

**UNIwersYTET RZESZOWSKI**  
**WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH**  
**INSTYTUT INFORMATYKI**



*Dominik Kuraś*  
134939

*Informatyka*

*Projekt i implementacja desktopowej aplikacji do zarządzania  
sklepem z wykorzystaniem języka Java i bazy danych MySQL*

Praca projektowa

Praca wykonana pod kierunkiem  
mgr inż. Ewa Żesławska

Rzeszów 2025



## Spis treści

<b>1. Streszczenie</b>	7
1.1. Streszczenie w języku polskim	7
1.2. Summary in English	7
<b>2. Wizja i cele projektu</b>	8
2.1. Kluczowe możliwości aplikacji	8
2.2. Jakość i doświadczenie użytkownika	8
<b>3. Opis struktury projektu</b>	9
3.1. Wykorzystane technologie i narzędzia	9
3.2. Architektura i hierarchia klas	9
3.2.1. Diagram klas UML	10
3.3. Zarządzanie danymi i baza danych	11
3.4. Wymagania systemowe	12
<b>4. Harmonogram i Zarządzanie Projektem</b>	13
4.1. Harmonogram realizacji - Diagram Ganta	13
4.2. Napotkane wyzwania i rozwiązania	14
4.3. System kontroli wersji i repozytorium	14
<b>5. Prezentacja warstwy użytkowej projektu</b>	15
5.1. Ekran Główny i Proces Logowania	15
5.2. Interfejs Klienta	16
5.2.1. Panel Klienta Detalicznego	16
5.2.2. Panel Klienta Hurtowego	17
5.3. Panel Administratora	17
5.3.1. Zarządzanie Klientami i ich szczegóły	18
5.3.2. Zarządzanie Magazynem	19
5.3.3. Zarządzanie Transakcjami	19
<b>6. Testowanie aplikacji</b>	20
6.1. Testowanie manualne	20
6.1.1. Testy scenariuszy logowania i rejestracji	21
6.1.2. Testy procesu zamawiania dla klienta detalicznego	21
6.1.3. Testy specyficznych scenariuszy klienta hurtowego	22
6.1.4. Testy walidacji pól i zabezpieczeń	22
<b>7. Implementacja i Kluczowe Fragmenty Kodu</b>	23
7.1. Wybrane Metody Usprawniające Pracę	23
7.1.1. Metoda <code>createStyledButton</code> - Ujednolicenie Wyglądu Przycisków	23
7.1.2. Metoda <code>getConnection</code> - Efektywne Zarządzanie Połączeniami z Bazą Danych	24
7.2. Inne Istotne Fragmenty Kodu	24

7.2.1. Integracja i Użycie Kalendarza LGoodDatePicker .....	24
<b>8. Podsumowanie</b> .....	25
8.1. Wnioski .....	25
8.2. Propozycje dalszego rozwoju .....	25
<b>Bibliografia</b> .....	25
<b>Spis rysunków</b> .....	27
<b>Spis listingów</b> .....	28
<b>Oświadczenie studenta o samodzielności pracy</b> .....	29

# 1. Streszczenie

## 1.1. Streszczenie w języku polskim

Głównym celem projektu było stworzenie desktopowego systemu do zarządzania sklepem. Aplikacja składa się z dwóch głównych modułów: interfejsu dla klienta (z podziałem na sprzedaż detaliczną i hurtową) oraz panelu dla administratora. System został napisany w języku **Java**, jego interfejs graficzny powstał przy użyciu biblioteki **Swing**, a za przechowywanie danych odpowiada baza **MySQL**. Zastosowanie wzorca projektowego **DAO** pozwoliło na oddzielenie logiki biznesowej od operacji na bazie danych, co ułatwia utrzymanie i rozwój kodu. Klient może przeglądać produkty i finalizować zakupy, a administrator zarządza asortymentem, klientami, transakcjami i stanami magazynowymi.

## 1.2. Summary in English

The main objective of this project was to create a desktop store management system. The application consists of two main modules: a customer interface (with separate retail and wholesale functionalities) and an administrator panel. The system was developed in **Java**, with its graphical user interface built using the **Swing** library, and a **MySQL** database for data storage. The use of the **DAO** design pattern allowed for the separation of business logic from database operations, simplifying code maintenance and scalability. The customer can browse products and complete purchases, while the administrator manages inventory, customers, transactions, and stock levels.

## 2. Wizja i cele projektu

Celem projektu było stworzenie aplikacji desktopowej, która kompleksowo symuluje działanie sklepu. Wizja zakładała budowę narzędzia, które z jednej strony zapewnia klientom (zarówno detalicznym, jak i hurtowym) **prostą i wygodną ścieżkę zakupową**, a z drugiej daje administratorowi **rozbudowane centrum dowodzenia** do zarządzania całym zapleczem sklepu – od produktów i stanów magazynowych, po klientów i historię transakcji. Aplikacja ma stanowić w pełni funkcjonalny ekosystem sprzedażowy.

### 2.1. Kluczowe możliwości aplikacji

Aby zrealizować postawione cele, aplikacja musi oferować szereg przemyślanych funkcji, które razem tworzą spójne doświadczenie dla każdego użytkownika. Każdy, nawet bez logowania, będzie mógł swobodnie przeglądać **katalog produktów**. Gdy użytkownik zdecyduje się na zakupy, system umożliwi mu założenie konta i zalogowanie się lub kontynuację bez zakładania konta. Proces logowania jest kluczowy, ponieważ na jego podstawie aplikacja rozpoznaje, czy ma do czynienia z klientem, czy z administratorem, i dostosuje dostępne opcje. Dla **klienta** przewidziano intuicyjną ścieżkę zakupową: od dodawania produktów do **wirtualnego koszyka**, przez jego modyfikację, aż po sfinalizowanie transakcji w prostym formularzu zamówienia. Dla **administratora**, po zalogowaniu, otworzy się dostęp do rozbudowanego **panelu zarządczego**. Będzie to jego centrum dowodzenia, w którym zyska pełną kontrolę nad asortymentem – będzie mógł dodawać nowe produkty, edytować istniejące i zarządzać stanami magazynowymi. Panel pozwoli również na przeglądanie listy zarejestrowanych klientów oraz monitorowanie wszystkich złożonych zamówień.

### 2.2. Jakość i doświadczenie użytkownika

Poza samymi funkcjami, fundamentalne znaczenie ma to, jak aplikacja będzie działać. Założeniem jest, aby korzystanie z niej było efektywne i bezproblemowe. Priorytetem jest, aby aplikacja była **prosta w obsłudze i responsywna**. Zarówno klient przeglądający ofertę, jak i administrator wprowadzający nowy produkt, powinni czuć, że program działa **płynnie i intuicyjnie**. Każda akcja musi wywoływać natychmiastową, przewidywalną reakcję systemu, bez irytujących opóźnień. Równie ważna jest **stabilność i niezawodność**. Aplikacja musi być przygotowana na nieprzewidziane sytuacje, takie jak chwilowe problemy z dostępem do bazy danych. W takim przypadku system nie może się zawiesić, lecz powinien w **zrozumiały dla użytkownika sposób poinformować go o problemie** i pozwolić na kontynuację pracy, gdy tylko będzie to możliwe. Na koniec, dzięki zastosowaniu technologii Java, aplikacja będzie **uniwersalna**.

## 3. Opis struktury projektu

Ten rozdział opisuje techniczną stronę projektu. Skupiam się w nim na użytych technologiach, architekturze oraz sposobie, w jaki zorganizowane są dane. Rozdział zamyka omówienie kluczowych klas i mechanizmów oraz wymagań niezbędnych do uruchomienia aplikacji.

### 3.1. Wykorzystane technologie i narzędzia

Sercem aplikacji jest język **Java**. To, co użytkownik widzi na ekranie, czyli interfejs graficzny, to zasługa biblioteki **Java Swing**. Wszystkie dane – o produktach, klientach czy zamówieniach – trafiają do bazy danych **MySQL**, a „mostem”, który łączy aplikację z bazą, jest technologia **JDBC**.

Nad całym kodem i historią jego zmian czuwał system kontroli wersji **Git**, a jego centralne repozytorium znajdowało się na platformie **GitHub**. Cały proces programowania odbywał się w środowisku **IntelliJ IDEA**. Aby ułatwić użytkownikowi wybieranie dat w kalendarzu, skorzystałem też z gotowej, zewnętrznej biblioteki **LGoodDatePicker** [1].

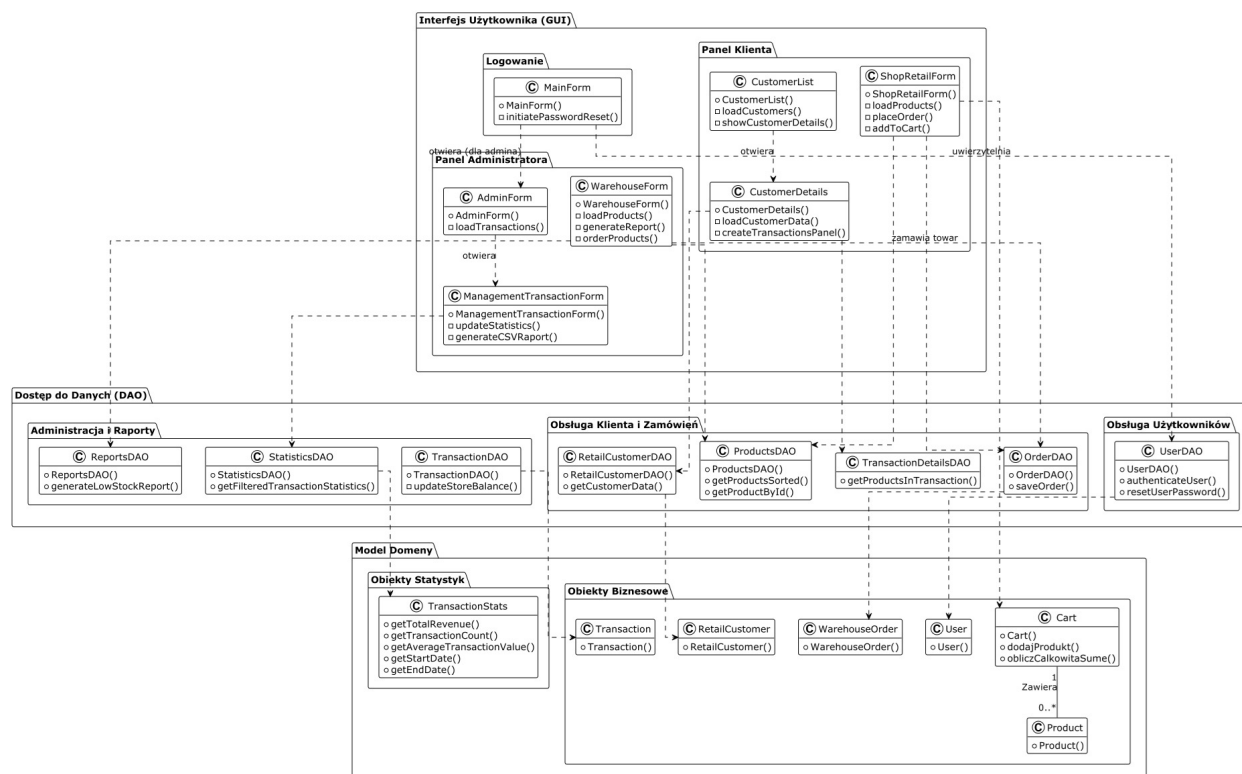
### 3.2. Architektura i hierarchia klas

Aplikacja ma przejrzystą budowę, ponieważ jest podzielona na trzy oddzielne warstwy. Każda z nich ma inne zadanie i nie miesza się z pozostałymi. Dzięki temu łatwiej jest zarządzać kodem i rozwijać program bez obawy, że zmiana w jednym miejscu zepsuje coś w innym.

1. **Warstwa Prezentacji (View):** To po prostu wszystko to, co użytkownik widzi na ekranie i z czym może wejść w interakcję. Są to więc wszystkie okna, formularze i przyciski aplikacji, zdefiniowane w klasach takich jak `MainForm.java`, `AdminForm.java` czy `ShopRetailForm.java`.
2. **Warstwa Logiki Biznesowej (Model):** To prawdziwe serce aplikacji – tu znajduje się cała „inteligencja” programu. Klasy w tej warstwie (np. `Product.java`, `Cart.java`, `Customer.java`) odwzorowują obiekty ze świata rzeczywistego i zawierają logikę ich działania, np. jak obliczyć sumę w koszyku czy jak dodać produkt.
3. **Warstwa Dostępu do Danych (Data Access Layer):** Ta warstwa to pośrednik, który zajmuje się wyłącznie komunikacją z bazą danych. Zastosowałem tu wzorzec **DAO**, aby zebrać w jednym miejscu cały kod odpowiedzialny za operacje na bazie. Dzięki temu reszta aplikacji nie musi przejmować się tym, jak dane są zapisywane i odczytywane, co upraszcza kod i czyni go bezpieczniejszym. Kluczowe metody, jak `OrderDAO.saveOrder()`, dbają o to, by operacje na bazie danych były bezpieczne i spójne.

### 3.2.1. Diagram klas UML

Diagram klas (Rys. 3.1) to wizualna „mapa” projektu. Musiałem ją jednak znacznie uprościć. W aplikacji jest tak wiele klas i powiązań, że pokazanie wszystkich na raz stworzyłoby chaos, a nie czytelny obraz. Mimo to, diagram dobrze pokazuje podział na warstwy i najważniejsze zależności między nimi. Do jego wygenerowania użyłem narzędzia **PlantUML**, działającego jako wtyczka w IntelliJ IDEA.



**Rys. 3.1.** Uproszczony diagram klas UML projektu.

Źródło: Opracowanie własne przy użyciu PlantUML





### 3.4. Wymagania systemowe

Aby uruchomić projekt na własnym komputerze i móc go rozwijać, potrzebne jest kilka darmowych narzędzi. Poniższa lista wyjaśnia, co i dlaczego należy zainstalować.

- **Pakiet XAMPP:** To gotowy zestaw narzędzi, który w prosty sposób instaluje serwer bazy danych **MariaDB** (kompatybilny z MySQL). Jest on niezbędny, aby aplikacja miała gdzie przechowywać swoje dane.
- **Java Development Kit (JDK):** Jest to środowisko niezbędne do kompilowania i uruchamiania aplikacji napisanych w języku Java.
- **Środowisko IntelliJ IDEA:** To zaawansowany edytor kodu, w którym projekt był tworzony. Ułatwia on pracę z kodem, kompilację i uruchamianie programu.

Aktualne wymagania sprzętowe dla tych programów można znaleźć na ich oficjalnych stronach:

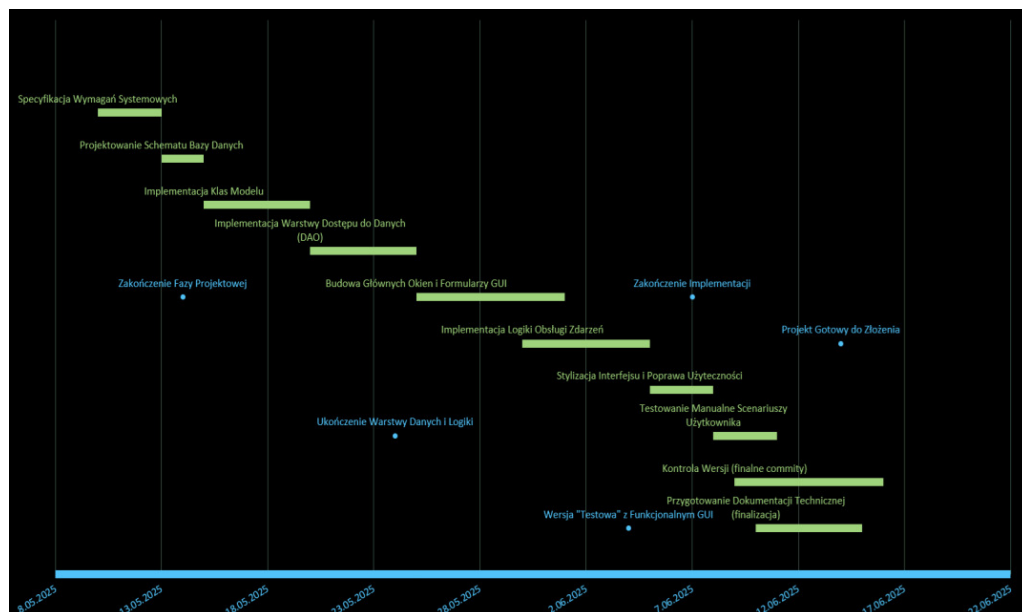
- **XAMPP:** <https://www.apachefriends.org/download.html>
- **IntelliJ IDEA:** <https://www.jetbrains.com/help/idea/installation-guide.html#requirements>

## 4. Harmonogram i Zarządzanie Projektem

### 4.1. Harmonogram realizacji - Diagram Ganta

Proces tworzenia aplikacji został podzielony na kilka kluczowych etapów, których realizację w czasie ilustruje diagram Gantta (Rys. 4.1). Prace nad projektem rozłożono w sposób umożliwiający systematyczny postęp i iteracyjne dostarczanie funkcjonalności.

- **Faza 1: Analiza i Projektowanie (ok. 15% czasu pracy):** Ten początkowy etap obejmował zdefiniowanie wymagań funkcjonalnych i нефункциональных, zaprojektowanie trójwarstwowej architektury systemu oraz stworzenie schematu relacyjnej bazy danych w MySQL.
- **Faza 2: Implementacja Warstwy Danych i Logiki Biznesowej (ok. 30% czasu pracy):** Największą część tej fazy zajęło stworzenie warstwy dostępu do danych (DAO) oraz klas modelu (np. `Product`, `Customer`), które stanowią fundament komunikacji z bazą danych.
- **Faza 3: Implementacja Interfejsu Graficznego Użytkownika (ok. 40% czasu pracy):** Był to najbardziej czasochłonny etap, obejmujący budowę wszystkich okien i paneli aplikacji w technologii Java Swing (np. `AdminForm`, `ShopRetailForm`), implementację logiki obsługi zdarzeń (np. kliknięcia przycisków) oraz stylizację komponentów w celu zapewnienia spójnego i intuicyjnego interfejsu.
- **Faza 4: Testowanie i Dokumentacja (ok. 15% czasu pracy):** Ostatni etap poświęcono na manualne testy kluczowych scenariuszy użytkowania, takich jak proces logowania, składanie zamówień czy zarządzanie danymi przez administratora. Równolegle tworzono niniejszą dokumentację techniczną w systemie  $\text{\LaTeX}$ .



Rys. 4.1. Harmonogram realizacji projektu (Diagram Ganta).

Źródło: Opracowanie własne na podstawie szablonu Microsoft Gantt Chart Template.

## 4.2. Napotkane wyzwania i rozwiązania

W trakcie prac nad projektem nie obyło się bez pewnych trudności, które wymagały kreatywnego podejścia. Jednym z wyzwań była implementacja intuicyjnego mechanizmu wyboru daty w panelu administratora. Początkowe rozwiązanie, oparte na standardowych komponentach biblioteki Swing, takich jak `JComboBox` dla miesiąca i roku oraz `JButton` dla dni, okazało się nieestetyczne i mało wygodne dla użytkownika.

W poszukiwaniu lepszej alternatywy przeprowadzono analizę dostępnych rozwiązań, w tym bibliotek udostępnianych na platformie GitHub, aby znaleźć gotowy komponent kalendarza. Ostatecznie wybór padł na bibliotekę `LGoodDatePicker` [1]. Jej integracja z projektem była strzałem w dziesiątkę. Biblioteka dostarczyła atrakcyjny wizualnie i prosty w obsłudze kalendarz, co znacząco poprawiło użyteczność aplikacji w miejscach wymagających operowania na datach, łącząc funkcjonalność z nowoczesnym wyglądem.

## 4.3. System kontroli wersji i repozytorium

Zarządzanie kodem źródłowym projektu opierało się na systemie kontroli wersji **Git**, który jest standardem w nowoczesnym wytwarzaniu oprogramowania. Wszystkie operacje, takie jak tworzenie commitów, zarządzanie gałęziami (branching) czy scalanie zmian (merging), były wykonywane przy użyciu zintegrowanego klienta Git w środowisku programistycznym **IntelliJ IDEA**.

Jako centralne, zdalne repozytorium kodu wykorzystano platformę **GitHub**. Cały projekt, wraz z historią zmian, jest publicznie dostępny pod adresem:

*[https://github.com/Deskalisko/Projekt\\_JAVA](https://github.com/Deskalisko/Projekt_JAVA)*

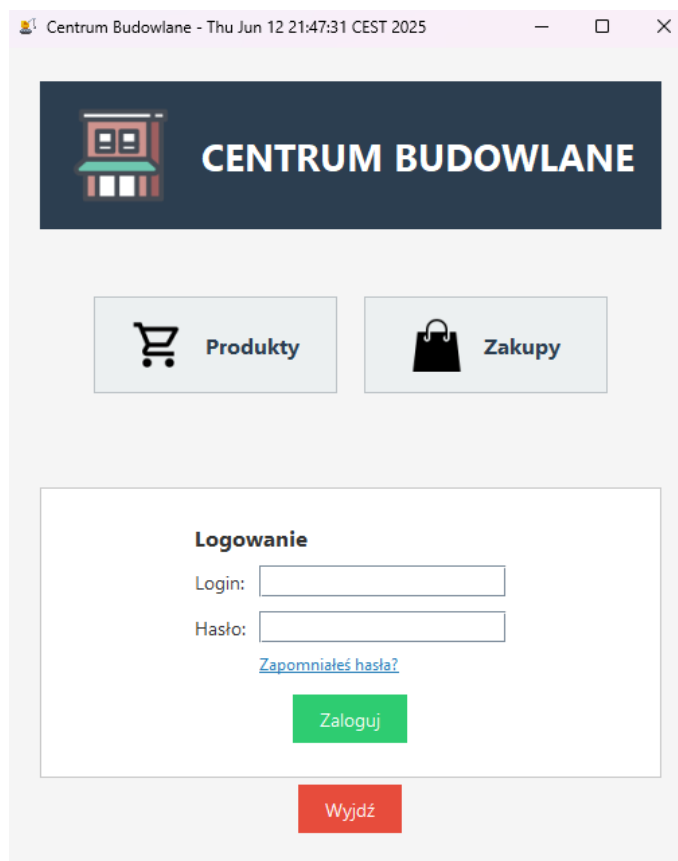
Wykorzystanie systemu kontroli wersji pozwoliło na systematyczne śledzenie postępów, bezpieczne eksperymentowanie z nowymi funkcjonalnościami w osobnych gałęziach oraz zapewniło stałą kopię zapasową projektu.

## 5. Prezentacja warstwy użytkowej projektu

W tym rozdziale przyjrzymy się aplikacji od strony użytkownika. Omówię kluczowe widoki i funkcje interfejsu graficznego (GUI) – od ekranu głównego, przez panele zakupowe, aż po moduły administratora.

### 5.1. Ekran Główny i Proces Logowania

Punktem startowym aplikacji jest ekran główny (`MainForm.java`), przedstawiony na Rys. 5.1, który pełni rolę centrum nawigacyjnego. Z jego poziomu użytkownik może przejść do przeglądania listy produktów, rozpocząć proces zakupowy lub zalogować się do systemu. Panel logowania jest kluczowym elementem, który na podstawie wprowadzonych danych uwierzytelnia użytkownika i kieruje go do odpowiedniego modułu aplikacji. Zaimplementowano również funkcjonalność resetowania hasła, która po weryfikacji tożsamości użytkownika pozwala na ustawienie nowego hasła.



**Rys. 5.1.** Główny ekran aplikacji z widocznym panelem logowania.

Źródło: Opracowanie własne

## 5.2. Interfejs Klienta

Interfejs przeznaczony dla klienta został zaprojektowany z myślą o intuicyjności i wygodzie procesu zakupowego. Mimo że logika biznesowa i wygląd paneli dla klienta detalicznego (`ShopRetailForm.java`) i hurtowego (`ShopWholeSaleForm.java`) są do siebie bardzo zbliżone, istnieją między nimi kluczowe różnice.

### 5.2.1. Panel Klienta Detalicznego

Główny widok sklepu dla klienta detalicznego (Rys. 5.2) składa się z trzech głównych sekcji: panelu danych klienta, listy dostępnych produktów oraz panelu koszyka. Klient może swobodnie przeglądać produkty, dodawać je do koszyka za pomocą przycisku i dedykowanego pola 'JSpinner' do określania ilości, a następnie sfinalizować zamówienie. Kluczową cechą tego modułu jest to, że klient nie musi posiadać konta w systemie. Dane do wysyłki podawane są jednorazowo w oknie dialogowym 'AddEditRetailCustomer', które jest wywoływane przed złożeniem zamówienia.

ID	NAZWA	CENA	ILOSĆ	KATEGORIA	TYP
19	Pompa wodna	799.99 zł	25	Instalacje i hydraulika	elektronika
20	Filtr do wody	129.99 zł	59	Instalacje i hydraulika	sprzet
21	Młotek stolarski	59.99 zł	49	Sprzęt ręczny	sprzet
22	Miara zwijana 5m	19.99 zł	94	Sprzęt ręczny	sprzet
23	Śrubokręt krzyżowy	24.99 zł	76	Sprzęt ręczny	sprzet
24	Wkrętarka elektryczna	249.99 zł	23	Elektronika	elektronika
25	Farba akrylowa 10L	99.99 zł	39	Materiały budowlane	materiały
26	Spodnie robocze	129.99 zł	32	Odzież ochronna	odzież
27	Koszulka odblaskowa	39.99 zł	91	Odzież ochronna	odzież
28	Multimaster odblaskowy	79.00 zł	18	Elektronika	elektronika

Produkt	Ilość	Cena jednostkowa	Łączna cena
Młot udarowy	2	599.99 zł	1199.98 zł
Piła tarczowa	1	749.99 zł	749.99 zł
Młotek stolarski	3	59.99 zł	179.97 zł
Łączna suma:			2129.94 zł

Rys. 5.2. Widok panelu zakupowego dla klienta detalicznego.

Źródło: Opracowanie własne

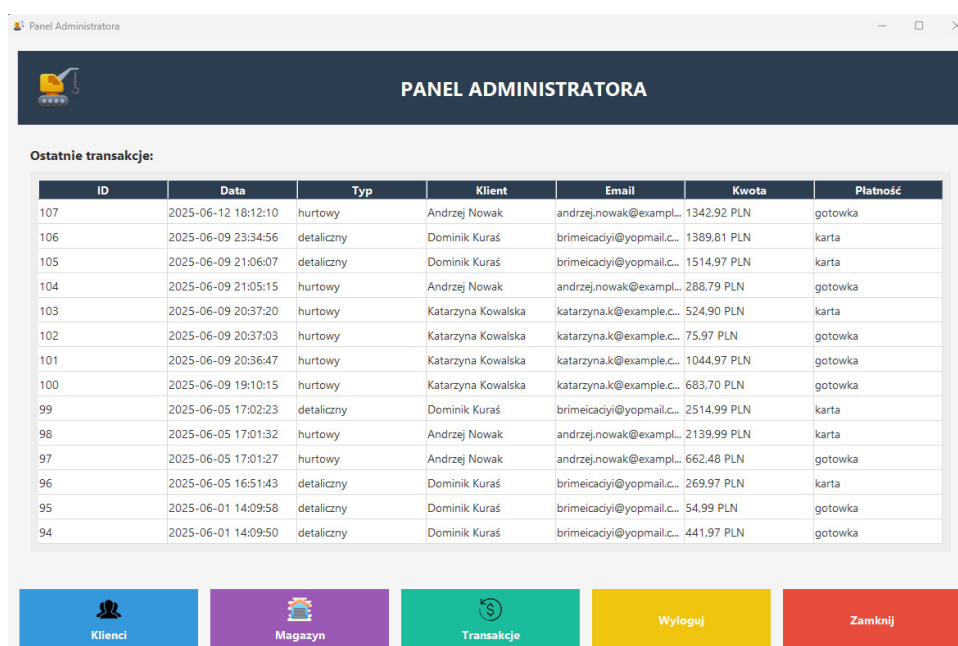
### 5.2.2. Panel Klienta Hurtowego

Panel klienta hurtowego (`ShopWholeSaleForm.java`) jest wizualnie niemal identyczny z panelem detalicznym, jednak dostosowany do specyfiki klienta biznesowego. Główne różnice to:

- **Ceny hurtowe:** Tabela produktów automatycznie ładuje niższe ceny hurtowe pobierane z bazy danych.
- **Stałe dane klienta:** Dane klienta, takie jak nazwa firmy czy NIP, są ładowane automatycznie po zalogowaniu i wyświetlane w górnej części panelu, co eliminuje potrzebę ich każdorazowego wprowadzania. Klient ma możliwość edycji swoich danych oraz zmiany hasła.
- **Warunki zamówienia:** System waliduje, czy zamówienie hurtowe spełnia minimalne progi, takie jak łączna liczba produktów w koszyku (minimum 3) oraz minimalna wartość zamówienia (50 zł).

### 5.3. Panel Administratora

Panel administratora (`AdminForm.java`) stanowi centrum dowodzenia aplikacją. Jest to główny pulpit, z którego administrator ma bezpośredni dostęp do wszystkich kluczowych modułów zarządczych. Został zaprojektowany w sposób modułowy, aby zapewnić szybki i intuicyjny dostęp do poszczególnych funkcjonalności. W centralnej części panelu wyświetlana jest tabela z listą ostatnich transakcji, co pozwala na bieżące monitorowanie aktywności w sklepie.



**PANEL ADMINISTRATORA**

Ostatnie transakcje:

ID	Data	Typ	Klient	Email	Kwota	Płatność
107	2025-06-12 18:12:10	hurtowy	Andrzej Nowak	andrzej.nowak@exampl...	1342.92 PLN	gotowka
106	2025-06-09 23:34:56	detaliczny	Dominik Kuraś	brimeicadiyi@yopmail.c...	1389.81 PLN	karta
105	2025-06-09 21:06:07	detaliczny	Dominik Kuraś	brimeicadiyi@yopmail.c...	1514.97 PLN	karta
104	2025-06-09 21:05:15	hurtowy	Andrzej Nowak	andrzej.nowak@exampl...	288.79 PLN	gotowka
103	2025-06-09 20:37:20	hurtowy	Katarzyna Kowalska	katarzyna.k@example.c...	524.90 PLN	karta
102	2025-06-09 20:37:03	hurtowy	Katarzyna Kowalska	katarzyna.k@example.c...	75.97 PLN	gotowka
101	2025-06-09 20:36:47	hurtowy	Katarzyna Kowalska	katarzyna.k@example.c...	1044.97 PLN	gotowka
100	2025-06-09 19:10:15	hurtowy	Katarzyna Kowalska	katarzyna.k@example.c...	683.70 PLN	gotowka
99	2025-06-05 17:02:23	detaliczny	Dominik Kuraś	brimeicadiyi@yopmail.c...	2514.99 PLN	karta
98	2025-06-05 17:01:32	hurtowy	Andrzej Nowak	andrzej.nowak@exampl...	2139.99 PLN	karta
97	2025-06-05 17:01:27	hurtowy	Andrzej Nowak	andrzej.nowak@exampl...	662.48 PLN	gotowka
96	2025-06-05 16:51:43	detaliczny	Dominik Kuraś	brimeicadiyi@yopmail.c...	269.97 PLN	karta
95	2025-06-01 14:09:58	detaliczny	Dominik Kuraś	brimeicadiyi@yopmail.c...	54.99 PLN	gotowka
94	2025-06-01 14:09:50	detaliczny	Dominik Kuraś	brimeicadiyi@yopmail.c...	441.97 PLN	gotowka

Buttons: Klienci, Magazyn, Transakcje, Wyloguj, Zamknij

Rys. 5.3. Główny widok panelu administratora.

Źródło: Opracowanie własne

Główne moduły, dostępne za pomocą dedykowanych przycisków, to:

- **Zarządzanie Klientami** (`CustomerList.java`)
- **Zarządzanie Magazynem** (`WarehouseForm.java`)
- **Zarządzanie Transakcjami** (`ManagementTransactionForm.java`)

Poniżej opisano szczegółowo działanie każdego z tych modułów.

### 5.3.1. Zarządzanie Klientami i ich szczegóły

Moduł `CustomerList.java` pozwala na przeglądanie, dodawanie (tylko hurtowych), edytowanie oraz usuwanie klientów. Za pomocą przełączników 'JRadioButton' administrator może płynnie przełączać się między listą klientów detalicznych a hurtowych, a dane w tabeli są dynamicznie odświeżane.

Kluczową funkcjonalnością tego panelu jest możliwość wglądu w szczegółowe dane klienta po naciśnięciu przycisku "Szczegóły". Otwiera to nowe okno (`CustomerDetails.java`), które prezentuje kompleksowe informacje na temat wybranego klienta w trzech osobnych zakładkach:

- **Informacje ogólne:** Wyświetlane są tu podstawowe dane, takie jak imię i nazwisko, typ klienta (detaliczny/hurtowy), łączna suma zakupów, adres oraz dane kontaktowe (telefon, e-mail, a w przypadku klienta hurtowego również NIP i nazwa firmy).
- **Historia transakcji:** Tabela zawierająca listę wszystkich transakcji dokonanych przez danego klienta, wraz z datą, kwotą, typem i listą zakupionych produktów.
- **Statystyki produktów:** Tabela grupująca wszystkie produkty zakupione przez klienta, przedstawiająca sumaryczną ilość oraz łączną wartość dla każdego z nich.

**Szczegóły klienta**

**SZCZEGÓŁY KLIENTA**

Informacje Transakcje Statystyki produktów

**Dane klienta:**

Klient: Katarzyna Kowalska

Typ: Hurtowy

Suma zakupów: 59739,48 zł

Adres: Gdańsk, ul. Morska 8

Telefon: 987654321, Email: katarzyna.k@example.com, NIP: 9876543210, Firma: Kowalska Instalacje

Informacje Transakcje Statystyki produktów

ID	Data	Kwota	Typ	Płatność	Produkty
62	2025-05-02 09:30:0...	7499,94 zł	hurtowy	karta	Płyta gipsowo-karto...
100	2025-06-09 19:10:1...	683,70 zł	hurtowy	gotowka	Laserowy miernik odl...
101	2025-06-09 20:36:4...	1044,97 zł	hurtowy	gotowka	Młot udarowy, Laser...
102	2025-06-09 20:37:0...	75,97 zł	hurtowy	gotowka	Miara zwijana 5m
103	2025-06-09 20:37:2...	524,90 zł	hurtowy	karta	Wełna mineralna, Ko...

Informacje Transakcje Statystyki produktów

ID Produktu	Nazwa	Ilość	Wartość
16	Płyta gipsowo-kartonowa	100	3999,00 zł
15	Gips szpachlowy	50	1249,50 zł
22	Miara zwijana 5m	5	84,95 zł
17	Rura PVC 2m	4	84,96 zł
1	Kask ochronny	2	135,98 zł
31	Wełna mineralna	2	322,98 zł
10	Laserowy miernik odległości	2	819,98 zł

Użytkownik: ADMINISTRATOR | Data: Thu Jun 12 21:55:20 CEST 2025

**Rys. 5.4.** Panel szczegółowych informacji o kliencie.

Źródło: Opracowanie własne

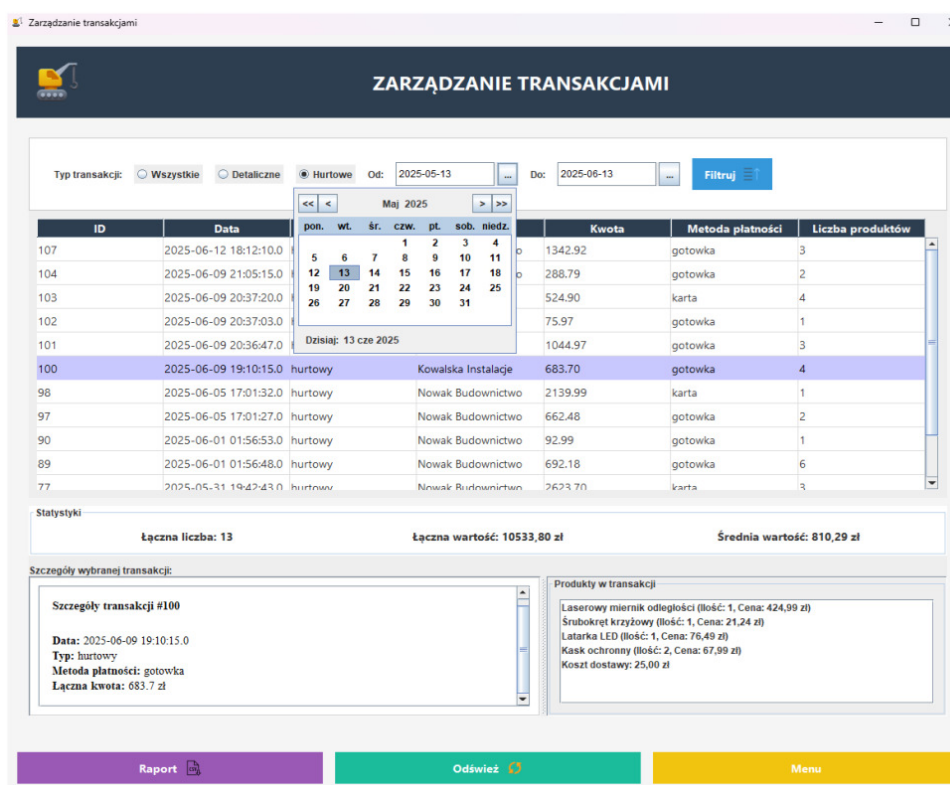


### 5.3.2. Zarządzanie Magazynem

Widok magazynu (`WarehouseForm.java`) jest interfejsem bardzo zbliżonym wizualnie do panelu zarządzania klientami, jednak w pełni dostosowanym do operacji na produktach. Udostępnia narzędzia do pełnej kontroli nad asortymentem, pozwalając administratorowi na przeglądanie stanów magazynowych, sortowanie i filtrowanie danych (np. wyświetlając tylko produkty, których ilość jest niska), a także edytowanie ilości sztuk danego produktu. Zaimplementowano tu również funkcję składania zamówień do dostawców (`createWarehouseOrder`) oraz generowania raportów o niskich stanach magazynowych w formacie CSV. W panelu wyświetlane jest także aktualne saldo finansowe sklepu, które jest modyfikowane przy każdym zamówieniu magazynowym.

### 5.3.3. Zarządzanie Transakcjami

Ostatni moduł, `ManagementTransactionForm.java`, służy do przeglądania i analizy wszystkich transakcji w systemie. Administrator może filtrować transakcje według typu (detaliczne, hurtowe) oraz zakresu dat, korzystając z zintegrowanego komponentu kalendarza `LGoodDatePicker` [1]. Po wybraniu transakcji z tabeli, w dedykowanym panelu wyświetlane są jej szczegółowe dane, w tym lista zakupionych produktów. Panel ten umożliwia również generowanie raportów CSV z przefiltrowanych danych.



Rys. 5.5. Widok panelu zarządzania transakcjami z zintegrowanym kalendarzem.

Źródło: Opracowanie własne

## **6. Testowanie aplikacji**

W niniejszym rozdziale przedstawiono proces testowania aplikacji, zarówno w aspekcie manualnym. Testowanie ma na celu weryfikację poprawności działania wszystkich funkcjonalności systemu, sprawdzenie jego stabilności oraz zapewnienie pozytywnych doświadczeń użytkownika.

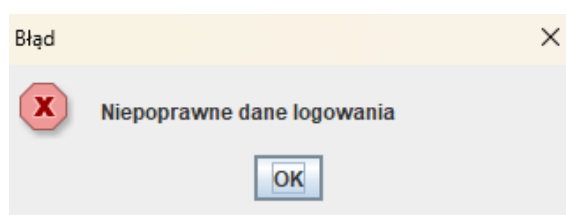
### **6.1. Testowanie manualne**

Testowanie manualne stanowi kluczowy etap w zapewnieniu jakości aplikacji, zwłaszcza w przypadku interfejsów graficznych (GUI). Polega ono na interakcji z programem z perspektywy końcowego użytkownika, pozwalając na wykrycie problemów z użytecznością, wizualną spójnością oraz błędów, które mogą być trudne do zidentyfikowania za pomocą testów automatycznych. Poniżej przedstawiono wybrane scenariusze testowe, które zostały przeprowadzone manualnie.

### 6.1.1. Testy scenariuszy logowania i rejestracji

Kluczowym elementem każdego systemu jest bezpieczne i poprawne zarządzanie kontami użytkowników. Manualne testy tej sekcji obejmowały:

- Pomyślne logowanie dla istniejącego użytkownika (klienta detalicznego i administratora).
- Logowanie z błędnymi danymi (niepoprawna nazwa użytkownika, niepoprawne hasło).
- Testowanie walidacji pól formularza logowania (np. puste pola).
- Próba rejestracji użytkownika z istniejącą nazwą użytkownika lub adresem e-mail.
- Weryfikacja wyświetlanych komunikatów o błędach. Przykład takiego komunikatu, który informuje o błędnie wprowadzonych danych logowania, widoczny jest na Rysunku 6.1.



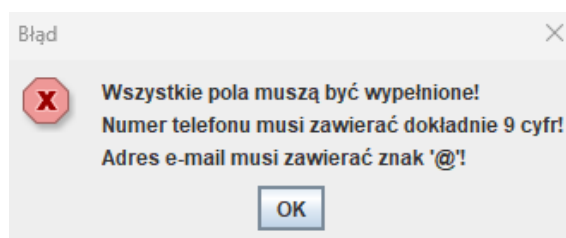
**Rys. 6.1.** Komunikat o błędnym wprowadzeniu danych logowania.

Źródło: Opracowanie własne

### 6.1.2. Testy procesu zamawiania dla klienta detalicznego

Szczególą uwagę zwrócono na poprawność procesu składania zamówienia przez klienta detalicznego. Obejmowało to:

- Dodawanie różnych produktów do koszyka i weryfikacja poprawności sumy.
- Usuwanie produktów z koszyka i zmiana ich ilości.
- Finalizacja zamówienia z poprawnymi danymi klienta (imię, nazwisko, adres, telefon, e-mail).
- Testowanie walidacji pól formularza zamówienia (np. puste pola, niepoprawny format e-maila/telefonu). Na Rysunku 6.2 przedstawiono przykład sytuacji, w której dane są niekompletne lub zawierają nieprawidłowe dane, co skutkuje wyświetleniem błędu.



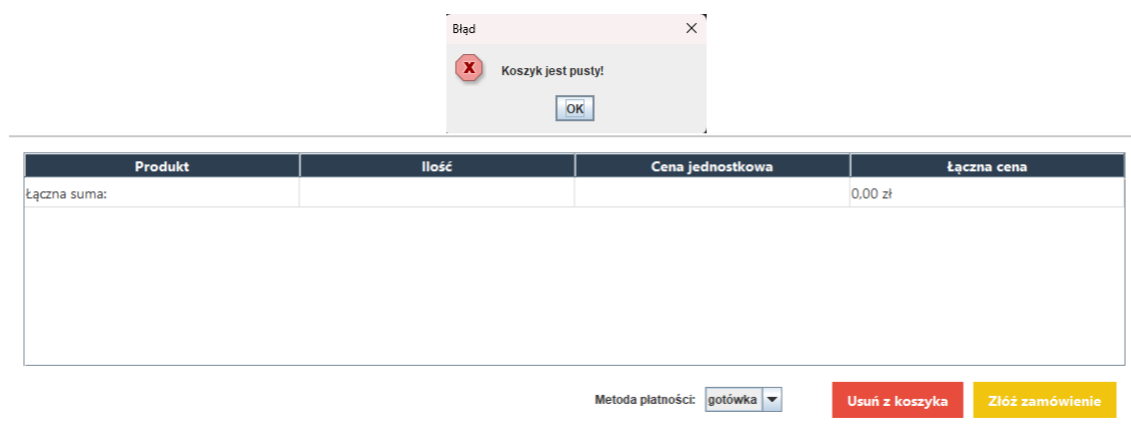
**Rys. 6.2.** Przykład błędnego wprowadzenia danych przez klienta detalicznego.

Źródło: Opracowanie własne

### 6.1.3. Testy specyficznych scenariuszy klienta hurtowego

Dla klienta hurtowego przeprowadzono dodatkowe testy, uwzględniające specyfikę jego uprawnień:

- Sprawdzenie, czy ceny produktów są wyświetlane jako ceny hurtowe.
- Próba złożenia zamówienia z pustym koszykiem. Komunikat informujący o tej sytuacji jest widoczny na Rysunku 6.3.
- Testowanie złożenia zamówienia, które nie spełnia minimalnego wymogu dla zamówień hurtowych (jeśli taki istnieje w aplikacji).
- Walidacja, czy system odpowiednio reaguje na próby zamówienia zbyt małej ilości produktów.



Rys. 6.3. Komunikat informujący o próbie złożenia zamówienia z pustym koszykiem.

Źródło: Opracowanie własne

### 6.1.4. Testy walidacji pól i zabezpieczeń

W aplikacji zaimplementowano szereg zabezpieczeń i walidacji, które zostały dokładnie przetestowane manualnie. Szczególną uwagę zwrócono na:

- Sprawdzenie, czy pola przeznaczone na wartości liczbowe (np. ilość produktów, cena) prawidłowo odrzucają próby wprowadzenia tekstu i akceptują tylko cyfry.
- Weryfikację formatu wprowadzanych danych, takich jak adresy e-mail (muszą zawierać znak '@') oraz numery telefonów (muszą składać się z 9 cyfr).
- Przetestowanie mechanizmu zmiany hasła, w którym sprawdzono, czy system wymusza identyczność haseł w polach "nowe hasło" i "powtórz nowe hasło". W przypadku niezgodności, aplikacja wyświetlała odpowiedni komunikat o błędzie.
- Dokładne przetestowanie komponentu wyboru daty, LGoodDatePicker [1]. Weryfikacja objęła otwieranie i zamykanie kalendarza, wybór daty, ręczne wprowadzanie oraz obsługę błędnych formatów.
- Symulację utraty połączenia z bazą danych, aby sprawdzić, czy aplikacja w takiej sytuacji nie ulega awarii, lecz wyświetla użytkownikowi zrozumiały komunikat o problemie.

## 7. Implementacja i Kluczowe Fragmenty Kodu

### 7.1. Wybrane Metody Usprawniające Pracę

#### 7.1.1. Metoda `createStyledButton` - Ujednolicenie Wyglądu Przycisków

Metoda `'createStyledButton'` jest przykładem dobrej praktyki programistycznej, pozwalającej na tworzenie spójnych stylistycznie przycisków w interfejsie użytkownika. Dzięki niej unika się powielania kodu odpowiedzialnego za ustawianie czcionki, kolorów, ramek i innych właściwości wizualnych dla każdego przycisku z osobna. Znacząco przyspieszyło to budowanie interfejsu graficznego, co widać na przykładzie przedstawionym w Listing 7.1.

**Listing 7.1.** Metoda `createStyledButton`

```
1 private JButton createStyledButton(String text, String iconPath) {
2     ImageIcon icon = new ImageIcon(getClass().getResource(iconPath));
3     Image scaledIcon = icon.getImage().getScaledInstance(40, 40, Image.SCALE_SMOOTH);
4     JButton button = new JButton(text, new ImageIcon(scaledIcon));
5
6     button.setFont(new Font("Segoe UI", Font.BOLD, 16));
7     button.setForeground(new Color(44, 62, 80));
8     button.setBackground(new Color(236, 240, 241));
9     button.setBorder(BorderFactory.createCompoundBorder(
10         BorderFactory.createLineBorder(new Color(189, 195, 199)),
11         BorderFactory.createEmptyBorder(15, 15, 15, 15)));
12     button.setFocusPainted(false);
13     return button;
14 }
```

### 7.1.2. Metoda `getConnection` - Efektywne Zarządzanie Połączeniami z Baza Danych

Zarządzanie połączeniami z bazą danych jest kluczowe dla wydajności i niezawodności aplikacji. Metoda `getConnection` z klasy `DatabaseConnection` (Listing 7.2) zapewnia scentralizowany i prosty sposób na uzyskanie aktywnego połączenia z bazą danych MySQL. Jej wykorzystanie znacznie uprościło operacje na danych, minimalizując ryzyko błędów związanych z otwieraniem i zamykaniem połączeń.

**Listing 7.2.** Metoda `getConnection`

```
1 public class DatabaseConnection {
2     private static final String URL= "jdbc:mysql://localhost:3306/sklep_db";
3     private static final String USER = "root";
4     private static final String PASSWORD = "";
5     public static Connection getConnection() throws SQLException {
6         return DriverManager.getConnection(URL, USER, PASSWORD);
7     }
8 }
9 }
```

## 7.2. Inne Istotne Fragmenty Kodu

### 7.2.1. Integracja i Użycie Kalendarza `LGoodDatePicker`

Aby ułatwić użytkownikowi wybór dat, w projekcie zintegrowano zewnętrzną bibliotekę `LGoodDatePicker` [1]. Komponent ten jest kluczowy w panelu zarządzania transakcjami, gdzie pozwala administratorowi na intuicyjne filtrowanie wyników według zadanego okresu. Jego konfigurację w projekcie przedstawia Listing 7.3.

Zastosowanie gotowego kalendarza wyeliminowało konieczność ręcznego wprowadzania dat i ryzyko błędów formatowania, zapewniając spójny wygląd i wygodę obsługi w całej aplikacji. Finalny wygląd panelu z zaimplementowanym kalendarzem przedstawiono na Rys. 5.5.

**Listing 7.3.** Metoda tworząca i konfiguruje komponent `DatePicker`.

```
1 import com.toedter.calendar.JDateChooser;
2 import javax.swing.*;
3 import java.awt.*;
4 import java.text.SimpleDateFormat;
5 import java.util.Date;
6
7 private JDateChooser createDateChooser() {
8     JDateChooser dateChooser = new JDateChooser();
9     dateChooser.setDateFormatString("yyyy-MM-dd");
10    dateChooser.setFont(new Font("Segoe UI", Font.PLAIN, 14));
11    dateChooser.setPreferredSize(new Dimension(150, 30));
12    dateChooser.setDate(new Date());
13    return dateChooser;
14 }
```

## 8. Podsumowanie

### 8.1. Wnioski

Zrealizowany projekt zakończył się sukcesem, osiągając główny cel, którym było stworzenie w pełni funkcjonalnej, dwumodułowej aplikacji desktopowej do zarządzania sklepem. System, oparty o technologie Java, Swing oraz bazę danych MySQL, poprawnie realizuje wszystkie kluczowe założenia, oferując zarówno panel przeznaczony dla klienta, jak i rozbudowane centrum zarządcze dla administratora.

Zastosowanie trójwarstwowej architektury z wzorcem projektowym DAO (Data Access Object) okazało się kluczowe dla zapewnienia czystości i skalowalności kodu. Udało się skutecznie oddzielić warstwę prezentacji od logiki biznesowej i dostępu do danych, co ułatwiło implementację, testowanie oraz potencjalny dalszy rozwój poszczególnych komponentów systemu.

Wszystkie zdefiniowane wymagania funkcjonalne, takie jak proces logowania, zarządzanie asortymentem i stanami magazynowymi, obsługa koszyka zakupowego oraz finalizacja transakcji, zostały pomyślnie zaimplementowane i zweryfikowane w testach manualnych. Aplikacja stanowi solidną i kompletną podstawę, która może być w przyszłości rozwijana o nowe, zaawansowane funkcjonalności.

### 8.2. Propozycje dalszego rozwoju

Mimo że obecna wersja aplikacji jest w pełni funkcjonalna, istnieje znaczny potencjał do jej dalszej rozbudowy. Poniżej przedstawiono trzy kluczowe kierunki możliwego rozwoju:

- Wdrożenie testów automatycznych:** W celu podniesienia niezawodności i ułatwienia przyszłego utrzymania kodu, kluczowym krokiem byłoby wprowadzenie testów automatycznych. Przy użyciu frameworka **JUnit** oraz bibliotek takich jak **Mockito**, można by stworzyć testy jednostkowe dla klas w warstwie DAO, weryfikując poprawność operacji bazodanowych w izolacji. Umożliwiłoby to szybko wykrywanie regresji po wprowadzeniu nowych zmian.
- Rozbudowa modułu raportowania i analiz:** Obecny system generuje podstawowe raporty w formacie CSV. Znacznym usprawnieniem byłoby stworzenie zaawansowanego modułu analitycznego, który wizualizowałby dane za pomocą wykresów (np. z wykorzystaniem biblioteki **JFreeChart**). Moduł ten mógłby generować raporty sprzedaży w ujęciu czasowym, analizować najpopularniejsze produkty czy przedstawiać statystyki dotyczące aktywności poszczególnych klientów.
- Funkcjonalność resetowania hasła dla użytkowników:** Wdrożenie bezpiecznego mechanizmu resetowania hasła, co zwiększyłoby użyteczność i bezpieczeństwo aplikacji dla użytkowników końcowych. Wymagałoby to m.in. implementacji wysyłki wiadomości e-mail z linkiem do resetowania oraz odpowiedniej obsługi po stronie serwera.

## Bibliografia

[1] LGoodDatePicker Team. Lgooddatepicker, 2024. *<https://github.com/LGoodDatePicker>*.



# Spis rysunków

3.1	Uproszczony diagram klas UML projektu. . . . .	10
3.2	Schemat bazy danych (ERD) aplikacji sklepu. . . . .	11
4.1	Harmonogram realizacji projektu (Diagram Ganta). . . . .	13
5.1	Główny ekran aplikacji z widocznym panelem logowania. . . . .	15
5.2	Widok panelu zakupowego dla klienta detalicznego. . . . .	16
5.3	Główny widok panelu administratora. . . . .	17
5.4	Panel szczegółowych informacji o kliencie. . . . .	18
5.5	Widok panelu zarządzania transakcjami z zintegrowanym kalendarzem. . . . .	19
6.1	Komunikat o błędnym wprowadzeniu danych logowania. . . . .	21
6.2	Przykład błędnego wprowadzenia danych przez klienta detalicznego. . . . .	21
6.3	Komunikat informujący o próbie złożenia zamówienia z pustym koszykiem. . . . .	22

## Spis listingów

7.1	Metoda <code>createStyledButton</code> . . . . .	23
7.2	Metoda <code>getConnection</code> . . . . .	24
7.3	Metoda tworząca i konfigurująca komponent <code>DatePicker</code> . . . . .	24

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

## OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY

.....Dominik Kuraś.....

Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

.....Informatyka.....

Nazwa kierunku

.....134939.....

Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Projekt i implementacja desktopowej aplikacji do zarządzania sklepem z wykorzystaniem języka Java i bazy danych MySQL
  - 1) została przygotowana przeze mnie samodzielnie\*,
  - 2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
  - 3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
  - 4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.
2. Jednocześnie wyrażam zgodę/nie wyrażam zgody\*\* na udostępnienie mojej pracy projektowej do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

---

(miejscowość, data)

---

(czytelny podpis studenta)

\* Uwzględniając merytoryczny wkład prowadzącego przedmiot

\*\* – niepotrzebne skreślić