

**Program wystawiający rachunki.**

Opracowali:

Jarosław Tomczuk Nr. Albumu 22261

Przemysław Bondyra Nr. Albumu …..

**Cel projektu:**

Wprowadzenia do UML – poprzez wykonanie projektu programu za pomocą wybranych diagramów UML. Zapoznanie z językiem modelowania UML umożliwia projektowanie oprogramowania za pomocą różnorodnych diagramów, które reprezentują różne aspekty programu.

1. **Opis biznesowy „świata rzeczywistego” programu wystawiającego rachunki.**

1.1.**Opis zasobów ludzkich.**

Uprawniony pracownik firmy może generować faktury na podstawie danych dotyczących klienta biznesowego oraz produktów/usług, cen, podatków i innych szczegółów potrzebnych do utworzenia faktury. Użytkownik może przeglądać profile klientów wraz z wystawionymi fakturami. Administrator ma dodatkowe uprawnienie umożliwiające zarządzanie kontami uprawnionych pracowników. Użytkowników oprogramowania identyfikuje unikalna nazwa użytkownika ustalona według potrzeb firmy. Klient biznesowy nie jest użytkownikiem oprogramowania. Jego unikalnym identyfikatorem jest numer NIP, który umożliwia identyfikację i szybkie odnalezienie klienta w systemie.

1.2. **Przepisy, strategia firmy**

Odpowiedzialność za poprawność danych leży po stronie użytkownika korzystającego z programu. System wystawiania faktur powinien być intuicyjny i łatwy w obsłudze.

1.3. **Dane techniczne**

Program ma być obsługiwany przez pojedynczego użytkownika w trybie konsolowym. Wykorzystuje prosty interfejs tekstowy, umożliwiając wprowadzanie danych z poziomu linii poleceń systemu operacyjnego. Zakłada się, że klientów w bazie profilów może być 1000, a faktur kilkadziesiąt tysięcy. Liczba użytkowników uprawnionych do wystawiania faktur w firmie jest ograniczona do stu kont. Zalecana jest implementacja w języku Java.

1. **Wymagania programu opracowane na podstawie opisu „świata rzeczywistego”.**

**A diagram of a company

Description automatically generated with medium confidence**

**A diagram of a computer program

Description automatically generated with medium confidence**

2.1. **Wymagania funkcjonalne.**

* + 1. Zakładanie, aktualizacja oraz usuwanie konta uprawnionego podmiotu wystawiającego faktury.
    2. Zakładanie, aktualizacja, usuwanie oraz przeglądanie profilów klientów w celu usprawnienia procesu wystawiania faktur - katalogowanie klientów.
    3. Zarządzanie "bazą faktur" przypisaną do profilu klienta (wystawianie faktur na podstawie wprowadzonych danych tj. informacji dotyczących klienta, produktów/usług, cen, podatków i innych szczegółów potrzebnych do utworzenia faktury, usuwanie, aktualizacja, automatyczny binarny zapis, eksport do pliku PDF).
  1. **Wymagania niefunkcjonalne.**
     1. Zarządzanie kontami podmiotu wystawiającego faktury, profilami klientów może odbywać się tylko z poziomu konta administratora.
     2. Wymagana jest autoryzacja i uwierzytelnianie podmiotu wystawiającego faktury przy każdorazowym uruchomieniu programu.
     3. Należy zabezpieczyć odpowiednio wrażliwe dane (dane powiązane z profilami klientów oraz dane logowania do programu).
     4. System powinien być dostosowany pod obsługę min. 1000 profilów klientów i przechowywanie w kopii kilkudziesięciu tysięcy faktur. Sugeruje się ograniczenie liczby użytkowników do stu kont (w tym jedno konto administratora).
     5. Zalecana jest implementacja w języku Java.
     6. Interfejsem użytkownika powinien być terminal, konsola poleceń systemu. Należy uprościć proces wystawiania faktury poprzez stworzenie prostego, przejrzystego interfejsu.

**3. Diagram przypadków użycia reprezentujący specyfikację wymagań funkcjonalnych**.

3.1 **Diagram przypadków użycia.**

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

3.2. **Scenariusze przypadków użycia.**

* **Nazwa PU:** ZARZĄDZANIE PROFILAMI KLIENTÓW

**Cel:** Zakładanie profilu klienta. Nadanie niepowtarzalnej nazwy klienta. Aktualizowanie, usuwanie oraz przeglądanie profili klientów.

**Warunki początkowe:** Uruchomienie programu jako aplikacji.

**Warunki końcowe:** Wprowadzenie niepowtarzalnej nazwy klienta.

**Scenariusze:**

1. Należy podać niepowtarzalna nazwę klienta.
2. Należy wywołać PU SZUKANIE PROFILÓW KLIENTA w celu sprawdzenia czy nazwa klienta się nie powtarza.
3. Jeżeli zwrócony wynik oznacza brak profilu klienta o podanej nazwie, możemy założyć nowy profil klienta i zakończyć PU. W przeciwnym wypadku należy zakończyć PU bez zakładania nowego profilu klienta.

* **Nazwa PU:** SZUKANIE PROFILU KLIENTA.

**Cel:** Wyszukanie profilu klienta o podanej nazwie.

**Warunki początkowe:** Uruchomienie z PU ZARZĄDZANIE PROFILAMI KLIENTÓW.

**Warunki końcowe:** Zwraca wynik określający, czy istnieje profil o podanym numerze NIP.

**Scenariusze:**

1. Szukanie profilu klienta przebiega według atrybutu: numeru .
2. Jeśli istnieje profil klienta o podanym numerze NIP zwracany jest profil. W przeciwnym wypadku zwracana jest informacja o braku profilu.

* **Nazwa PU:** ZARZĄDZANIE BAZĄ FAKTUR.

**Cel:** Zarządzanie "bazą faktur" przypisaną do profilu klienta (wystawianie faktur na podstawie wprowadzonych danych tj. informacji dotyczących klienta, produktów/usług, cen, podatków i innych szczegółów potrzebnych do utworzenia faktury usuwanie, aktualizacja, automatyczny binarny zapis, eksport do pliku PDF).

**Warunki początkowe:** Uruchomienie programu jako aplikacji.

**Warunki końcowe:** Wprowadzenie niepowtarzalnej nazwy faktury.

**Scenariusze:**

1.Należy podać niepowtarzalną nazwę faktury.

2.Należy wywołać PU SZUKANIE FAKTUR w celu sprawdzenia, czy nazwa faktury się powtarza.

3. Jeśli zwrócony wynik z PU SZUKANIE FAKTUR wskazuje na istnienie faktury, można dokonać aktualizacji.

* **Nazwa PU:** SZUKANIE FAKTUR

**Cel:** Szukanie faktury.

**Warunki początkowe:** Uruchomienie z PU ZARZĄDZANIE BAZĄ FAKTUR.

**Warunki końcowe:** Zwraca wynik określający, czy istnieje faktura o podanych atrybutach.

**Scenariusze:**

1.Szukanie faktury przebiega na podstawie daty oraz numeru NIP do którego przypisana jest faktura.

2.Jeśli istnieje faktura zgodna z kryterium wyszukiwania, zwracana jest informacja o wystawionej fakturze. W przeciwnym wypadku zwracana jest informacja o braku faktury spełniającej kryterium wyszukiwania.

* **Nazwa PU:** ZARZĄDZANIE KONTAMI UPRAWNIONYCH PRCOWNIKÓW

**Cel:** Zakładanie, aktualizacja oraz usuwanie konta uprawnionego podmiotu wystawiającego faktury.

**Warunki początkowe:** Uruchomienie programu jako aplikacji.

**Warunki końcowe:** Wprowadzanie niepowtarzalnej nazwy pracownika.

**Scenariusze:**

1.Należy podać niepowtarzalną nazwę pracownika.

2.Należy wywołać PU SZUKANIE KONTA UPRAWNIONEGO PRACOWNIKA w celu sprawdzenia czy nazwa konta uprawnionego pracownika się nie powtarza.

3.Jeżeli zwrócony wynik oznacza brak profilu pracownika o podanej nazwie, możemy założyć nowy profil pracownika i zakończyć PU. W przeciwnym wypadku należy zakończyć PU bez zakładania nowego profilu pracownika.

**4. Diagram oraz identyfikacja klas.**

**4.1 Identyfikacja klas.**

Zidentyfikowano następujące klasy oraz opisano implantację i relacje zachodzące między poszczególnymi klasami. Dla każdej pary klas: ClientProfile-Clients, EmployeeProfile-Employees, Invoice-Invoices, zastosowano wzorzec strukturalny pyłek. Dodatkowo zaprojektowano klasę InvoiceApp, która steruje działaniem aplikacji. Klasa InvoiceApp reprezentuje wzorzec strukturalny Fasada.

|  |  |  |  |
| --- | --- | --- | --- |
| **Przypadek użycia** | **Implementacja** | **Relacja** | **Klasa** |
| **Zarządzanie profilami klientów** | Odczytywanie danych  po starcie programu  i zalogowaniu. Odczytywanie odbywa się z pliku do listy obiektów typu ClientProfile.  Po wczytaniu danych, klientami można zarządzać  z poziomu obiektu typu Clients. | Dwukierunkowa jeden do wielu | ClientProfile |
| Zakładanie profilu klienta. Nadanie niepowtarzalnej nazwy klienta. Aktualizowanie, usuwanie oraz przglądanie profili klientów z poziomu obiektu typu Clients. | Clients |
| **Szukanie profilu klienta** | Szukanie odbywa się według unikalnej nazwy klienta, numeru nip lub adresu siedziby. Przeszukiwana jest lista obiektów typu ClientProfile. | brak | ClientProfile |
| **Zarządzanie kontami uprawnionych pracowników** | Odczytywanie danych po starcie programu i zalogowaniu. Odczytywanie odbywa się z pliku do listy obiektów typu EmployeeProfile. | Dwukierunkowa jeden do wielu | EmployeeProfile |
| Zakładanie, aktualizacja oraz usuwanie konta uprawnionego podmiotu wystawiającego faktury z poziomu obiektu typu Employees. | Employees |
| **Szukanie profilu uprawnionego pracownika** | Szukanie odbywa się według unikalnej nazwy pracownika. Przeszukiwana jest lista obiektów typu EmployeeProfile. | brak | EmployeeProfile |
| **Zarządzanie bazą faktur** | Odczytywanie danych po starcie programu i zalogowaniu. Odczytywanie odbywa się z pliku do listy obiektów typu Invoice. | Dwukierunkowa jeden do wielu | Invoice |
| Zarządzanie "bazą faktur" przypisaną do profilu klienta (wystawianie faktur na podstawie wprowadzonych danych tj. informacji dotyczących klienta, produktów/usług, cen, podatków i innych szczegółów potrzebnych do utworzenia faktury usuwanie, aktualizacja, automatyczny binarny zapis, eksport do pliku PDF) z poziomu obiektu typu Invoices. | Invoices |
| Profil klienta jest wymagany do wystawienia faktury. | Jednokierunkowa jeden do jednego (ClientProfile->Invoice) | ClientProfile |
| **Szukanie faktury** | Szukanie odbywa się według numeru NIP do którego przypisana jest faktura, daty lub unikalnego identyfikatora. Przeszukiwana jest struktura obiektów typu Invoice. | brak | Invoice |

**4.2 Diagram klas.**

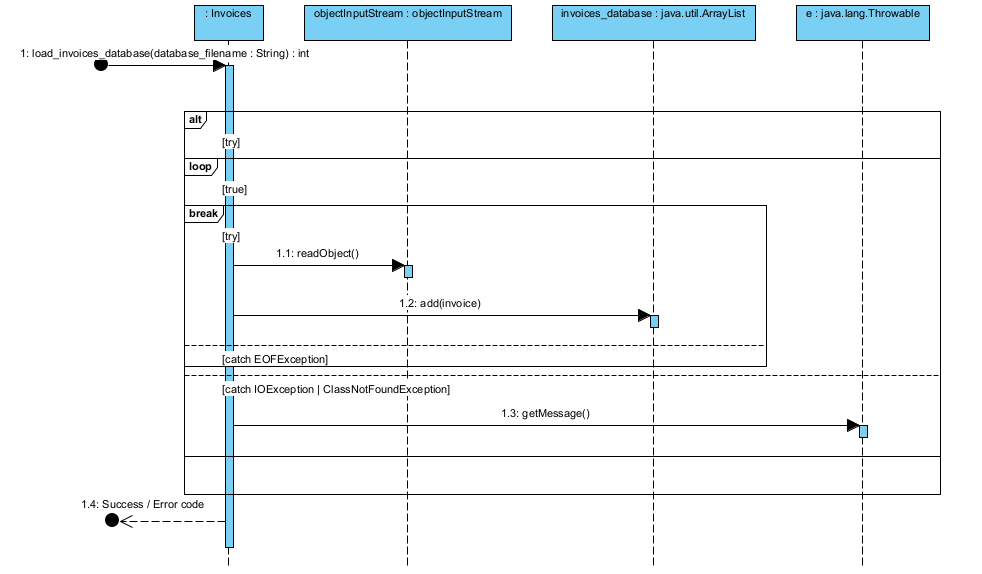
**A diagram of a software company

Description automatically generated with medium confidence**

**5. Diagramy sekwencji.**

**5.1. Diagram sekwencji metody Invoices:**

**load\_invoices\_database(String database\_filename) → Invoices class**

****

**5.1.1. Kod metody:** **load\_invoices\_database(String database\_filename) → Invoices class**

public int load\_invoices\_database(String database\_filename) {

try (ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream(database\_filename))) {

// Read all invoices from the file

while (true) {

try {

Invoice invoice = (Invoice) objectInputStream.readObject();

invoice\_database.add(invoice);

}

catch (EOFException e) {

// Reached the end of the file

break;

}

catch (IOException | ClassNotFoundException e) {

System.err.println("Error reading object from file " + database\_filename + ": " + e.getMessage());

return -3; // Error reading object

}

}

}

catch (FileNotFoundException e) {

System.err.println("Error: File " + database\_filename + " not found.");

return -1;

}

catch (IOException e) {

System.err.println("Error: Unable to read file " + database\_filename + ".");

return -2;

}

return 0; // Success

}

**5.2 Diagram sekwencji metody Invoices:**

**save\_invoices\_database(String database\_filename) → Invoices class**

**A screenshot of a computer

Description automatically generated**

**5.2.1. Kod metody: save\_invoices\_database(String database\_filename) → Invoices class**

public int save\_invoices\_database(String database\_filename) {

// Try to open the database file for writing

try (ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream(database\_filename))) {

// Write all invoices to the file

for (Invoice invoice : invoices\_database) {

objectOutputStream.writeObject(invoice);

}

}

catch (FileNotFoundException e) {

System.err.println("Error: Cannot create file " + database\_filename);

return -4; // Unable to create file

}

catch (IOException e) {

System.err.println("Error: Unable to write to file " + database\_filename);

return -5; // Error writing to file

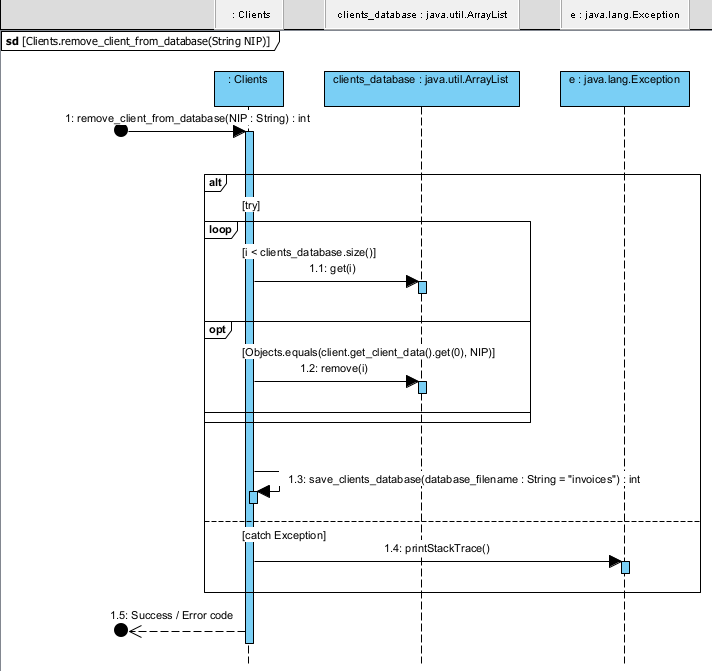
}

return 0; // Success

}

**5.3.Diagram sekwencji metody Clients:**

**remove\_client\_from\_database(NIP: String): Int → Clients class**

****

**5.3.1. Kod metody: remove\_client\_from\_database(NIP: String): Int → Clients class**

public int remove\_client\_from\_database(String NIP) {

try {

for (int i = 0; i < clients\_database.size(); i++) {

ClientProfile client = clients\_database.get(i);

if (Objects.equals(client.get\_client\_data().get(0), NIP)) {

// Remove the client from the database

clients\_database.remove(i);

}

}

// Save the updated client database

save\_clients\_database("invoices");

return 0; // Success

}

catch (Exception e) {

System.err.println("Error removing client from database");

e.printStackTrace();

return -1; // Failure

}

}

**5.4. Diagram sekwencji metody ClientProfile:**

**get\_client\_data() → ClientProfile class**

**A screenshot of a computer

Description automatically generated**

**5.4.1. Kod metody: get\_client\_data() → ClientProfile class**

public ArrayList<String> get\_client\_data() {

// Create an ArrayList to store the client data

ArrayList<String> clientData = new ArrayList<String>();

clientData.add("NIP: " + NIP);

clientData.add("Company name: " + company\_name);

clientData.add("Address: " + address);

clientData.add("Tel number: " + tel\_number);

clientData.add("Owner firstname: " + owner\_firstname);

clientData.add("Owner lastname: " + owner\_lastname);

// Print the client data

for (String dataItem : clientData) {

System.out.println(dataItem);

}

// Return the client data

return clientData;

}