=priority;  
 this.description=description;  
 this.category = category;  
 this.norm=norm;  
 this.quantityOK = quantityOK;  
 this.quantityNOK = quantityNOK;  
 this.product = product;  
 this.quantity = quantity;  
 this.zone =zone;  
 this.equipmentName = equipment;  
 this.componentName = component;  
 this.status=status;  
  
  
 }  
  
 public Task(int taskID, String name, String priority, String description, String productName, int quantity, int norm, String status, String equipment, String component) {  
 this.taskID=taskID;  
 this.name=name;  
 this.priority=priority;  
 this.description=description;  
 this.norm=norm;  
 this.product = productName;  
 this.quantity = quantity;  
 this.equipmentName = equipment;  
 this.componentName = component;  
 this.status = status;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getProductID() {  
 return productID;  
 }  
  
 public int getOrderID() {  
 return orderID;  
 }  
  
 public String getPriority() {  
 return priority;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public int getNorm() {  
 return norm;  
 }  
  
 public String getZone() {  
 return zone;  
 }  
  
 public List<Component> getComponent() {  
 return component;  
 }  
  
 public Equipment getEquipment() {  
 return equipment;  
 }  
  
 public int getQuantity() {  
 return quantity;  
 }  
  
 public int getTaskID() {  
 return taskID;  
 }  
 public void viewTaskDetails(){  
  
 }  
  
 public int getId() {  
 return taskID;  
 }  
}

Klasa Task reprezentuje zadanie w systemie produkcyjnym i przechowuje informacje takie jak identyfikator zadania, nazwa, priorytet, opis, norma, lista komponentów, przypisane urządzenie, status, czas w danym etapie, pracownik oraz inne parametry związane z produkcją. Klasa ta została zaprojektowana do obsługi różnych aspektów zadań, takich jak ich tworzenie, przypisywanie, monitorowanie postępu czy przeglądanie szczegółów.