****

**UNIWERSYTET ŁÓDZKI**

**WYDZIAŁ MATEMATYKI I INFORMATYKI**

**Dokumentacja projektu**

**System ERP**

**Autor:**

**Dominik Sobierański**

**Łódź, 2016**

Spis treści

1. System ERP i jego historia.
2. Definicja systemu ERP.
3. Historia systemu ERP.
4. Środowisko.
5. Dokumentacja techniczna.
6. Opis klas występujących w projekcie.
7. Zastosowane algorytmy.
8. Dokumentacja użytkownika.

**1. System ERP i jego historia**

1. Definicja systemu ERP

System ERP służy wspomaganiu zarządzania przedsiębiorstwem lub współdziałania grupy współpracujących ze sobą przedsiębiorstw, poprzez gromadzenie danych oraz umożliwienie wykonywania operacji na zebranych danych. Wspomaganie to może obejmować wszystkie lub część szczebli zarządzania i ułatwia optymalizację wykorzystania zasobów przedsiębiorstwa oraz zachodzących w nim procesów.

Wyróżniamy kilka rodzajów systemów ERP:

1. modułowe, tj. składają się z niezależnych od siebie choć współpracujących ze sobą aplikacji.
2. zintegrowane, tj. składające się z jednej bazy danych oraz jednej platformy biznesowej, gdzie nie występuje żadna wymiana danych pomiędzy modułami, ponieważ wszystkie funkcjonalności korzystają z tych samych danych w czasie rzeczywistym.

System ERP jest rozwinięciem systemu MRP II, który z kolei jest rozwinięciem MRP (ang. Material Requirements Planning, planowanie zapotrzebowania materiałowego), wzbogaconym o moduł CRP czyli planowanie zdolności produkcyjnych.

2. Historia systemu ERP

a) EOQ

W czasach gdy przemysł nie był zdominowany przez komputery używano metod takich jak EOQ (Economic Order Quantity). Służyły one do zarządzania zapasami w łańcuchu dostaw. Definiowały one optymalną wartość do zamówienia tak aby zminimalizować koszty zmienne.

b) TPS

Pierwszym poważnym systemem tego typu był wprowadzony pod koniec lat 40' Toyota Production System (TPS). Był on zbiorem unikatowych japońskich metod zarządzania. Obejmował szerokie zasady kultury przyjęte w tej firmie, a także sposób postrzegania świata i prowadzenia działalności. System koncentrował się na organizacji produkcji i logistyki, wliczając w to pozytywne relacje z dostawcami i klientami. TPS był konglomeratem wielu technik, koncepcji i zasad mających na celu eliminację 3M (tj. muri – nadwyrężenie i trudności, mura – nieregularność, muda – marnotrawstwo).

c) MRP

Stworzony w 1964 przez Josepha Orlicky'ego MRP był odpowiedzią dla systemu Toyota Production System. Pierwszą firmą w której został wprowadzony była Black & Decker. Z czasem liczba firm wzrosła do 700 w 1975 roku, a w 1981 roku do 8 000. Służył on do wyznaczania zapotrzebowań na zasoby materiałowe (surowce, materiały, komponenty itp.). Za jego pomocą można było obliczyć dokładną ilość materiałów i terminarz dostaw w taki sposób, by sprostać ciągle zmieniającemu się popytowi na poszczególne produkty.

Jego głównymi założeniemi były:

* Redukcja zapasów.
* dokładne określenie czasów dostaw surowców i półproduktów.
* dokładne wyznaczenie kosztów produkcji.
* lepsze wykorzystanie posiadanej infrastruktury (magazynów, możliwości wytwórczych).
* szybsze reagowanie na zmiany zachodzące w otoczeniu.
* kontrola realizacji poszczególnych etapów produkcji.

d) MRP II

W 1983 na podstawie MRP powstał MRPII, który został rozbudowany o planowanie zdolności produkcyjnych (CRP) oraz o elementy związane z procesem sprzedaży i wspierające podejmowanie decyzji na szczeblach strategicznego zarządzania produkcją. Poza materiałami związanymi bezpośrednio z produkcją, MRP II uwzględnia także materiały pomocnicze, zasoby ludzkie, pieniądze, czas, środki trwałe i inne.

e) ERP

W latach dziewięćdziesiątych systemy MRP i MRPII wyewoluowały w system ERP, który jest zbiorem wielu modułów, a jego podstawowym elementem jest wspólna baza danych.

Poszczególne moduły systemu ERP obejmują następujące obszary:

* Magazynowanie.
* Zarzadzanie zapasami.
* Śledzenie realizowanych dostaw.
* Planowanie produkcji.
* Zaopatrzenie.
* Sprzedaż.
* Zarządzanie relacjami z klientami.
* Księgowość.
* Finanse.
* Zarządzanie zasobami ludzkimi(płace, kadry).

W skład systemów ERP mogą wchodzić również inne moduły, jak np. moduł zarządzania transportem, controlling, czy zarządzanie projektami. Systemy ERP umożliwiają dopasowanie ich do specyfiki poszczególnych przedsiębiorstw, m.in. dlatego, iż poszczególne moduły mogą być wzajemnie niezależne od siebie (tzn. mogą pracować bez obecności innych modułów). Systemy te zazwyczaj pozwalają też na ustalenie uprawnień dostępu dla poszczególnych użytkowników.

**2. Środowisko**

1. MySQL Workbench - rozwijane przez firmę Oracle wolnodostępne narzędzie administracyjne do zarządzania relacyjnymi bazami danych.
2. Netbeans - zintegrowane środowisko programistyczne (IDE) dla języka Java, którego głównym celem jest przyspieszenie budowy aplikacji Java, w tym również usług sieciowych orazaplikacji mobilnych.
3. iText - biblioteka dla programistów, która umożliwia tworzenie i manipulowanie dokumentów w formacie PDF, w poziomie języka Java.
4. Java DataBase Connectivity -  interfejs umożliwiający niezależnym od platformy aplikacjom napisanym w języku Java porozumiewać się z bazami danych za pomocą języka SQL.
5. Java Development Kit (JDK) – oprogramowanie  udostępniające środowisko niezbędne do programowania w języku Java.

**3. Dokumentacja techniczna.**

Opisz klas występujących w projekcie:

1. Category - odpowiedzialna za przechowywanie w polu 'name' kategorii produktu. Pole id odnosi się do pola o tej samej nazwie w klasie Product. Zabieg ten pozwala na uniknięcie redundancji w bazie danych. Kategoryzowanie produktów pozwala na łatwiejsze ich wyszukiwanie, zamawianie i sprzedaż.

public class Category {

private int id;

private String name;

}

2. Email - pozwala na tworzenie, wysyłanie i odbieranie wiadomości między użytkownikami systemu. Pole 'id\_sender' zawiera id nadawcy, 'id\_receiver' - id odbiorcy. 'Text' przechowuje treść wiadomości, 'date' - dokładną datę wysłania wiadomości. Wartość 'checked' informuje program czy dana wiadomość została już kiedyś odczytana.

public class Email {

private int id;

private int id\_sender;

private int id\_receiver;

private String text;

private Timestamp date;

private boolean checked;

}

3. Employee - każdy obiekt tej klasy odwzorowuje jednego pracownika/użytkownika systemu przechowując jego dane takie jak imie, nazwisko, email, zaszyfrowane hasło oraz pozycję w firmie, która to z kolei jest pobierana z bazy za pośrednictwem kolumny 'id\_position'. Użytkownik logując się informuje system o swojej obecności, a każdy jego ruch jest odnotowany za pośrednictwem pola 'id'.

public class Employee {

private int id;

private String name;

private String full\_name;

private String email;

private String password;

private String position;

}

4. Log - dostarcza dokładniejszych informacji na temat realizowania zamówień. Dzięki niej użytkownik może się dowiedzieć kiedy złożono zamówienie lub kiedy je przyjęto.

public class Log {

private int id;

private int id\_object;

private Timestamp date;

private String action;

}

5. Order - obiekty tej klasy służą głównie do przechowywania listy zamawianych produktów ('list'). Oprócz tego zawierają informację o tym kto je zamówił('id\_employee'), kiedy('date'), czy przyjęto dostawę('executed') i ile wynosiły koszty('price'). Informacja o produktach z listy jest pobierana z bazy jako obiekt typu String, który jest przetwarzany na listę za pomocą specjalnego algorytmu, który zostanie omówiony później.

public class Order {

private int id;

private int id\_employee;

private Timestamp date;

private List<Product> list;

private boolean executed;

private Double price;

}

6. PdfFiles - klasa której zadaniem jest generowanie pliku .Pdf zawierającego fakturę w formie tabeli wraz z łączną ceną i datą. Zawiera ona następujące metody:

* public static void createPdf(java.util.List<Product> selling\_list) - tworzy pełny dokument.
* public static void addTitlePage(Document document) - dodaje nagłówek do dokumentu, który zawiera m. in. bieżącą datę.
* private static void addEmptyLine(Paragraph paragraph, int number) - dodaje do paragrafu puste linie w ilości podanej w drugim argumencie.
* private static void createTable(Document document,java.util.List<Product> selling\_list) - najważniejsza metoda, która generuje główną tabelę z produktami i cenę na podstawie podanej listy produktów.

7. Position - klasa podobna do Category z tą różnicą że przechowuje pełną nazwę stanowiska pracownika do którego pracownik odwołuje się za pomocą odpowiedniego pola.

public class Position {

private int id;

private String name;

}

8. Product - klasa przechowująca dane na temat produktu takie jak m. in. nazwa('name'), cena detaliczna('retail\_price'), VAT('vat'), kategoria('category') oraz ilość na magazynie('quantity').

public class Product {

private int id;

private String name;

private Double retail\_price;

private Double vat;

private String category;

private int quantity;

}

9. Repair - obiekt tej klasy odzwierciedla złożone przez klienta zamówienie na naprawę sprzętu. Zawiera ono informację o pracowniku przyjmującym('id'), imię i nazwisko klienta('client\_name', 'client\_full\_name'), opis problemu('description'), datę złożenia zamówienia('date') oraz informację o tym czy zamówienie zostało wykonane i jest gotowe do odbioru('executed').

public class Repair {

private int id;

private int id\_employee;

private String client\_name;

private String client\_full\_name;

private String description;

private Timestamp date;

private boolean executed;

}

10. Function - główna klasa, która jest swoistą biblioteką funkcyjną. Obiekt tej klasy gwarantuje dostęp do wszelkich niezbędnych metod używanych w aplikacji. Zawiera pole typu Connection dzięki któremu konstruktor automatycznie tworzy połączenie z bazą danych wykorzystując do tego informacje zawarte w zewnętrznym pliku DBInfo.properties. Pola 'ALGORITHM' i 'keyValue' używane są w funkcjach szyfrujących.

public class Function {

Connection myConn = null;

private static final String ALGORITHM = "AES";

private static final byte[] keyValue

= new byte[]{'T', 'h', 'i', 's', 'I', 's', 'A', 'S', 'e', 'c', 'r', 'e', 't', 'K', 'e', 'y'};

}

Do klasy Function należą następujące funkcje:

* **public String encrypt(String valueToEnc)** - szyfruje podany jako argument ciąg znaków.
* **public String decrypt(String encryptedValue)** - deszyfruje ciąg znaków.
* **private static Key generateKey()** - pomocnicza metoda tworząca klucz niezbędny do szyfrowania i odszyfrowywania.
* **private Employee convertRowToEmployee(ResultSet rs)** - przetwarza wynik zapytania do bazy na obiekt typu Employee.
* **public String getPosition(int index)** - zwraca nazwę stanowiska na podstawie podanego indeksu.
* **public List<Employee> getAllEmployees()** - pobiera z bazy wszystkich pracowników i zwraca ich w formie listy.
* **private Email convertRowToEmail(ResultSet rs)** - przetwarza wynik zapytania do bazy na obiekt typu Email.
* **public List<Email> getAllEmails()** - pobiera z bazy wszystkie wiadomości email i zwraca je w formie listy.
* **private Repair convertRowToRepair(ResultSet rs)** - przetwarza wynik zapytania do bazy na obiekt typu Repair.
* **public List<Repair> getAllRepairs()** - pobiera z bazy wszystkie formularze naprawy i zwraca je w formie listy.
* **private Product convertRowToProduct(ResultSet rs)** - przetwarza wynik zapytania do bazy na obiekt typu Product.
* **public String getCategory(int index)** - zwraca nazwę kategorii na podstawie podanego indeksu.
* **public List<Product> getAllProducts()** - pobiera z bazy wszystkie produkty i zwraca je w formie listy.
* **public List<Product> convertStringCodeToProductsList(String string)** - metoda, która przetwarza ciąg znaków na listę produktów. // tutaj dać przykład i wyjaśnienie albo niżej. jeszcze sie pomysli.\
* **private Product getProductById(int id)** - pobiera z bazy danych produkt o podanym id.
* **private Order convertRowToOrder(ResultSet rs)** - przetwarza wynik zapytania do bazy na obiekt typu Order.
* **public List<Order> getAllOrders()** - pobiera z bazy wszystkie zamówienia produktów i zwraca je w formie listy.
* **public Employee getEmployeeById(int id)** - pobiera z bazy danych użytkownika o podanym id.
* **public void fillComboboxWithEmployees(JComboBox<Employee> combobox)** - czyści comboboxa i wypełnia go pobranymi z bazy danych pracownikami.
* **public void addEmail(Email temp)** - dodaje do bazy nową wiadomość mailową.
* **void fillComboboxWithEmployeesWithoutUser(JComboBox combobox, Employee user)** - czyści comboboxa i wypełnia go pobranymi z bazy danych pracownikami za wyjątkiem jednego podanego w argumencie.
* **public List<Email> getAllEmailsForUser(Employee user)** - pobiera z bazy wszystkie wiadomości email adresowane dla podanego użytkownika i zwraca je w formie listy.
* **public void addRepair(Repair temp)** - dodaje do bazy nowy formularz naprawy.
* **public void setEmailChecked(Email email)** - oznacza podany email jako przeczytany.
* **public void fillTableWithEmailsForUser(JTable jTableMailbox, Employee user)** - wypełnia tabelę wiadomościami email adresowanymi do podanego użytkownika.
* **public boolean thereIsNewMail(Employee user)** - sprawdza czy dla podanego użytkownika istnieje nieprzeczytany jeszcze mail.
* **public void fillTableWithRepairs(JTable jTableRepairs)** - wypełnia podaną tabelę formularzami napraw.
* **public void setRepairExecuted(Repair temp)** - oznacza podany formularz naprawy jako wykonany.
* **public void fillTableWithAllProducts(JTable jTableProducts)** - wypełnia podaną tabelę wszystkimi produktami bez względu na kategorie.
* **public List<Product> getFoodProducts()** - pobiera z bazy produkty z kategorii 'Jedzenie' i zwraca je w formie listy.
* **public void fillTableWithFoodProducts(JTable jTableProducts)** - wypełnia podaną tabelę tylko produktami z kategorii 'Jedzenie'.
* **public List<Product> getPartsProducts()** - pobiera z bazy wszystkie produkty z wyjątkiem tych z kategorii 'Jedzenie' i zwraca je w formie listy.
* **public void fillTableWithPartsProducts(JTable jTableProducts)** - wypełnia podaną tabelę wszystkimi produktami z wyjątkiem tych z kategorii 'Jedzenie'.
* **public List<Product> searchPartsProducts(String name)** - zwraca listę Produktów z wyjątkiem tych z kategorii 'Jedzenie' i tych które w nazwie zawierają podany wzorzec.
* **public void fillTableWithPartsProducts(JTable jTableProducts,String name)** - wypełnia podaną tabelę wszystkimi produktami z wyjątkiem tych z kategorii 'Jedzenie' które pasują do podanego wzorca.
* **public void removeColumn(JTable table, int x)** - usuwa z podanej tabeli columne o numerze x.
* **public void selectColumns(JTable jTableFoodOrdered, JTable jTableFoodToOrder)** - pomocnicza funkcja, która dostosowywuje podane tabele usuwając niepotrzebne kolumny.
* **void refreshOrderedFood(JTable jTableFoodOrdered, List<Product> ordered\_list)** - odświeża podaną tabelę wypełniając ją na nowo produktami z listy.
* **public boolean containsProductID(List<Product> ordered\_list, Product temp)** - sprawdza czy w liście istnieje już produkt o podanym id.

Zastosowane algorytmy

Algorytm wykorzystany w wyszukiwarce zaawansowanej pozwala na efektywne wyszukiwanie produktów według następujących kategorii:

* Nazwa
* Cena minimalna
* Cena maksymalna
* Ilość minimalna
* Ilość maksymalna
* Kategoria

Pierwszym wykonywanym krokiem jest utworzenie pustej liczby produktów. Lista ta jest następnie wypełniana wszystkimi produktami z bazy z wyjątkiem produktów żywnościowych. Jeśłi użytkownik nie zaznaczyl żadnej kategorii skrypt nie będzie kontynuowany. Kolejny punkt to pobranie nazwy, wartości cen oraz ilości. W przypadku gdy te pola są puste program nie bierze ich pod uwagę. Później tworzona jest lista indeksów, która będzie informowała, który obiekt nie jest zgodny ze wzorcem i należy go usunąć. Teraz algorytm sprawdza każdy wzorzec wypełniony przez użytkownika i porównuje z produktami w liście. Jeśli dochodzi do konfiktu na listę indexów zostaje wpisany indeks danego produktu (uwzględniając sytuację, że dany obiekt może się tam już znajdować). Kolejnym istotnym krokiem jest posortowanie listy indeksów w kolejności od najmniejszego do największego. Następnie lista jest odwracana. W tym momencie iterując po indeksach usuwamy każdy niepasujący element z listy produktów. Zabieg odwrócenia indeksów i usuwania obiektów od końca zapobiega powstawaniu luk i przesuwaniu się indeksów w liście produktów. Na tym etapie lista produktów zawiera tylko te elementy, które pasują do wzorca podanego przez użytkownika. Przebieg algorytmu przedstawia następujacy wykres.

Kolejnym istotnym algorytmem jest sposób na konwersję ciągu znaków przechowywanego w zamówieniu na listę produktów. Jest on zawarty w funkcji convertStringCodeToProductsList w klasie Function. Kod wygląda następująco:

public List<Product> convertStringCodeToProductsList(String string) {

List<Product> list = new ArrayList<Product>();

String index = "";

String quantity = "";

boolean existsIndex = false;

try {

for (char x : string.toCharArray()) {

if (x != ',' && x != ';' && existsIndex == false) {

index += x;

continue;

}

if (x != ',' && x != ';' && existsIndex == true) {

quantity += x;

continue;

}

if (x == ',') {//jest już index

existsIndex = true;

continue;

}

if (x == ';') {

int nr = Integer.parseInt(index);

int count = Integer.parseInt(quantity);

Product temp = getProductById(nr);

temp.setQuantity(count);

list.add(temp);

index = "";

quantity = "";

existsIndex = false;

continue;

}

}

} catch (Exception e) {

JOptionPane.showMessageDialog(null, "Error while getting products list.");

}

return list;

}

Lista produktów jest kodowana w następującym formacie:

[ID produktu],[Ilość produtu]; [ID produktu],[Ilość produtu];....

W ten sposób zamówienie na 5 produtów o ID = 4, 30 produtów o ID = 14 i 2 produkty o ID = 110 wygląda tak:

**4,5;14,30;110,2;**

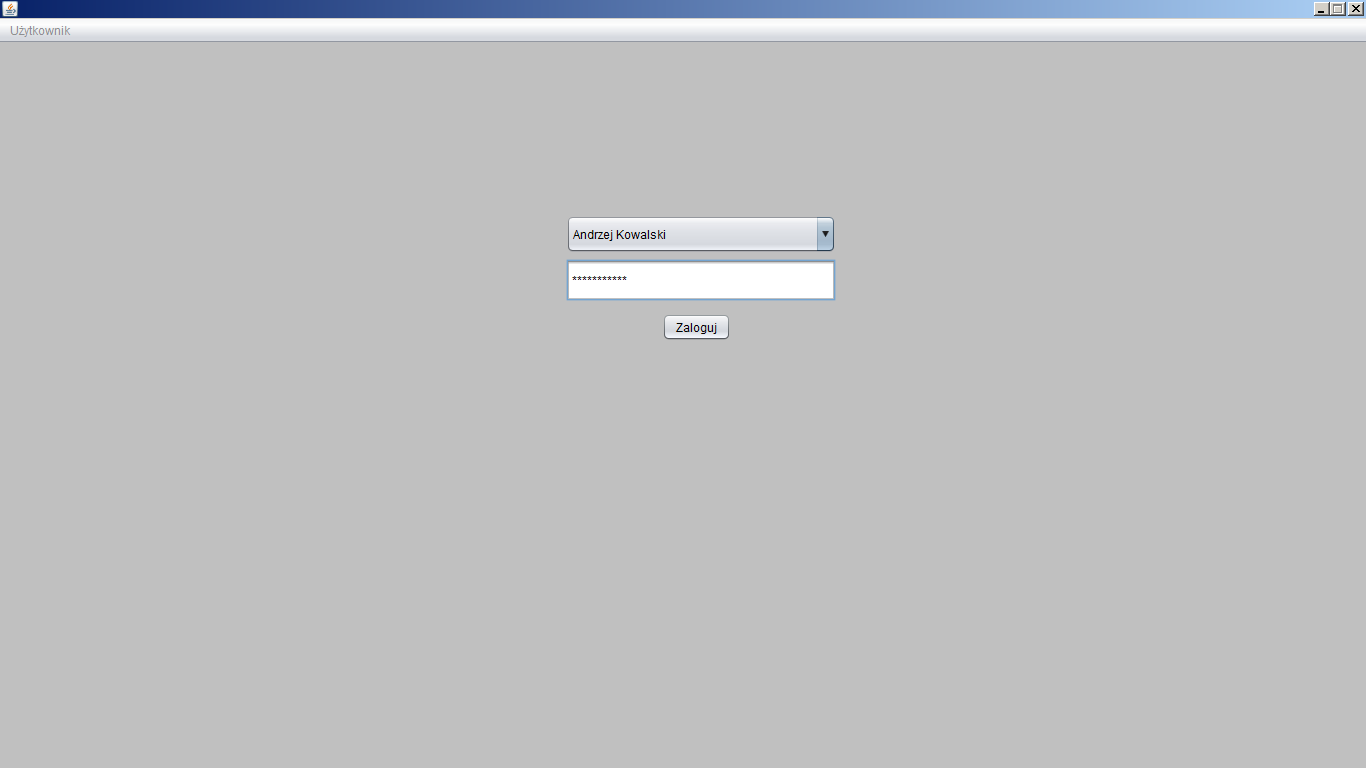
Poniżej przedstawione są kolejne iteracje algorytmu (iterator odczytuje każdy znak po kolei). Za przykład posłuży ciąg

**12,3;4,56;**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr iteracji | Odczytany znak | index | quantity | existsIndex |
| 1 | 1 | „1” | „” | false |
| 2 | 2 | „12” | „” | false |
| 3 | , | „12” | „” | true |
| 4 | 3 | „12” | „3” | true |
| 5 | ; | „12” | „3” | true |
| W piątej iteracji mamy wszystkie dane na temat produktu. Dodajemy go do listy i zerujemy zmienne index, quantity i existsIndex. | | | | |
| 6 | 4 | „4” | „” | False |
| 7 | , | „4” | „” | true |
| 8 | 5 | „4” | „5” | true |
| 9 | 6 | „4” | „56” | true |
| 10 | ; | „4” | „56” | true |
| Tutaj znowu dodajemy produkt do listy. Na tym kończy się pętla i mamy gotową listę z produktami. | | | | |

**4. Dokumentacja użytownika.**

Pierwszym oknem jakie widzi użytkownik po włączeniu aplikacji jest okno logowania.

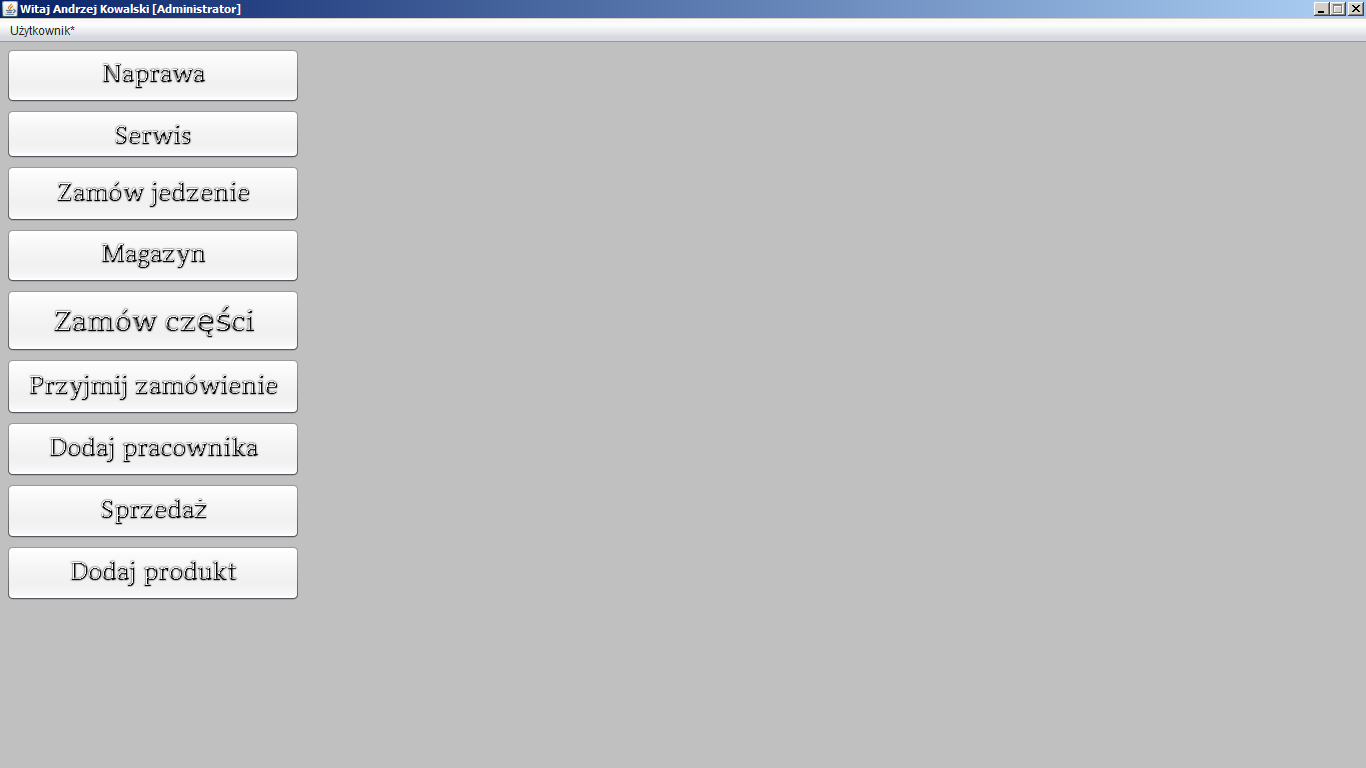
****

Należy tutaj wybrać z listy rozwijanej odpowiedniego pracownika, wpisać hasło i kliknąć ‘Zaloguj’. Hasła w bazie są przechowywane w formie zaszysfrowanej, więc nawet jeśli ktoś zdobędzie dostęp do skryptu z informacjami na temat pracowników, dalej nie będzie mógł zalogować się na nieswoje konto. W lewym górnym rogu znajduje się przycisk z ustawieniami i pocztą meilową. Dostęp do niego uzyska się dopiero po zalogowaniu.

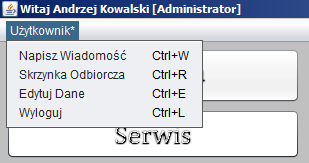
Aplikacja opiera się na prawach dostępu tj. każdy użytkownik ma dostęp tylko do tej części oprogramowania, która jest mu niezbędna do pracy. Obrazuje to poniższy wykres.

Administrator ma dostęp do wszystkich funkcji i możliwości. Z tego względu dalsza część opisu będzie oparta o konto administratora.

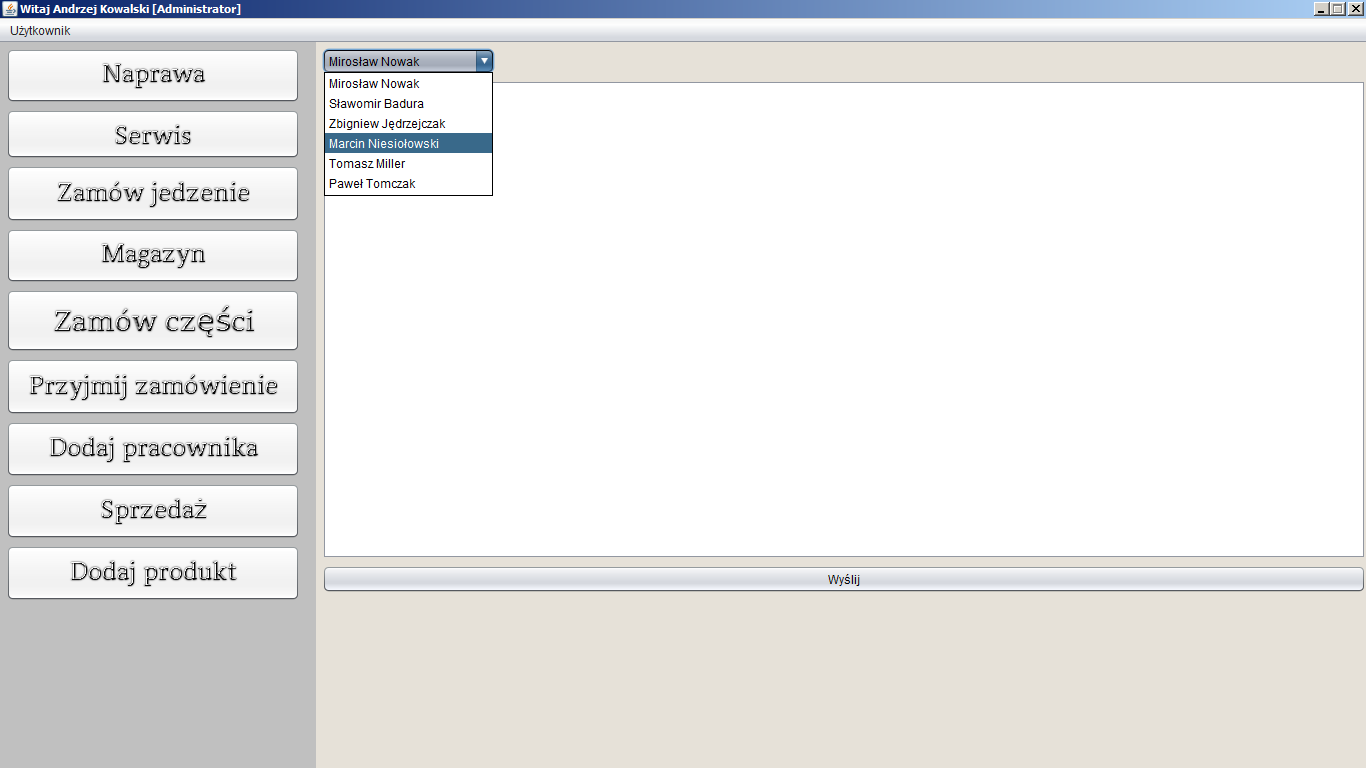
Po zalogowaniu ukazuje się następujące okno.



Podzielone jest ono na dwie części. Pionowy pasek po lewej zawiera główne przyciski przełączające między panelami. Te z kolei wyświetlane są na prawej części okna. W tej chwili obszar ten jest pusty. Na górnym pasu widnieje informacja o zalogowanym użytkowniku. Pod nią mieści się przycisk z rozwijanym menu, którym można zmienić prywatne ustawienia konta lub zarządzać meilami. Gwiazdka informuje użytownika o tym, że w jego skrzynce meilowej znajduje się nieprzeczytana wiadomość.

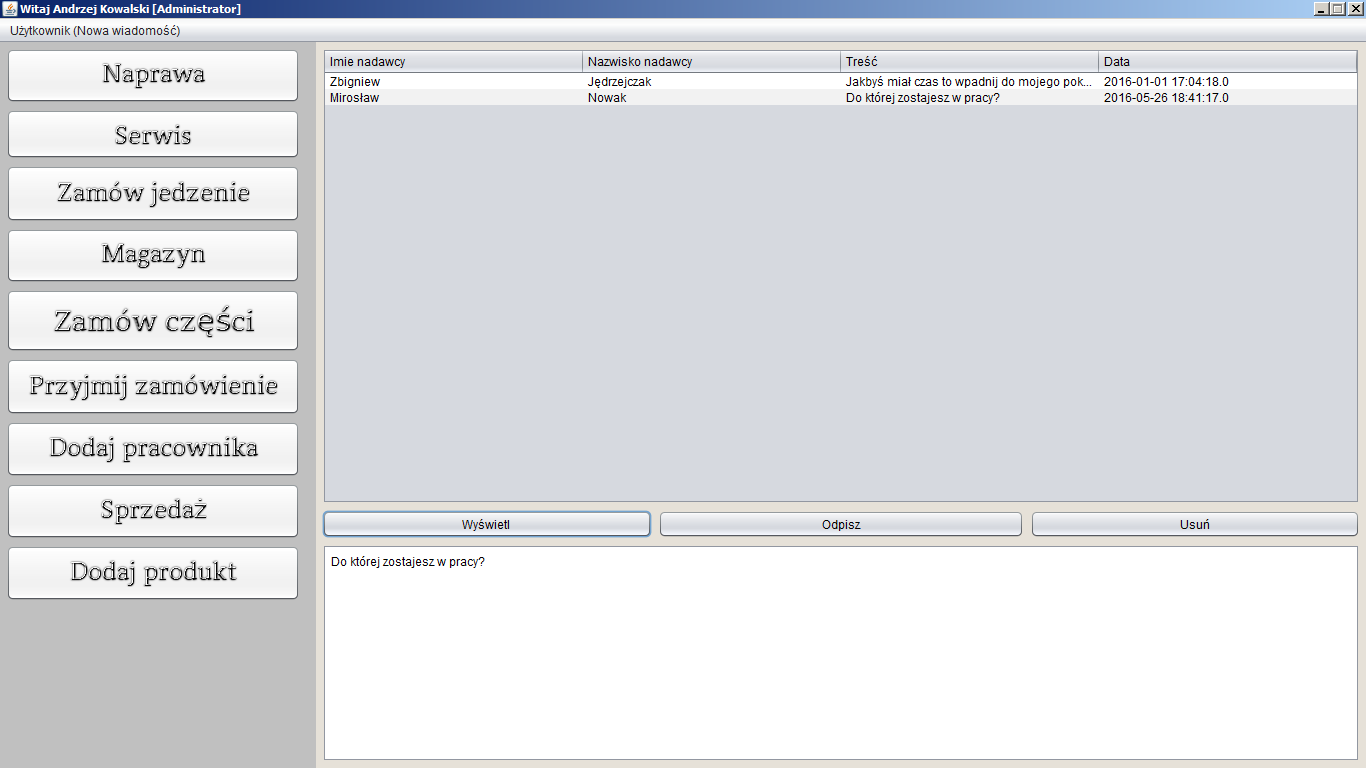


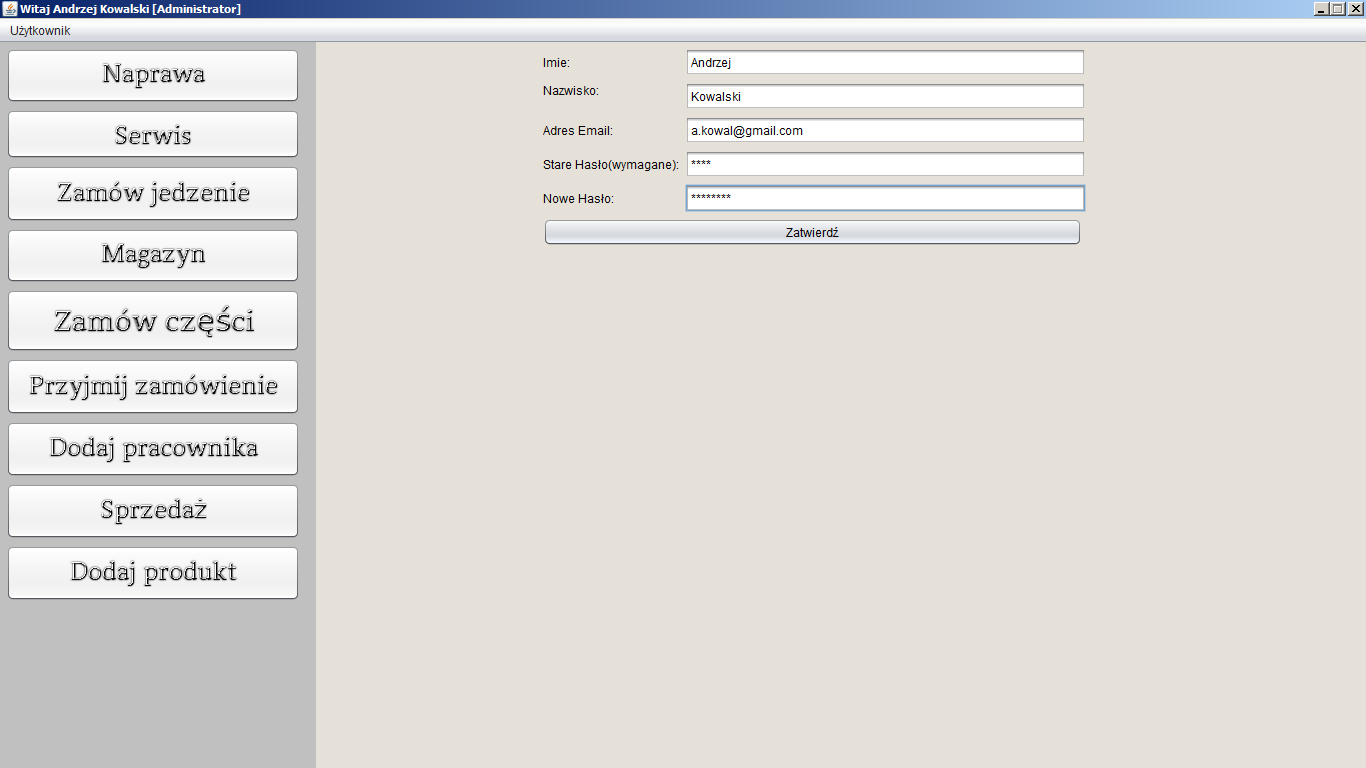
W rozwijalnym menu mamy widoczne opcje. Przydzielone są do nich odpowiednie skróty klawiszowe. Po kliknięciu w ‘Napisz Wiadomość’ pokazuje się to okno.



Z listy rozwijanej wybieramy adresata, na białym polu piszemy treść, a przycisk ‘Wyślij’ służy do wysyłania wiadomości. Zostaje ona zapisana w bazie i dostęp do niej ma tylko jej adresat.

Poniżej mamy skrzynkę odbiorczą. Możemy z jej poziomu wyświetlić zaznaczoną wiadomość, odpisać lub ją usunąć. Po każdym wyświetleniu lub usunięciu skrypt sprawdza czy w skrzynce znajdują się nieprzeczytane wiadomości i jeśli znajdzie takowe, ustawia je na liście w pierwszej kolejności. Jeśli użytkownik będzie chciał odpisać, program automatycznie przeniesie go do okna pisania wiadomości i sam wybierze adresata.

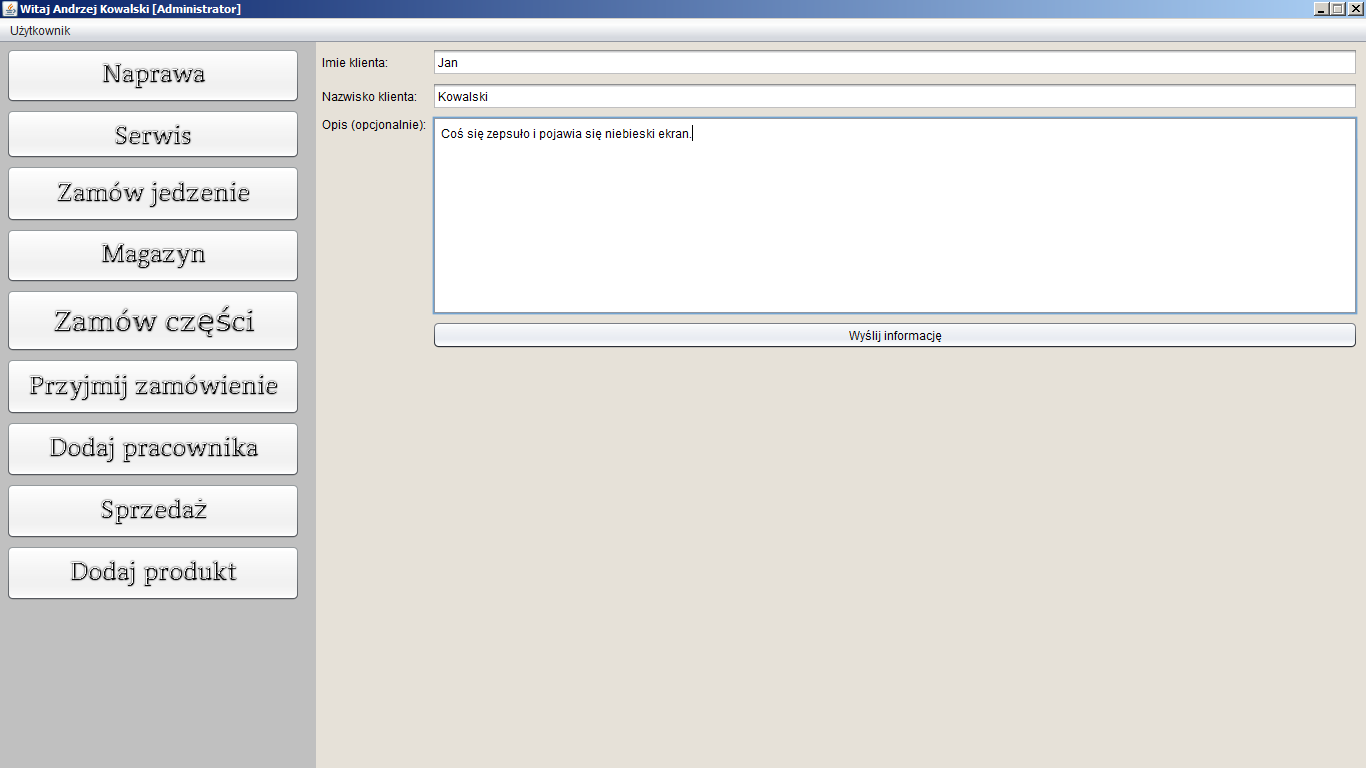




Powyższe okno służy do konfiguracji ustawień swojego konta. Możemy w nim zmienić takie dane jak imię, nazwisko, adres mailowy, oraz hasło. Niezbędnym warunkiem jest podanie swojego obecnego hasła.

Przycisk ‘Wyloguj’ przenosi do ekranu logowania i odłącza aktywnego użytkownika od aplikacji.

Przycisk ‘Naprawa’ służy do włączania następującego panelu.



Jest to formularz jaki składa klient przynoszący sprzęt do naprawy. Pracownik obsługujący wypełnia go danymi takimi jak imie i nazwisko oraz opcjonalnie podaje opis problemu. Formularz ten jest następnie wysyłany do bazy i dostęp do niego zyskują wszyscy technicy. Ich zadaniem jest wykonanie zamówienia i wysłanie potwierdzenia. Technik może też wyświeltlić opis osobno. W tabeli zawarte są informacje o tym który z pracowników nadał zlecenie, dane klienta, opis problemu, data utworzenia i powiadomienie o tym czy zlecenie zostało już wykonane.

