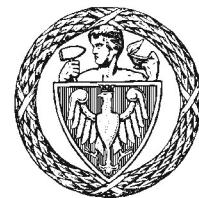


# Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI  
I TECHNIK INFORMACYJNYCH



Projektowanie i integracja systemów (PIS)

MAGAZYN



Daniel Kobiałka  
[daniel.kobiałka.stud@pw.edu.pl](mailto:daniel.kobiałka.stud@pw.edu.pl)  
310744

Bartłomiej Dudek  
[bartłomiej.dudek.stud@pw.edu.pl](mailto:bartłomiej.dudek.stud@pw.edu.pl)  
310623

Adam Sudół  
[adam.sudol.stud@pw.edu.pl](mailto:adam.sudol.stud@pw.edu.pl)  
310933

Karol Rogoziński  
[karol.rogozinski.stud@pw.edu.pl](mailto:karol.rogozinski.stud@pw.edu.pl)  
310883

Gabriela Topczewska  
[gabriela.topczewska.stud@pw.edu.pl](mailto:gabriela.topczewska.stud@pw.edu.pl)  
310961

09.01.2023

# Spis treści

<b>1 Postawiony problem</b>	<b>3</b>
1.1 Pomysł . . . . .	3
1.2 Główne funkcjonalności . . . . .	3
<b>2 Proponowana implementacja</b>	<b>3</b>
<b>3 Stos technologiczny</b>	<b>4</b>
<b>4 Organizacja pracy</b>	<b>4</b>
4.1 Zespół . . . . .	4
4.2 Komunikacja . . . . .	5
<b>5 Struktura projektu - repozytorium</b>	<b>5</b>
<b>6 Architektura</b>	<b>6</b>
<b>7 Backend</b>	<b>6</b>
7.1 PDFBox . . . . .	7
7.2 ELK Stack . . . . .	7
7.2.1 ElasticSearch . . . . .	7
7.2.2 Logstash . . . . .	7
<b>8 Frontend</b>	<b>7</b>
<b>9 Baza danych</b>	<b>9</b>
9.1 Serwer . . . . .	9
9.2 Diagram ER . . . . .	10
9.3 Dokładne charakterystyki tabel . . . . .	11
9.3.1 Magazyny . . . . .	11
9.3.2 Sekcje . . . . .	11
9.3.3 Produkty . . . . .	11
9.3.4 Pracownicy . . . . .	12
9.3.5 Zlecenia . . . . .	12
9.3.6 Historia zleceń . . . . .	12
9.3.7 Raporty . . . . .	12
<b>10 Testy</b>	<b>12</b>
10.1 Testy jednostkowe . . . . .	13
10.2 Testy integracyjne . . . . .	13
<b>11 Deployment oraz uruchomienie aplikacji</b>	<b>13</b>
<b>12 Napotkane problemy i ich rozwiązania</b>	<b>13</b>
<b>13 Wnioski końcowe</b>	<b>14</b>

# 1 Postawiony problem

## 1.1 Pomysł

Pomysł na projekt zrodził się z zapotrzebowania osób trzecich na program służący do obsługi magazynów. Opis problemu został dostarczony drogą mailową i załączony jest poniżej:

*Jest plac, który stanowi powierzchnie magazynową. Powierzchnia podzielona jest na jakieś mniejsze powierzchnie które stanowią sektory. Każdy sektor ma jakiś tam metraż powierzchni w m<sup>2</sup>. Każdy sektor to powierzchnia na której składamy palety z pustakiem ceramicznym. Pustak występuje w sumie w 25-30 rodzajach. Są dwa rodzaje palet jeśli chodzi o rozmiar 120/100 cm i 136/100. Niektóre z wyrobów piętruje się na 3 do góry, niektóre na 4 warstwy. Są też jakieś drobnicowe materiały.*

*(...) Od poniedziałku do piątku 7-17 są prowadzone załadunki samochodów paletami z pustakiem z tegoż magazynu, jednocześnie 7 dni w tygodniu 24h idzie produkcja, i wyroby z tej produkcji są składowane na magazyn. Każdy kierowca przyjeżdżając na załadunek rejestruje się w terminalu samoobsługowym, gdzie dostaje dokument dostawy w którym jest opisane min co dokładnie ma mieć załadowane.*

*(...) Operator wózka podejmując kolejne auto do załadunku widzi na tablecie oprócz jakiś podstawowych danych czyli nr rejestracyjny auta, nazwisko kierowcy, widzi również co dokładnie ma załadować na to auto, a co najważniejsze widzi skąd ma to załadować, czyli np. 5 palet PTH25E3 z sektora B3 i 15 palet PTH 11,5 z sektora D8. Pobierając te palety z wyznaczonych miejsc potwierdza iż załadunek co jednocześnie powoduje iż w danych sektorach stan magazynu zostaje zmniejszony o wydana ilość i rodzaj. Jednocześnie inny wózkowy w tym samym czasie prowadząc wywóz palet z produkcji odstawia te palety w konkretny wcześniej wyznaczony przez koordynatora magazynu sektor.*

*Założenie jest takie , żeby na bieżąco było widać gdzie mamy ile zmagazynowane jakiego asortymentu, ile mamy powierzchni wolnej, ile w danym sektorze przy zapełnieniu np. w 30 % zmieści mi się jeszcze palet itp. Żeby można było zaplanować kolejność w jakiej poszczególne sektory/ asortyment będzie wydawany do klienta lub też zapełniony z produkcji.*

## 1.2 Główne funkcjonalności

Po analizie problemu sformułowano i wyznaczono główne funkcjonalności aplikacji:

1. definiowanie magazynów i ich podsekcji, w tym ich wymiarów -rola administratora,
2. przypisywanie podsekcjom konkretnych produktów,
3. zarządzanie kontami użytkowników,
4. tworzenie zleceń przez kierownika,
5. zarządzanie (zmiana statusu) przez pracownika,
6. przeglądanie historii zleceń,
7. przeglądanie przez pracownika wystawionych zleceń,
8. generowanie raportów o stanie magazynu.

# 2 Proponowana implementacja

W ramach projektu zaproponowano dostarczenie aplikację webową z backendem pisany w języku Java, frontendem z użyciem biblioteki Spring i relacyjną bazą danych na serwerze MySQL.

Proponowana implementacja umożliwia łatwy dostęp do aplikacji z dowolnego urządzenia (laptop, tablet, telefon komórkowy), co stanowi ważny element postawionego zadania.

### **3 Stos technologiczny**

Poniżej przedstawiono spis narzędzi i technologii używanych w projekcie:

- Java 17,
- SpringBoot,
- React
- Maven,
- Git,
- GitHub,
- MySQL,
- Aplikacje ELK:
  - ElasticSearch,
  - Logstash,
- JUnit,
- AssertJ
- DataJpaTest,
- Mockito,
- PDFBox,
- JDBC,
- HTML,
- CSS, SCSS,
- TypeScript,
- JavaScript,
- pnpm,
- Vite,
- maszyny wirtualne Azure,
- IDE: IntelliJ Idea, VSCode,
- dokumentacja: LaTeX.

### **4 Organizacja pracy**

#### **4.1 Zespół**

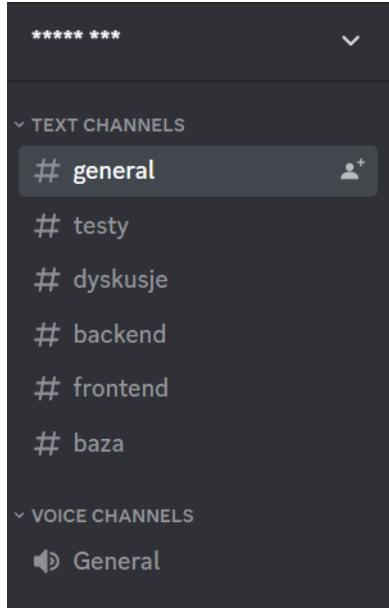
Zespół projektowy składa się z 5 osób. Są to, wraz z przydzielonymi im rolami i zadaniami w projekcie, kolejno:

- Daniel Kobiałka - lider techniczny zespołu; backend, deployment,
- Bartłomiej Dudek - komunikacja z mentorem, backend,
- Adam Sudoł - komunikacja z pomysłodawcą, testy,
- Karol Rogoziński - frontend,
- Gabriela Topczewska - baza danych, ELK Stack, dokumentacja,

W procesie tworzenia dokumentacji brali udział wszyscy członkowie zespołu.

## 4.2 Komunikacja

W celu organizacji pracy zdecydowano się używać portalu Discord jako głównego narzędzia komunikacyjnego. Serwer dostosowano do potrzeb projektu:



Oprócz Discorda do komunikacji używano komunikatora Messenger, kontaktu telefonicznego oraz spotkań na żywo. Komunikacja z mentorem zespołu przebiegała głównie drogą mailową. Stan prac i ich postęp kontrolowany był za pomocą narzędzi oraz opcji dostępnych w ramach serwisu GitHub oraz narzędzia Git.

## 5 Struktura projektu - repozytorium

Repozytorium projektu podzielone jest na 2 główne części:

- *magazyn-backend*,
- *magazyn-frontend*.

W folderze backendu, w folderze *src/main/java/pis.projekt* znajdziemy następujące podfoldery:

- *controller* - zawiera kontrolery służące do bezpośredniej komunikacji frontendu z serwisami,
- *interfaces* - deklaracje funkcjonalności serwisów,
- *models* - modele odpowiednich obiektów, tworzonych na wzór tabel z bazą danych,
- *repository* - obsługa komunikacji bezpośrednio z bazą danych,
- *services* - serwisy - warstwa pośrednia pomiędzy repozytoriami a kontrolerami,
- *utils* - dodatkowe funkcjonalności niezawarte w klasach z wcześniejszych lokalizacji.

W folderze */src* frontendu natomiast:

- *assets* - ikony,
- *components* - komponenty używane na ekranach aplikacji,
- *config* - pliki konfiguracyjne, w szczególności ścieżki do poszczególnych podstron,
- *context* - konteksty w aplikacji - nieużywane,
- *img* - wszystkie grafiki wykorzystane w projekcie,

- *pages* - implementacje każdej z podstron,
- *requests* - szablony do wysyłania GET i POST na backend,
- *styles* - ogólne style całej aplikacji,
- *templates* - przykładowe templatki.

## 6 Architektura

Architektura projektu opiera się na 3 podstawowych komponentach:

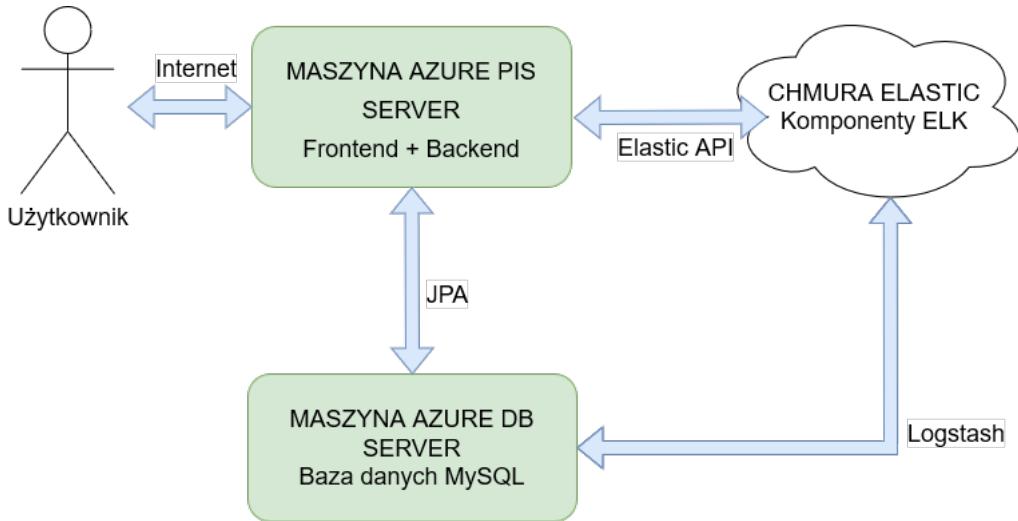
1. Maszyna Azure - Serwer dla frontendu oraz backendu,
2. Maszyna Azure - Serwer bazy danych MySQL,
3. Chmura ElasticCloud - chmura hostująca aplikacje z ELK Stack.

Odpowiednie komponenty komunikują się ze sobą w następujący sposób:

- Maszyny Azure komunikują się ze sobą za pośrednictwem JPA,
- Serwer frontendu i backendu komunikuje się z chmurą ElasticCloud za pomocą API wystawianego przez Elastic,
- Serwer bazy danych oraz usługi oferowane na ElasticCloud wykorzystują narzędzie Logstash do przesyłania danych między sobą.

Użytkownik końcowy korzysta z aplikacji łącząc się z maszyną hostującą frontend oraz backend za pośrednictwem internetu.

Poniżej przedstawiono poglądowy rysunek architektury zaproponowanego rozwiązania.



## 7 Backend

Backend aplikacji napisany został w języku Java z wykorzystaniem biblioteki SpringBoot. Aby udostępnić jednolite środowisko projektowe został stworzony projekt Maven. Dodatkowymi narzędziami użytymi w ramach backendu są: PDFBox oraz aplikacje ElasticStack.



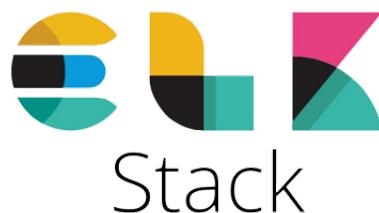
## 7.1 PDFBox

Narzędzie PDFBox zostało użyte do generowania raportów o stanie danego magazynu. Dzięki temu narzędziu było też możliwe odkodowanie zakodowanego do postaci ciągu bajtów pliku PDF, dzięki czemu zawartość pliku mogła zostać zaindeksowana w ElasticSearchu.



## 7.2 ELK Stack

W celu wykorzystania możliwości przeszukiwania pełnotekstowego w aplikacji wykorzystano narzędzia dostarczane przez ELK Stack. Skorzystano z możliwości hostowania serwisów w chmurze Elastic Cloud.



### 7.2.1 ElasticSearch

W chmurowej wersji ElasticSearchu przechowywana jest kopia danych z MySQLowej bazy w postaci indeksów i dokumentów. Dzięki temu możliwe jest skorzystanie z funkcjonalności oferowanej przez ElasticSearch.

Dla każdej tabeli z bazy relacyjnej istnieje odpowiadający jej indeks, w którym trzymane są dokumenty z rekordami. W szczególności istnieje indeks raportów, dzięki czemu generowane pliki PDF również mogą być przeszukiwane.

Funkcją która w tym projekcie jest szczególnie przydatna, jest wyszukiwanie dopuszczające popełnianie literówek.

### 7.2.2 Logstash

Dane pomiędzy bazą MySQL, a ElasticSearchem synchronizowane są za pomocą aplikacji Logstash. Napisany został skrypt konfiguracyjny, dzięki któremu z użyciem interfejsu JDBC dane z MySQLa są pobierane i wysyłane do odpowiadających im indeksów w ElasticSearchu. Z powodów technicznych skrypt Logstasha uruchamiany jest raz na jakiś czas, a dane aktualizowane są wsadowo (zob. sekcja *10. Napotkane problemy i ich rozwiązania*).

## 8 Frontend

Frontend został napisany z użyciem biblioteki React, która pozwala na reużywalność kodu. Dzięki temu poszczególne komponenty (np. różne rodzaje przycisków) mogą być zaimplementowane tylko raz, a następnie stosowane w wielu różnych miejscach. Wszystkie one znajdują się w folderze components.



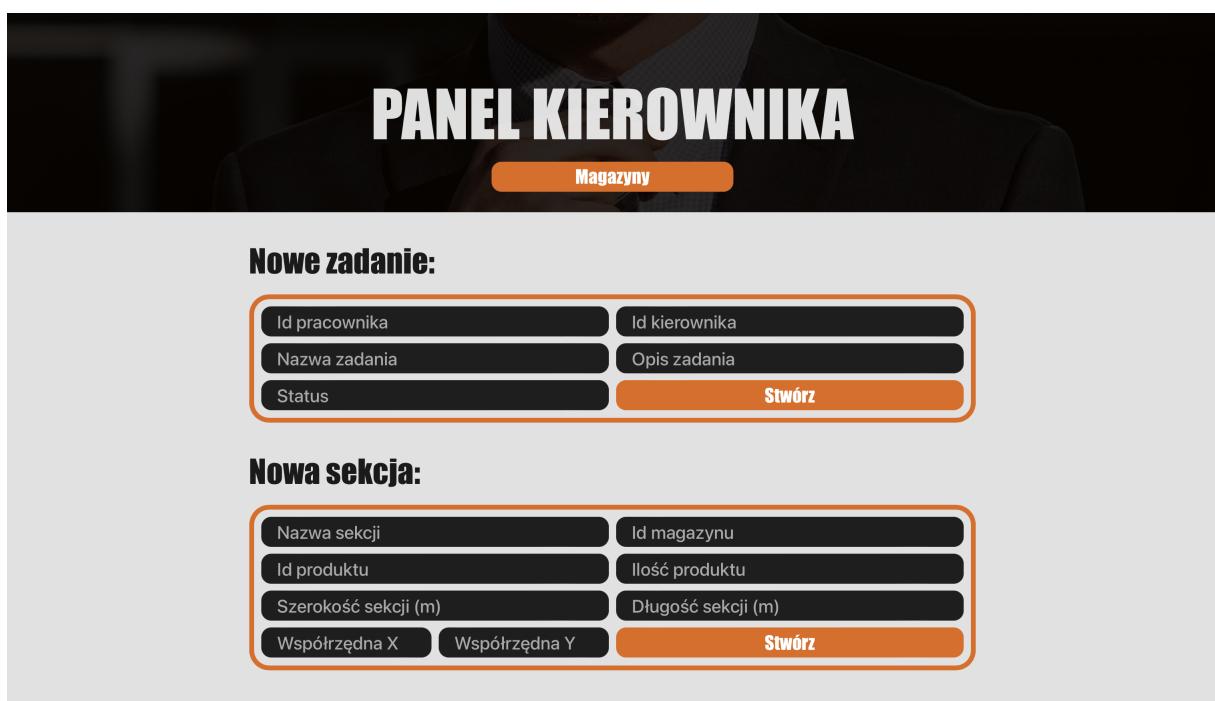
Przy projektowaniu ekranów postawiono na prostotę i intuicyjność ich używania, tak aby wszystkie funkcjonalności były łatwo i szybko dostępne zarówno przez ekran panelu kierownika, jak i pracownika niepełniającego tej funkcji.

Design ekranów ściśle nawiązuje do tematyki projektu - widoczny jest silny motyw związany z budownictwem i magazynowaniem. Szata graficzna, styl i zaproponowana paleta kolorystyczna tworzą spójną oraz estetyczną całość.

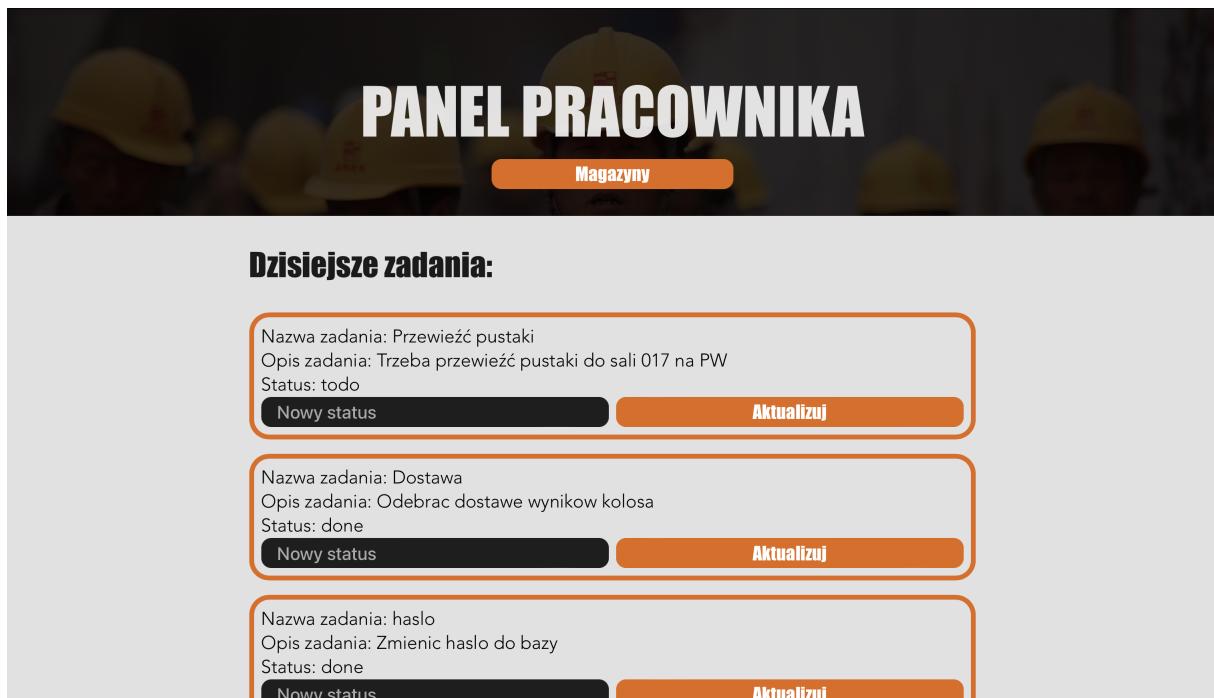
Poniżej zaprezentowano przykładowe ekranы dostępnego z aplikacji:



Ekran startowy aplikacji.



Panel kierownika magazynu.



Panel pracownika magazynu.



Panel przeglądu informacji o magazynach.

## 9 Baza danych

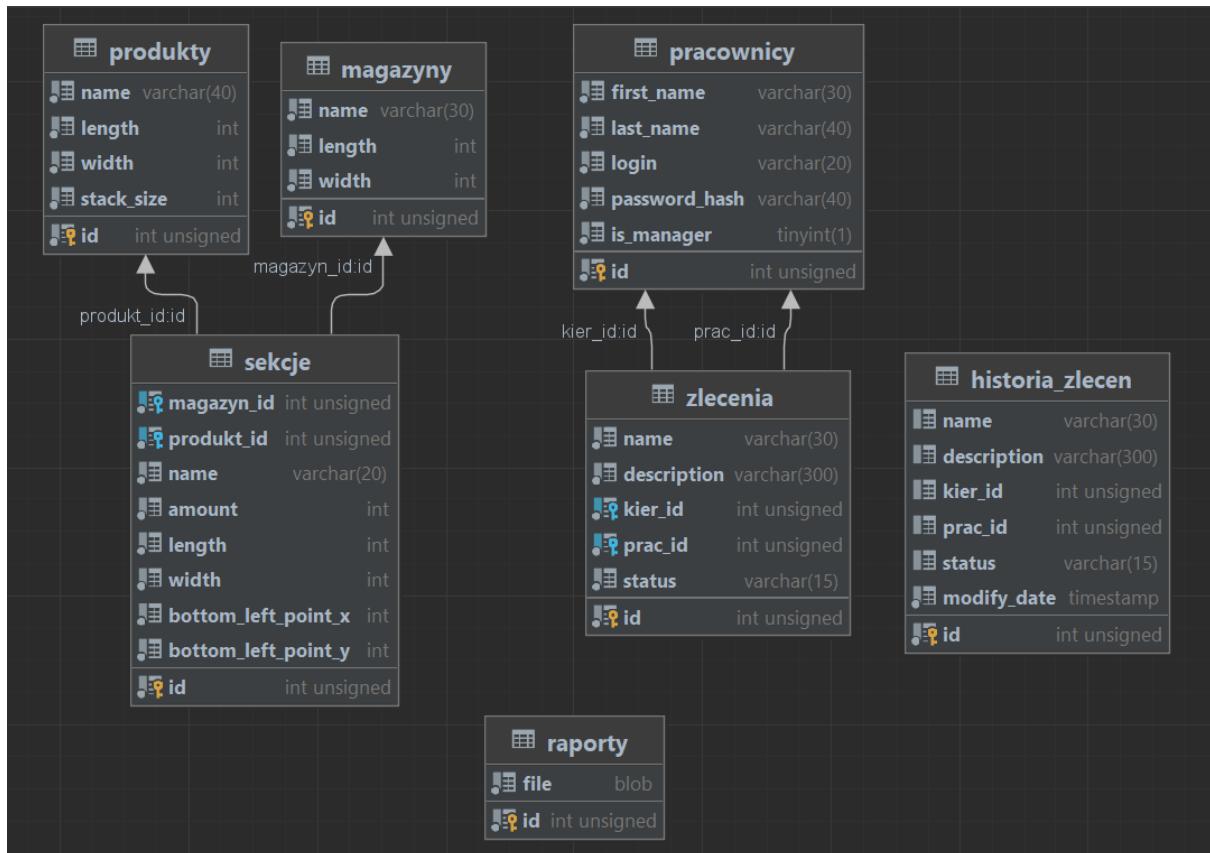
### 9.1 Serwer

Serwerem bazodanowym, którego użyto w projekcie jest serwer MySQL. Został on postawiony na maszynie wirtualnej Azure z systemem Ubuntu. Aby podnieść poziom bezpieczeństwa i jakości dostarczonego rozwiązania skonfigurowano replikę bazy, działającą na innej maszynie dostępnej przez portal Azure.



## 9.2 Diagram ER

Poniżej zaprezentowano diagram relacyjny stworzonej bazy:



Jak widać powyżej, w bazie znajdują się następujące tabele:

1. Magazyny
2. Sekcje
3. Produkty
4. Pracownicy
5. Zlecenia
6. Historia zleceń
7. Raporty

Struktura magazynu opisanego w bazie danych jest następująca:

Magazyn podzielony jest na sekcje, w których znajdują się określone produkty (w każdej z sekcji jeden produkt). Zarówno magazyn jak i sekcja mają swoje wymiary określone w metrach. Dodatkowo,

sekcja posiada współrzędne jednego z wierzchołków tak, aby można było jednoznacznie wyznaczyć jej obszar na mapie magazynu.

Pracownicy magazynu dzielą się na kierowników oraz pozostałych pracowników. Kierownicy mogą wystawiać innym pracownikom zlecenia, w których określają co należy wykonać.

Historia zleceń przetrzymuje historię wykonanych zleceń.

Kierownicy magazynu mogą generować raporty PDF opisujące aktualny stan magazynu. Raporty te, oprócz pobierania na maszynę użytkownika wysyłane są też do bazy danych, gdzie są przechowywane.

## 9.3 Dokładne charakterystyki tabel

### 9.3.1 Magazyny

Tabela *Magazyny* posiada następujące atrybuty:

1. *id* - id magazynu,
2. *name* - nazwa magazynu,
3. *length* - długość magazynu w metrach,
4. *width* - szerokość magazynu w metrach,

### 9.3.2 Sekcje

Tabela *Sekcje* posiada następujące atrybuty:

1. *id* - id sekcji,
2. *magazyn\_id* - id magazynu, w którym znajduje się dana sekcja,
3. *produkt\_id* - id produktu przechowywanego w danej sekcji,
4. *name* - nazwa sekcji,
5. *amount* - liczba sztuk produktu w sekcji,
6. *length* - długość sekcji w metrach,
7. *width* - szerokość sekcji w metrach,
8. *bottom\_left\_point\_x* - X-owa współrzędna lewego dolnego rogu sekcji,
9. *bottom\_left\_point\_y* - Y-owa współrzędna lewego dolnego rogu sekcji.

### 9.3.3 Produkty

Tabela *Produkty* posiada następujące atrybuty:

1. *id* - id produktu,
2. *name* - nazwa produktu,
3. *length* - długość magazynu w metrach,
4. *width* - szerokość magazynu w metrach,
5. *stack\_size* - liczba palet z produktem, które mogą być na siebie nałożone.

### 9.3.4 Pracownicy

Tabela *Pracownicy* posiada następujące atrybuty:

1. *id* - id pracownika,
2. *first\_name* - imię,
3. *last\_name* - nazwisko,
4. *login* - login w aplikacji,
5. *password\_hash* - hash hasła do konta w aplikacji,
6. *is\_manager* - oznaczenie, czy pracownik pełni rolę kierownika.

### 9.3.5 Zlecenia

Tabela *Zlecenia* posiada następujące atrybuty:

1. *id* - id zlecenia,
2. *name* - tytuł,
3. *description* - treść zlecenia,
4. *kier\_id* - id kierownika wystawiającego zlecenie,
5. *prac\_id* - id pracownika, który ma wykonać zlecenie,
6. *status* - status wykonania zlecenia.

### 9.3.6 Historia zleceń

Tabela *Historia\_zleceń* posiada te same atrybuty co tabela *Zlecenia* oraz jeden dodatkowy atrybut:

1. *modify\_date* - data, kiedy zlecenie zostało dodane do historii.

### 9.3.7 Raporty

Tabela *Raporty* posiada następujące atrybuty:

1. *id* - id raportu,
2. *file* - treść raportu w postaci BLOBa.

## 10 Testy

W celu sprawdzenia poprawności działania programu przeprowadzono szereg testów.

Poniżej zaprezentowano statystyki końcowe dotyczące pokrycia kodu testami:

Element ▲	Class, %	Method, %	Line, %
▼ all	100% (7/7)	86% (51/59)	90% (144/159)
> pis.projekt.controller	100% (7/7)	88% (40/45)	81% (61/75)
> pis.projekt.interfaces	100% (0/0)	100% (0/0)	100% (0/0)
> pis.projekt.repository	100% (0/0)	100% (0/0)	100% (0/0)
> pis.projekt.services	100% (7/7)	86% (51/59)	90% (144/159)

Jak widać, pokrycie testami jest zdecydowanie zadowalające. Niedoskonałości w niektórych miejscach wynikają głównie z faktu nietestowania prostych konstruktorów, getterów oraz setterów.

Poniżej znajdują się dokładniejsze informacje na temat etapu testowania.

## 10.1 Testy jednostkowe

W ramach testów jednostkowych przetestowane zostały te funkcjonalności aplikacji, które nie wymagały komunikacji z pozostałymi warstwami.

Wykorzystane narzędzia to AssertJ oraz JUnit.

## 10.2 Testy integracyjne

Testy integracyjne przeprowadzono dla trzech warstw backendu:

1. Warstwa *Repository* - do przetestowania warstwy komunikującej się z bazą danych i pobierającej z niej informacje wykorzystano narzędzie DataJpaTest. Wykorzystuje ono H2 do tworzenia bazy danych w pamięci, dzięki czemu możliwa jest symulacja modeli wykorzystywanych w projekcie. Baza testowa tego typu tworzona jest na potrzebę indywidualnego testu i usuwana wrac z jego zakończeniem.

Za pomocą tego narzędzia sprawdzono poprawność działania poleceń takich jak *SELECT*, czy *INSERT*.

2. Warstwa *Services* - aby móc dokonać testowania warstwy pośredniej pomiędzy kontrolerami, a repozytoriami wymagane było zamockowanie działanie repozytoriów, a także zasilić je danymi, używając *inject mock*.

Mockowanie zachowania repozytoriów było możliwe dzięki użyciu klauzuli *when*. Dzięki temu udało się przeprowadzić testy odizolowane, pozwalające na rzeczywistą obserwację zachowania serwisów.

3. Warstwa *Controllers* - w celu przetestowania warstwy pośredniczącej pomiędzy serwisami, a frontendem należało z kolei zamockować działanie serwisów. Zostało to osiągnięte analogicznie do mockowania repozytoriów - za pomocą klauzuli *when*.

Przy tworzeniu testowych zapytań MockMVC wykrywa, która z zaimplementowanych funkcji powinna zostać wywołana i symuluje jej działanie.

## 11 Deployment oraz uruchomienie aplikacji

Deployment aplikacji - zarówno frontendu, jak i backendu odbywa się za pomocą CI/CD.

Ponieważ dostarczone w ramach projektu rozwiązanie jest aplikacją webową, użytkownik końcowy nie musi specjalnie uruchamiać poszczególnych jej komponentów.

Aby skorzystać z aplikacji należy przez przeglądarkę wejść pod adres:

<http://40.114.226.113/>

Po wykonaniu tej czynności oczom użytkownika ukaże się ekran startowy aplikacji.

## 12 Napotkane problemy i ich rozwiązania

Podczas tworzenia aplikacji napotkano następujące problemy:

- **Przekazywanie czcionki wymaganej do generowania raportów** - problematycznym okazał się wstępnie zaproponowany sposób przekazywania czcionki z systemu Windows do projektu - należało zmienić sposób przekazywania z pliku na przekazywanie jej poprzez input stream.
- **Zbyt mało wydajne maszyny wirtualne dostępne w ramach portalu Azure** - podczas konfigurowania narzędzi z ELK Stack początkowo przyjętą taktyką było skonfigurowanie ich w Dockerze w sieci utworzonej na maszynach wirtualnych. Niestety okazało się, że zasoby wybranych maszyn wirtualnych są niewystarczające, by te dały radę hostować serwisy i umożliwiać ciągłą synchronizację bazy MySQL z jej dokumentową kopią w ElasticSearchu. Zmiana maszyn nie była dostępną opcją, ze względu na fundusze dostępne w ramach licencji studenckiej Azure. Z tego powodu przyjętym rozwiązaniem stało się hostowanie ElasticStacka w chmurze, a synchronizacja danych przeprowadzana zostaje wsadowo z urządzeniem jednego z członków projektu, posiadającego dostęp do serwera MySQL za pomocą skryptów Logstasha i interfejsu JDBC.

- **Poprawne mockowanie elementów do testowania** - wyzwanie sprawiło stworzenie i obsługę narzędzi używanych do testów integracyjnych pisanych w Javie, ponieważ osoba testująca do tej pory nie uskuteczniła mockowania na potrzebę testowania.
- **Dodanie testów do pipeline'a na GitHub'ie** - jedno z dependencies używanych w projekcie Maven'owym nie obsługiwało testów napisanych w JUnit 4. Aby rozwiązać ten problem zmieniono wersję JUnit na niższą.

## 13 Wnioski końcowe

Wszystkie części projektu zostały pomyślnie ukończone, a tematyka i forma zadania okazała się ciekawa i bardzo kształcąca.

Główną przeszkodą podczas prac nad projektem była nieznajomość języka i używanych narzędzi - sprawiało to sporo problemów i zagwostek technicznych, a wykonanie niektórych zadań okazywało się dużo bardziej czasochłonne, niż oczekiwaliśmy. Dobrym przykładem tego zjawiska może być np. implementacja dynamicznego wyświetlania obiektów na stronie.

Komunikacja w zespole przebiegała bardzo dobrze, a każdy z członków grupy dołożył się do powstania produktu końcowego.

Rady i wskazówki mentora grupy były bardzo pomocne i znacząco ułatwiały rozpoczęcie prac nad zadaniem. Podczas realizacji projektu nauczyliśmy się wielu narzędzi, z którymi nie mieliśmy okazji pracować w przeszłości. Jesteśmy pewni, że zdobyta wiedza i doświadczenie zaowocują w dalszym rozwoju na naszej programistycznej ścieżce.