

POLITECHNIKA WROCŁAWSKA

BAZY DANYCH 2

System firmy kurierskiej

Paweł Twardawa 235072
Justyna Zieniewicz 235735

prowadzący:
dr inż. Roman Ptak

Wrocław, 2018

Spis treści

1	Wstęp	3
1.1	Cel projektu	3
1.2	Zakres projektu	3
2	Analiza wymagań	3
2.1	Opis działania i schemat logiczny systemu	3
2.2	Wymagania funkcjonalne	4
2.2.1	Dla klienta:	4
2.2.2	Dla kuriera:	4
2.2.3	Dla magazyniera:	4
2.2.4	Dla administratora:	5
2.3	Wymagania нефункционалне	5
2.3.1	Wykorzystywane technologie i narzędzia	5
2.4	Wymagania dotyczące rozmiaru bazy danych	5
2.5	Wymagania dotyczące bezpieczeństwa systemu	5
3	Projekt systemu	5
3.1	Projekt bazy danych	6
3.1.1	Analiza rzeczywistości i uproszczony model konceptualny	6
3.1.2	Model logiczny i fizyczny oraz ograniczenia integralności danych	7
3.1.3	Inne elementy schematu – mechanizmy przetwarzania danych	7
3.1.4	Projekt mechanizmów bezpieczeństwa na poziomie bazy danych	8
3.2	Projekt aplikacji użytkownika	10
3.2.1	Architektura aplikacji i diagramy projektowe	10
3.3	Interfejs graficzny i struktura menu	10
3.4	Metoda podłączania do bazy danych – integracja z bazą danych	11
3.5	Projekt zabezpieczeń na poziomie aplikacji	12
4	Implementacja systemu bazy danych	12
4.1	Tworzenie tabel i definiowanie ograniczeń	12
4.2	Implementacja mechanizmów przetwarzania danych	12
4.3	Implementacja uprawnień i innych zabezpieczeń	14
4.4	Testowanie bazy danych na przykładowych danych	15
5	Implementacja i testy aplikacji	23
5.1	Proces tworzenia instalatora	23
5.2	Instalacja i konfigurowanie systemu	26
5.3	Instrukcja użytkownika aplikacji	30
5.3.1	Logowanie i tworzenie nowego konta	30

5.3.2	Instrukcja użytkowania aplikacji przez klienta	31
5.3.3	Instrukcja użytkowania aplikacji przez magazyniera . .	35
5.3.4	Instrukcja użytkowania dla kuriera	36
5.3.5	Instrukcja obsługi aplikacji przez administratora . . .	38
5.4	Testowanie opracowanych funkcji systemu	38
5.4.1	Testy manualne aplikacji	38
5.4.2	Testy jednostkowe	40
5.5	Omówienie wybranych rozwiązań programistycznych	42
5.5.1	Implementacja interfejsu dostępu do bazy danych . . .	43
5.5.2	Implementacja wybranych funkcjonalności systemu . .	45
5.5.3	Implementacja mechanizmów bezpieczeństwa	50
6	Podsumowanie i wnioski	52
	Literatura	53
	Spis rysunków	55

1 Wstęp

1.1 Cel projektu

Celem projektu było stworzenie bazy danych oraz aplikacji do obsługi firmy kurierskiej.

1.2 Zakres projektu

Zakresem projektu aplikacji do obsługi systemu firmy kurierskiej była aplikacja desktopowa, która dedykowana jest pracownikom i klientom firmy. Z punktu widzenia klienta, aplikacja umożliwia nadawanie i śledzenie przesyłki, edycję danych wysyłkowych i anulowanie transakcji. Klient korzystający z aplikacji będzie mógł, wprowadzając dane swoje, adresata oraz paczki, nadawać przesyłkę oraz, za pomocą unikatowego numeru przesyłki definowanego po pozytywnym nadaniu przesyłki, sprawdzać jej lokalizację. Magazynierzy oraz kurierzy, dla których aplikacja jest narzędziem pracy, będą mogli sprawdzić dane transportowanych przesyłek, adresy, pod które przesyłki mają trafić oraz zmienić status przesyłki np. po pomyślnym dostarczeniu przesyłki do adresata.

2 Analiza wymagań

Tematem projektu jest baza danych firmy kurierskiej. Pracowników firmy można podzielić na kurierów i magazynierów. Kurierów firma zatrudnia 15, natomiast magazynierów 5 oraz posiada jeden magazyn centralny. W celu skorzystania z usług firmy należy nadać przesyłkę przez aplikację. Kurierzy odbierają przesyłki od nadawców i dostarczają je do magazynu, w którym zostają posortowane, a magazynierzy przydzielają im identyfikatory kurierów odpowiedzialnych za ich dostarczenie do adresatów. Po dostarczeniu przesyłki pod adres odbiorcy kurierzy potwierdzają odebranie, bądź nieodebranie przesyłki, zmieniając jej status w aplikacji.

2.1 Opis działania i schemat logiczny systemu

Z aplikacji korzystać będzie trzech typów użytkowników: klient, kurier i magazynier.

Klient, aby korzystać z systemu użytkownik musi być zalogowany. Logowanie polega na wprowadzeniu loginu oraz hasła. Jeżeli użytkownika nie ma w bazie należy przejść przez proces rejestracji. Rejestracja polega na podaniu unikalnej nazwy użytkownika, hasła oraz adresu email. Aby umożliwić korzystanie ze wszystkich funkcjonalności systemu, adres email każdego klienta musi być potwierdzony. Klient ma możliwość nadawania nowych przesyłek, śledzenie nadanych oraz edycji danych (do pewnego momentu). Podczas nadawania

nowej przesyłki klient musi podać adres nadawcy, odbiorcy oraz dokładny wymiar paczki. W celu usprawnienia procesu nadawania, adres nadawcy może zostać zapisany. Po zaakceptowaniu nadania, system generuje unikalny numer przesyłki oraz dodaje wpis do bazy danych przypisując wygenerowany numer oraz przesyłkę do klienta. Śledzenie przesyłek realizowane jest poprzez podanie numeru przesyłki w odpowiednie pole aplikacji. Następnie system szuka przesyłki w bazie danych i zwraca ostatnią lokalizację przesyłki.

Magazynier ma wgląd do danych wszystkich przesyłek. Jego zadaniem jest posortowanie przesyłek które przybyły od nadawcy i przekazanie ich do kurierów. Po zalogowaniu do systemu widzi listę przesyłek wraz z ich adresami docelowymi.

Kurier ma wgląd do adresów odbiorcy przesyłek które aktualnie transportuje. Po wybraniu przesyłki pokazują się szczegółowe informacje o przesyłkach oraz formularz doręczenia. Można w nim zmienić status przesyłki na np. "nie doręczono". Jeżeli zostanie wybrana taka opcja należy podać również powód niedoręczenia.

Na potrzeby tworzenia kont pracowniczych, o typie użytkownika Courier i Storeman, stworzony został dodatkowo Administrator.

2.2 Wymagania funkcjonalne

2.2.1 Dla klienta:

- logowanie/rejestracja
- nadawanie przesyłek
- edycja danych przesyłek
- anulowanie nadania przesyłki
- śledzenie przesyłki

2.2.2 Dla kuriera:

- wyświetlanie danych dla aktualnie transportowanych przesyłek
- potwierdzenie odebrania przesyłki od klienta
- potwierdzenie dostarczenia przesyłki do klienta
- potwierdzenie niedostarczenia przesyłki pod wskazany adres

2.2.3 Dla magazyniera:

- nanoszenie w systemie informacji o zmianie położenia przesyłki w magazynie
- wyświetlanie danych wszystkich przesyłek

2.2.4 Dla administratora:

- tworzenie kont pracowników

2.3 Wymagania niefunkcjonalne

2.3.1 Wykorzystywane technologie i narzędzia

W projekcie będziemy korzystać z bazy danych MySQL, natomiast aplikacja zostanie napisana w języku Java. Stworzona zostanie desktopowa wersja aplikacji dla wszystkich użytkowników.

2.4 Wymagania dotyczące rozmiaru bazy danych

Baza danych będzie zawierać dziewięć tabel. W całej bazie będzie kilkadziesiąt rekordów. Liczba rekordów w bazie będzie się powiększać wraz z rozwojem firmy. Szacowana miesięczna ilość obsługiwanych paczek to 2000. Przyrost klientów z roku na rok to ok. 200. Wielkość bazy danych po roku działania firmy to ok. 13000 rekordów. Dane dotyczące transakcji przechowywane w bazie (tj. faktury, rachunki, historia przesyłek) będą przechowywane w bazie przez 5 lat. Dane klientów przechowywane będą przez cały czas istnienia bazy, chyba że klient sam wystąpi o ich usunięcie.

2.5 Wymagania dotyczące bezpieczeństwa systemu

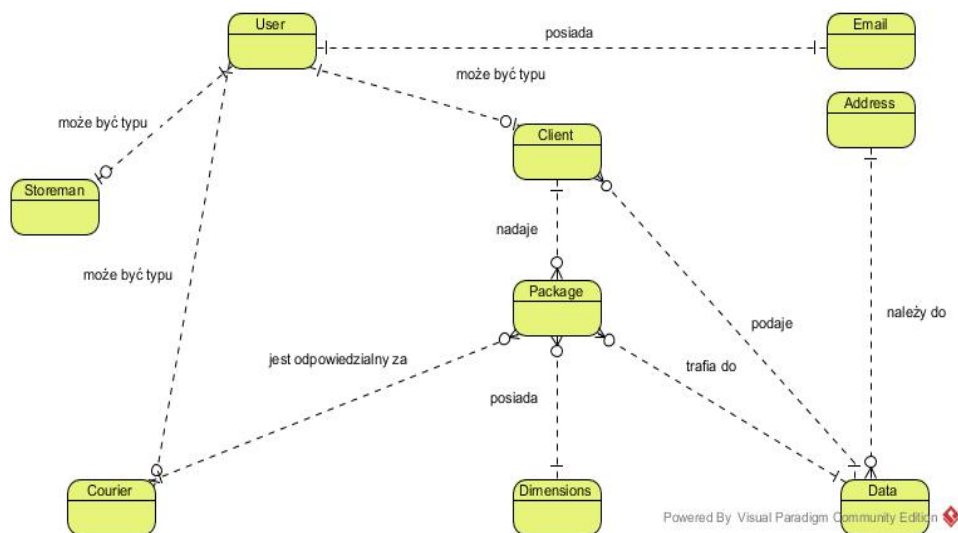
Każdy użytkownik systemu musi być w nim zarejestrowany (posiadać nazwę użytkownika oraz hasło), a przy każdym logowaniu do systemu uwierzytelnić użytkownika podając hasło.

3 Projekt systemu

Projekt i struktury bazy danych, mechanizmów zapewniania poprawności przechowywanych informacji, oraz kontroli dostępu do danych.

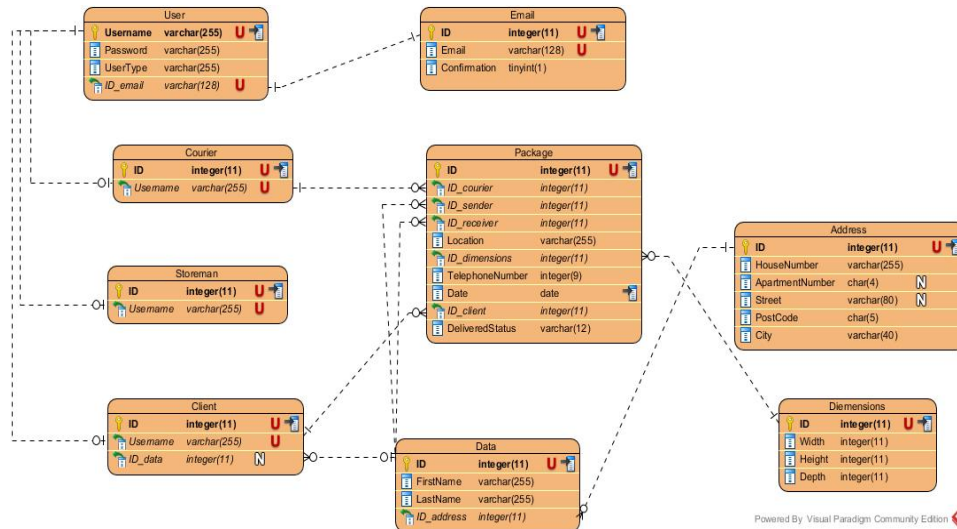
3.1 Projekt bazy danych

3.1.1 Analiza rzeczywistości i uproszczony model konceptualny



Rysunek 1: Model konceptualny bazy danych

3.1.2 Model logiczny i fizyczny oraz ograniczenia integralności danych



Rysunek 2: Model logiczny bazy danych, będący również modelem fizycznym

3.1.3 Inne elementy schematu – mechanizmy przetwarzania danych

Efekt zachowania kolejności numerowania kluczy głównych, jaki daje użycie sekwencji, zapewnia w naszej bazie funkcja `AUTO_INCREMENT`, którą zastosowaliśmy w tabelach: Address, Client, Courier, Data, Dimensions, Email, Package, Storeman przy wierszach nazywających się ID.

Wyzwalacze (trigger) wykorzystaliśmy do automatycznego dodawaniu użytkowników, rejestrujących się z określonym typem użytkownika, do tabel przynależnych danemu typowi (Przykład: UserType ustawiony jako Client, użytkownik po rejestracji zostanie dodany do tabeli Client). Kod dla przykładu wygląda następująco (pozostałe przypadki są analogiczne):

```

1 DELIMITER $$
2 CREATE TRIGGER add_memberships AFTER INSERT on User
3 BEGIN
4     IF (NEW.UserType = 'Client') THEN
5         INSERT INTO Client SET Username = NEW.Username ;
6     ELSEIF (NEW.UserType = 'Courier') THEN
7         INSERT INTO Courier SET Username = NEW.Username ;
  
```



```

8      ELSEIF (NEW.UserType = 'Storeman') THEN
9          INSERT INTO Storeman SET Username = NEW.Username ;
10     END IF;
11     END $$
12 DELIMITER ;

```

Indeksy zostały zastosowane w przypadkach tabel i kolumn, których wartość zostaje przeszukiwana, a ilość rekordów w tabelach zapewnia długi czas wyszukiwania wyniku. Należą do nich: Address - ID, User - Username, Package - ID, Courier - ID, Data - ID, Client - ID, Storeman - ID, Package - Date, Email - ID, Dimensions - ID.

Wykorzystanie widoków zapewnia wydajniejszy dostęp do wyświetlania danych. Zostały stworzone dwa widoki: CourierData i StoremanData, których struktura kolumn została zaprezentowana poniżej. Widok StoremanData łączy w sobie dane z tabel Address i Courier, co ułatwia magazynierowi przydzielanie paczek do poszczególnych kurierów. Widokiem dedykowanym kurierowi jest widok CourierData, który jest połączeniem tabel Package, Data i Address i ma usprawnić wyszukiwanie danych odbiorcy. Widok dla kuriera został stworzony poprzez dodanie następującego kodu:

```

1 CREATE VIEW CourierData
2 AS select Package.ID_courier, Package.ID, Package.ID_client,
3 Package.TelephoneNumber, r.FirstName
4 AS 'ReceiverFirstName', r.LastName
5 AS 'ReceiverLastName', ra.City
6 AS 'ReceiverCity', ra.PostCode
7 AS 'ReceiverPostCode', ra.Street
8 AS 'ReceiverStreet', ra.HouseNumber
9 AS 'ReceiverHouseNumber', ra.ApartmentNumber
10 AS 'ReceiverApartmentNumber'
11 FROM Package, (Data r JOIN Address ra ON r.ID_address = ra.ID)
12 where Package.ID_receiver = r.ID

```

Kod dla widoku magazyniera:

```

1 CREATE VIEW StoremanData
2 AS SELECT p.ID, p.ID_courier, a.City, a.PostCode, a.Street, a.HouseNumber
3 FROM Package p
4 JOIN( Data d JOIN Address a on d.ID_address = a.ID )on p.ID_receiver = d.ID

```

3.1.4 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

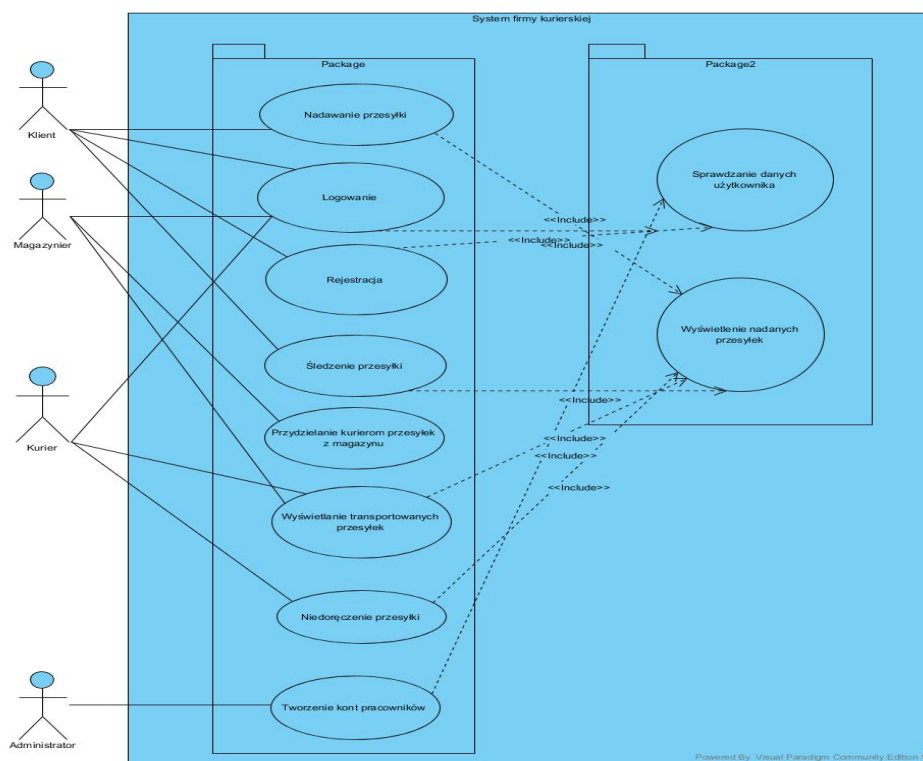
Mechanizmy bezpieczeństwa na poziomie bazy polegają na ograniczeniu dostępu do przeglądania i edycji danych znajdujących się w systemie w zależ-

ności od typu użytkownika. Żaden użytkownik nie ma prawa do usuwania poszczególnych tabel oraz kolumn. Każdy użytkownik ma prawo do zmiany hasła oraz adresu E-mail. Pozostałe własności zostały przedstawione w tabeli poniżej.

	Klient	Kurier	Magazynier
Address	zapis/edycja/odczyt	odczyt	odczyt
Client	zapis/odczyt	brak	brak
Courier	brak	zapis/odczyt	odczyt
Data	zapis/edycja/odczyt	odczyt	odczyt
Dimensions	zapis/edycja/odczyt	brak	odczyt
Email	zapis/odczyt	zapis/odczyt	zapis/odczyt
Package	zapis/edycja/odczyt	edycja/odczyt	edycja/odczyt
Storeman	brak	brak	zapis/odczyt
User	zapis/edycja/odczyt	zapis/edycja/odczyt	zapis/edycja/odczyt

3.2 Projekt aplikacji użytkownika

3.2.1 Architektura aplikacji i diagramy projektowe



Rysunek 3: Diagram przypadków użycia dla trzech aktorów modelu

3.3 Interfejs graficzny i struktura menu

Aplikacja nie jest rozbudowana graficznie, ale jej prosty wygląd sprawia, że korzystanie z niej jest intuicyjne. Stworzone zostały okienka spełniające przypadki użycia Rejestracja i Logowanie. Rejestracja potrzebna jest tylko klientom. Po poprawnym uzupełnieniu danych w okienku Registry do bazy zostaje dodany nowy użytkownik. Nazwa użytkownika musi być unikalna. Hasło i potwierdzenie hasła musi zawierać ten sam ciąg znaków. Adres Email musi zawierać znak @ oraz nazwę domeny znajdującą się po nim.

Rysunek 4: Okienko rejestracji

Korzystając z okienka Logowania, użytkownik może zalogować się do swojego konta. Podaje on nazwę użytkownika oraz hasło, dane te zostają sprawdzone w bazie, a po ich rozpoznaniu użytkownik zostaje zalogowany.

Rysunek 5: Okienko logowania

Po zalogowaniu, w zależności od typu użytkownika, okna aplikacji różnią się od siebie. Ich struktura została szczegółowo opisana poniżej, w punkcie Instrukcja użytkowania aplikacji.

3.4 Metoda podłączania do bazy danych – integracja z bazą danych

Do podłączania bazy danych użyte zostało łącze JDBC (Java DataBase Connectivity).

3.5 Projekt zabezpieczeń na poziomie aplikacji

Aby uzyskać dostęp do aplikacji należy stworzyć konto oraz potwierdzić adres mailowy. Jedynie zalogowany użytkownik ma dostęp do funkcjonalności systemu. Aplikacja łączy się z bazą danych w celu weryfikacji podanych danych. Hasła są szyfrowane przy użyciu SHA - 256.

4 Implementacja systemu bazy danych

4.1 Tworzenie tabel i definiowanie ograniczeń

Przy tworzeniu bazy danych skorzystaliśmy z podejścia model-first oraz systemu bazodanowego MySQL.

Przy tworzeniu poszczególnych tabel zastosowaliśmy pewne ograniczenia. W przypadku wszystkich tabel, ich ID, będące kluczem głównym, musi być unikalne. W przypadku tabel Client, Courier, Storeman oraz User nazwa użytkownika musi być unikalna. W tabeli Address przy kolumnach Street oraz ApartmentNumber dopuszcza się pozostawienie pustych pól. W tabeli Client ID_data może być puste. W tabeli Email wprowadzony adres Email musi być unikalny. W tabeli Package status dostarczenia DeliveredStatus może być puste. W tabeli User ID_email może być puste.

4.2 Implementacja mechanizmów przetwarzania danych

Zostały stworzone następujące widoki: ClientHistory, CourierData, StoremanData.

Widok dla kuriera CourierData:

```
1 CREATE VIEW CourierData AS
2 SELECT Package.ID_courier, Package.ID, Package.ID_client, Package.TelephoneNumber,
3 r.FirstName AS 'ReceiverFirstName',
4 r.LastName AS 'ReceiverLastName',
5 ra.City AS 'ReceiverCity',
6 ra.PostCode AS 'ReceiverPostCode',
7 ra.Street AS 'ReceiverStreet',
8 ra.HouseNumber AS 'ReceiverHouseNumber',
9 ra.ApartmentNumber AS 'ReceiverApartmentNumber'
10 FROM Package, (Data r JOIN Address ra ON r.ID_address = ra.ID)
11 WHERE Package.ID_receiver = r.ID
```

Widok dla magazyniera StoremanData

```
1 CREATE VIEW StoremanData AS
2 SELECT p.ID, p.ID_courier, a.City, a.PostCode, a.Street, a.HouseNumber
3 FROM Package p JOIN( Data d JOIN Address a on d.ID_address = a.ID )
4 on p.ID_receiver = d.ID
```

Widok dla klienta ClientHistory

```
1 CREATE VIEW ClientHistory AS
2 SELECT Package.ID, Package.Location, Package.ID_client, Package.TelephoneNumber,
3 s.FirstName AS 'senderFirstName',
4 s.LastName AS 'SenderLastName',
5 sa.City AS 'SenderCity',
6 sa.PostCode AS 'SenderPostCode',
7 sa.Street AS 'SenderStreet',
8 sa.HouseNumber AS 'SenderHouseNumber',
9 sa.ApartmentNumber AS 'SenderApartmentNumber',
10 r.FirstName AS 'ReceiverFirstName',
11 r.LastName AS 'ReceiverLastName',
12 ra.City AS 'ReceiverCity',
13 ra.PostCode AS 'ReceiverPostCode',
14 ra.Street AS 'ReceiverStreet',
15 ra.HouseNumber AS 'ReceiverHouseNumber',
16 ra.ApartmentNumber AS 'ReceiverApartmentNumber'
17 FROM Package,
18 ( Data s JOIN Address sa ON s.ID_address = sa.ID ),
19 (Data r JOIN Address ra ON r.ID_address = ra.ID)
20 WHERE Package.ID_sender = s.ID
21 AND Package.ID_receiver = r.ID
```

Stworzony został jeden wyzwalacz, który wykorzystaliśmy do automatycznego dodawania użytkowników, rejestrujących się z określonym typem użytkownika, do tabel przynależnych danemu typowi (Przykład: UserType ustawiony jako Client, użytkownik po rejestracji zostanie dodany do tabeli Client).

```
1 BEGIN
2     IF (NEW.UserType = 'Client') THEN
3         INSERT INTO Client SET Username = NEW.Username ;
4     ELSEIF (NEW.UserType = 'Courier') THEN
5         INSERT INTO Courier SET Username = NEW.Username ;
6     ELSEIF (NEW.UserType = 'Storeman') THEN
7         INSERT INTO Storeman SET Username = NEW.Username ;
8     END IF;
9     END
```

4.3 Implementacja uprawnień i innych zabezpieczeń

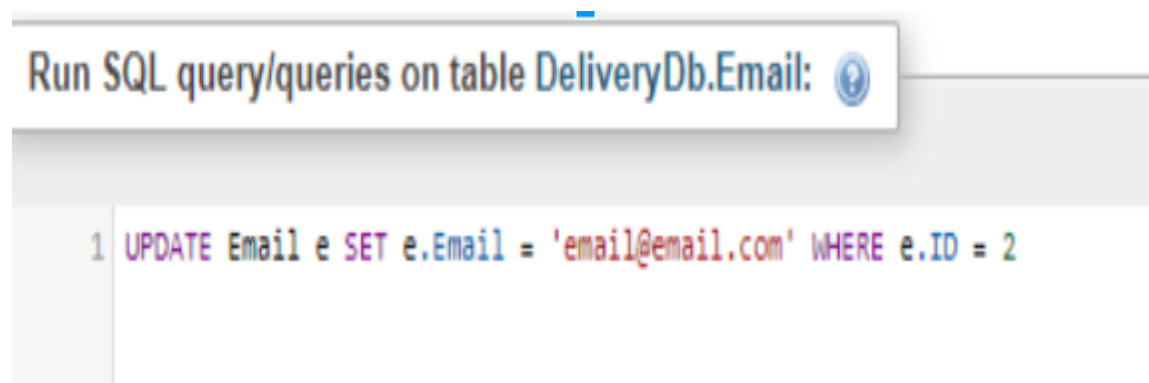
	DeliveryDbUserClient	DeliveryDbUserCourier	DeliveryDbUserStoreman	DeliveryDbUserLogin	DeliveryDbUser
Address	SELECT, INSERT, UPDATE	BRAK	BRAK	BRAK	BRAK
Client	SELECT, UPDATE	BRAK	BRAK	SELECT	BRAK
Courier	SELECT	SELECT	SELECT	SELECT	BRAK
Data	SELECT, INSERT, UPDATE	BRAK	BRAK	BRAK	BRAK
Dimensions	SELECT, INSERT	BRAK	BRAK	BRAK	BRAK
Email	BRAK	BRAK	BRAK	SELECT, INSERT	SELECT, INSERT
Package	SELECT, INSERT, UPDATE, DELETE	SELECT, UPDATE	BRAK	BRAK	BRAK
Storeman	BRAK	BRAK	SELECT	SELECT	BRAK
User	SELECT	SELECT	SELECT	SELECT, INSERT	SELECT, INSERT
ClientHistory	SELECT, UPDATE	BRAK	BRAK	BRAK	BRAK
CourierData	BRAK	SELECT	BRAK	BRAK	BRAK
StoremanData	BRAK	BRAK	SELECT, UPDATE	BRAK	BRAK

Rysunek 6: Tabela uprawnień dostępu do bazy danych

4.4 Testowanie bazy danych na przykładowych danych

Baza danych została poddana różnym testom, zarówno takim, które miała przejść pozytywnie, jak i takim, w których powinna uniemożliwić wykonanie niedozwolonej operacji.

Testom poddana została próba aktualizacji danych przez użytkownika bez dostępu do takich funkcji. Zakazaną czynność wychwyciły zabezpieczenia bazy, a zwrócony komunikat błędu został zaprezentowany poniżej.

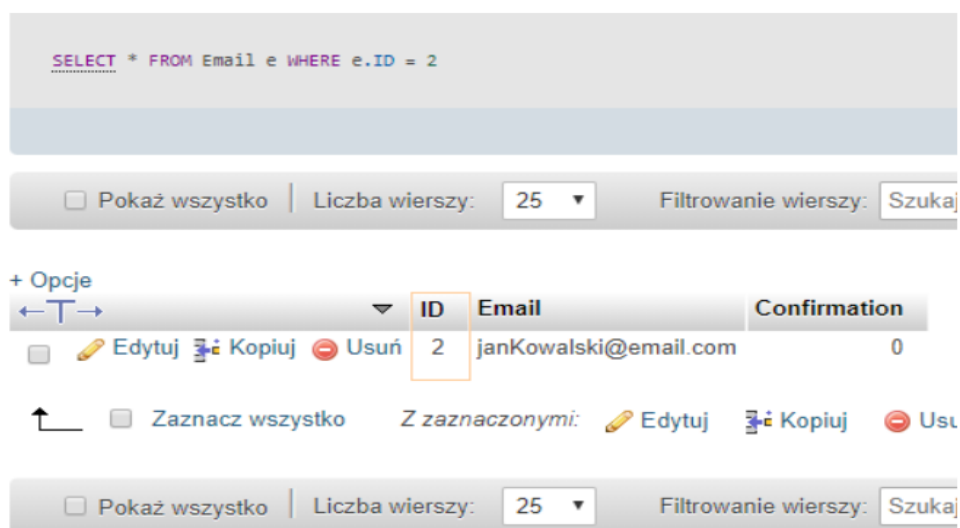


Rysunek 7: Próba aktualizacji adresu E-mail przez nieupoważnioną osobę



Rysunek 8: Brak dostępu użytkownika DeliveryDbUserLogin do aktualizacji danych w tabeli Email

Próba wyświetlenia zawartości bazy przez osobę do tego upoważnioną zakończyła się sukcesem. Nie spowodowało to błędu, a reakcją bazy na to działanie było wyświetlenie zawartości.

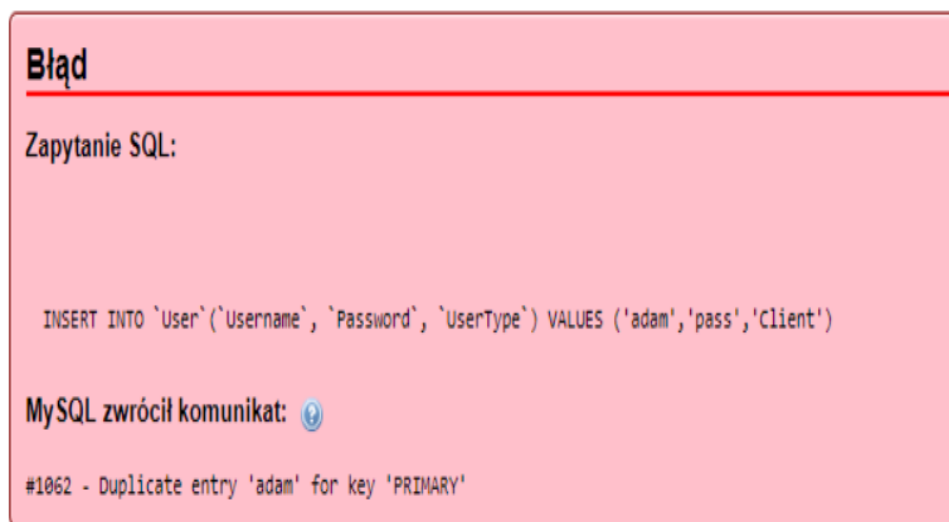


Rysunek 9: Użytkownik DeliveryDbUserLogin ma dostęp do wyświetlania danych z tabeli Email

Niektóre dane użytkowników, takie jak login, muszą być unikalne, a próba dodania danych istniejących w bazie nie może zakończyć się pomyślnie.



Rysunek 10: Wprowadzenie istniejących danych do kolumny z ograniczeniem unique



Rysunek 11: Wychwycenie próby dodania istniejących danych

Poniżej zaprezentowane zostało testowanie wyzwalaczy, które automatyzują dodawanie nowych użytkowników do tabel przeznaczonych pod różne typy użytkowników (np. Courier), po wprowadzeniu w tabeli User poprawnego typu użytkownika.

`SELECT * FROM `storeman``

☐ Pokaż wszystko | Liczba wierszy: 25 ▼ Filt

Sortuj wg klucza: Żaden ▼

+ Opcje

					ID	Username
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	2	mag1	
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	1	magazynier	
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	3	magazynier-zdziś	
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	4	TestUser	

Rysunek 12: Tabela Storeman przed dodaniem użytkownika

✓ Wstawionych rekordów: 1. (Wykonanie zapytania trwało 0.0032 sekund(y).)

```
INSERT INTO `User`(`Username`, `Password`, `UserType`) VALUES ('Janusz', 'pass', 'Storeman')
```

Rysunek 13: Dodawanie użytkownika “Janusz” do tabeli User

Efektem działania triggera było automatyczne dodanie użytkownika do tabeli Storeman, ponieważ pole UserType w tabeli User było równe “Storeman”.

```
SELECT * FROM `storeman`
```

☐ Pokaż wszystko | Liczba wierszy: 25 ▼ Filtro

Sortuj wg klucza: Żaden ▼

+ Opcje

				ID	Username
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	5	Janusz
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	2	mag1
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	1	magazynier
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	3	magazynier-zdziś
<input type="checkbox"/>	Edytuj	Kopiuj	Usuń	4	TestUser

Rysunek 14: Tabela Storeman po dodaniu użytkownika i poprawnym wykonaniu triggera

Wyświetlanie widoków zarezerwowane jest tylko dla upoważnionych użytkowników. Próba wyświetlenia przez kogoś niepowołanego skutkuje pojawieniem się błędu.



Rysunek 15: Próba wyświetlania danych z widoku CourierData bez uprawnień

W przypadku użytkownika, dla którego wskazane jest mieć dostęp do widoku, próba wyświetlenia przechodzi pozytywnie, co pokazuje poniższy zrzut ekranu.

✓ Pokazano wiersze 0 - ... (Wykonanie zapytania trwało 0.0014 sekund(y).)

`SELECT * FROM ClientHistory`

☐ Pokaż wszystko | Liczba wierszy: 25 | Filtrowanie wierszy: Szukaj w tej tabeli

+ Opcje	ID	Location	DeliveredStatus	ID_client	TelephoneNumber
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	1	PowrotDoMagazynu	undelivered	1	787878787
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	2	OdebranoOdNadawcy	pickedUp	2	565656565
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	6	Odebrano od nadawcy	pickedUp	3	123456789
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	7	Powrot do magazynu	undelivered	3	123456789
<input type="checkbox"/> Edytuj <input type="checkbox"/> Kopiuj <input type="checkbox"/> Usuń	10	PowrotDoMagazynu	undelivered	3	123456789

Rysunek 16: Efekt wyświetlania danych z widoku ClientHistory z poziomu użytkownika DeliveryDbUserClient

Ważnym zabezpieczeniem przed utratą informacji z bazy jest ograniczenie dostępu do usuwania danych z bazy. Przeprowadzony test nie pozwolił na wykonanie zapytania DELETE użytkownikowi bez uprawnień.

Błąd

Zapytanie SQL:

```
DELETE FROM User WHERE Username = 'dawid'
```

MySQL zwrócił komunikat:

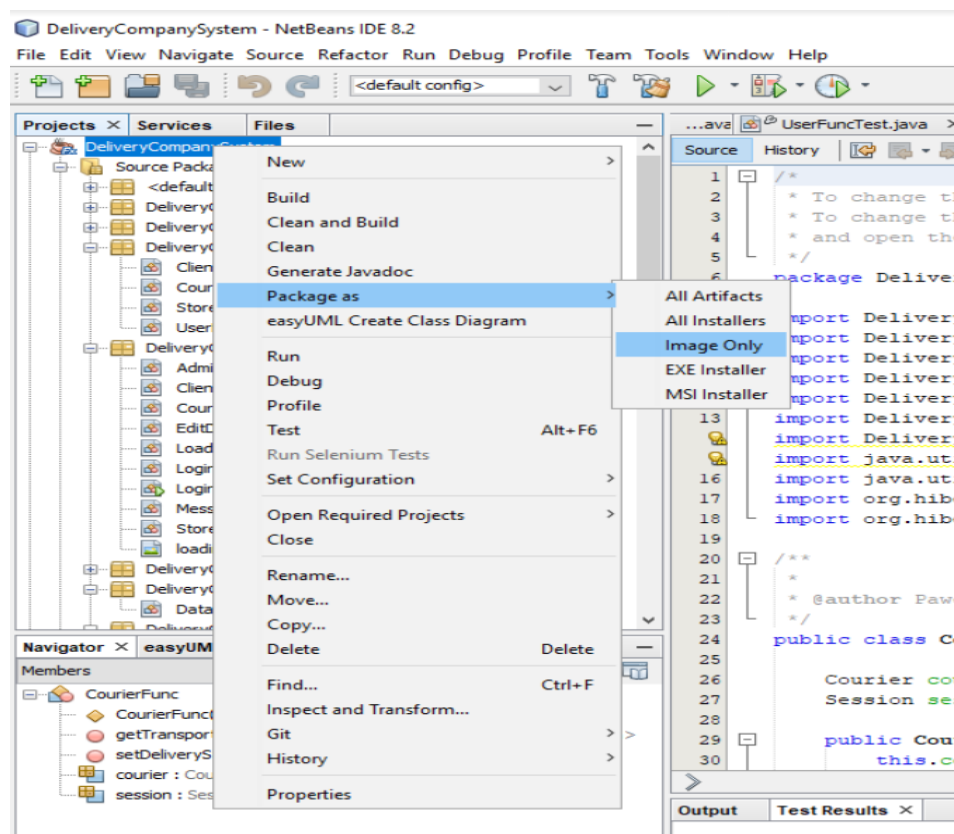
```
#1142 - DELETE command denied to user 'DeliveryDbUserLogin'@'localhost' for table 'User'
```

Rysunek 17: Próba usunięcia rekordu z tabeli User bez posiadania uprawnień

5 Implementacja i testy aplikacji

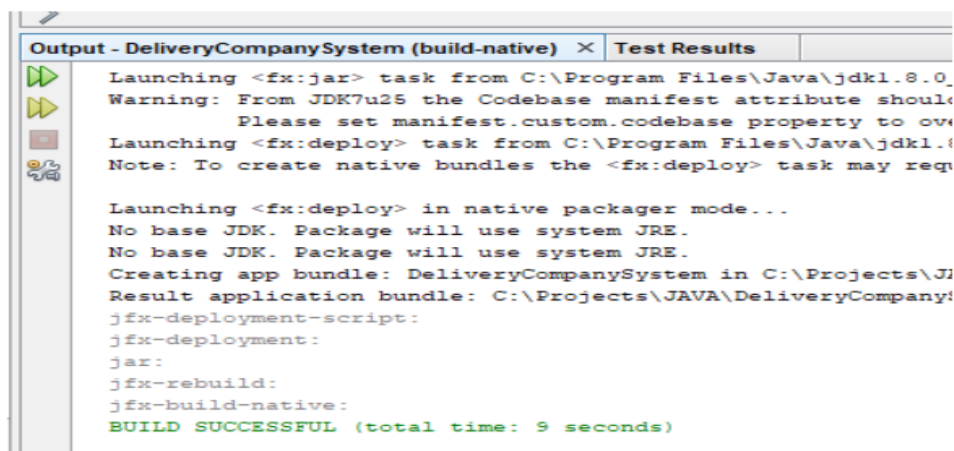
5.1 Proces tworzenia instalatora

Przed rozpoczęciem procesu tworzenia instalatora, w IDE należy spakować wybrany projekt jako obraz.

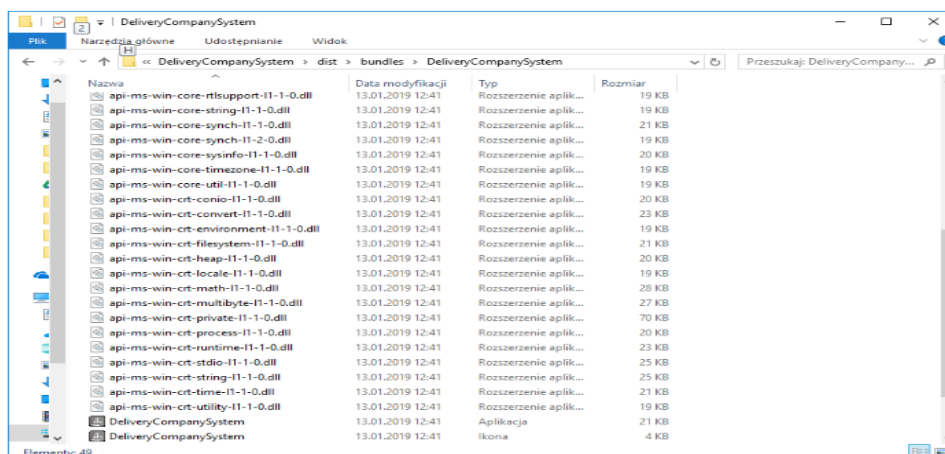


Rysunek 18: Pakowanie projektu jako obrazu

Po poprawnym zbudowaniu oraz wyeksportowaniu aplikacji w sekcji Output pojawi się komunikat "BUILD SUCCESSFUL", a w folderze projektu pojawi się aplikacja z rozszerzeniem .exe wraz z pozostałymi plikami potrzebnymi do poprawnej pracy programu.



Rysunek 19: Komunikat potwierdzający poprawne wyeksportowanie



Rysunek 20: Zawartość folderu projektu

Instalator został stworzony za pomocą programu Inno Setup Compiler. Na początku należy stworzyć Inno Script. Skrypt można stworzyć na dwa sposoby, przez kreator lub pisząc skrypt ręcznie. Poprawnie stworzony skrypt wygląda następująco:

```

; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

```

```

#define MyAppName "DeliverySystem"
#define MyAppVersion "1.1"

```

```

#define MyAppPublisher "Pawel Twardawa"
#define MyAppURL "http://www.example.com/"
#define MyAppExeName "DeliveryCompanySystem.exe"

[Setup]
; NOTE: The value of AppId uniquely identifies this application.
; Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{D29C611F-CFFA-4026-8D80-603A6A16B72A}}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
; AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
DisableProgramGroupPage=yes
LicenseFile=C:\Projects\JAVA\DeliveryCompanySystem
\DeliveryCompanySystem\license.txt
OutputBaseFilename=DeliverySystem
Compression=lzma
SolidCompression=yes

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription:
"{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "C:\Projects\JAVA\DeliveryCompanySystem\DeliveryCompanySystem\dist\bundles
\DeliveryCompanySystem\DeliveryCompanySystem.exe"; DestDir: "{app}";
Flags: ignoreversion
Source: "C:\Projects\JAVA\DeliveryCompanySystem\DeliveryCompanySystem\dist\bundles
\DeliveryCompanySystem\*"; DestDir: "{app}"; Flags: ignoreversion recursesubdirs
createallsubdirs; NOTE: Don't use "Flags: ignoreversion" on any shared system files

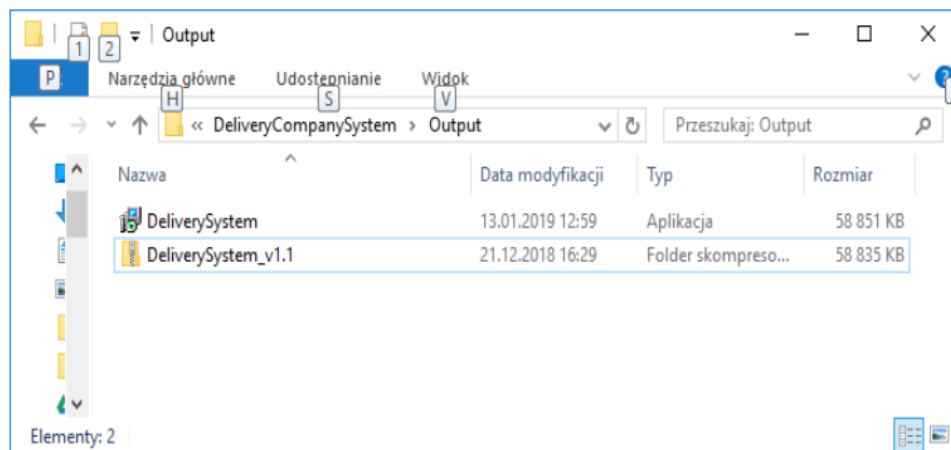
[Icons]
Name: "{commonprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}";
Tasks: desktopicon

```

[Run]

```
Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,  
{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait postinstall skipifsilent
```

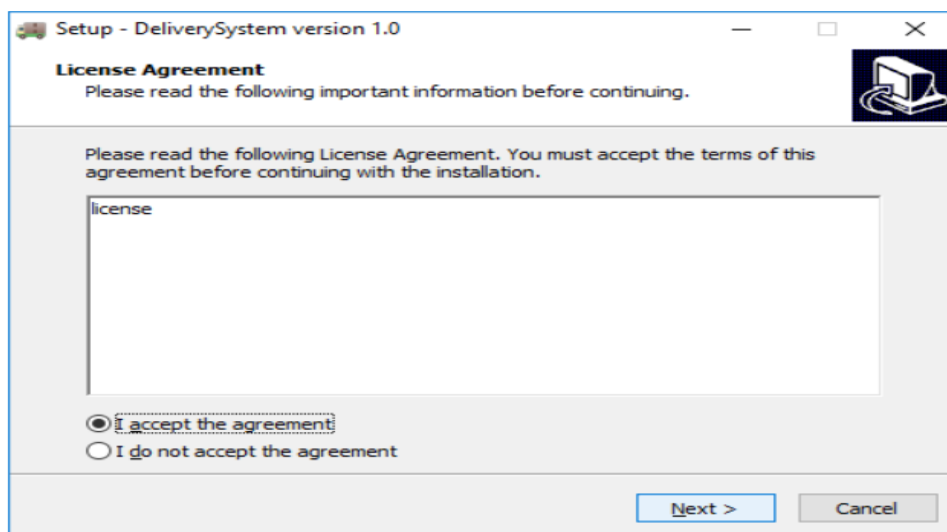
Po stworzeniu skryptu przechodzimy do menu Build i klikamy przycisk Compile lub Ctrl + F9. Po poprawnym stworzeniu instalatora w sekcji Compiler Output pojawi się napis “Finished” natomiast w projekcie, w katalogu Output pojawi się instalator aplikacji.



Rysunek 21: Instalator aplikacji w katalogu Output

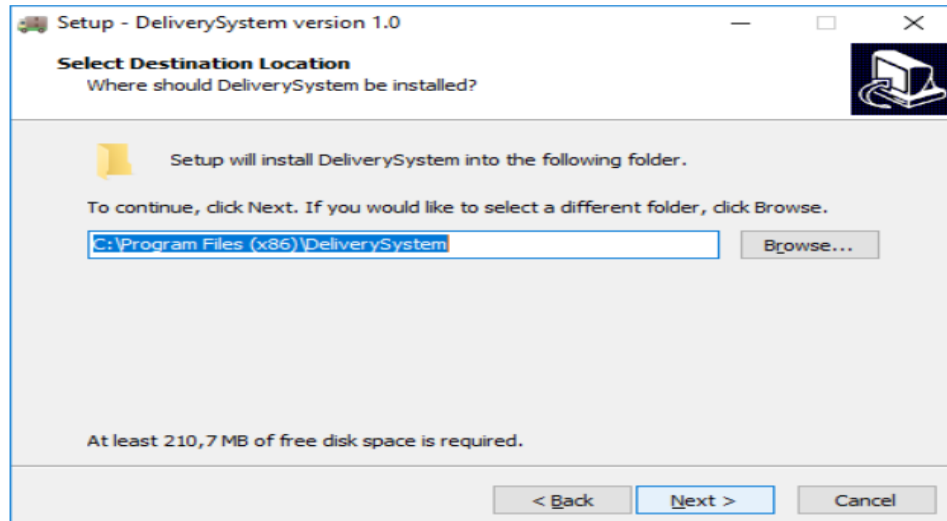
5.2 Instalacja i konfigurowanie systemu

W celu korzystania z programu należy go pobrać i rozpakować. Dwukrotne kliknięcie w wypakowaną ikonkę uruchamia instalator. Należy zaakceptować licencję następnie nacisnąć przycisk Next.



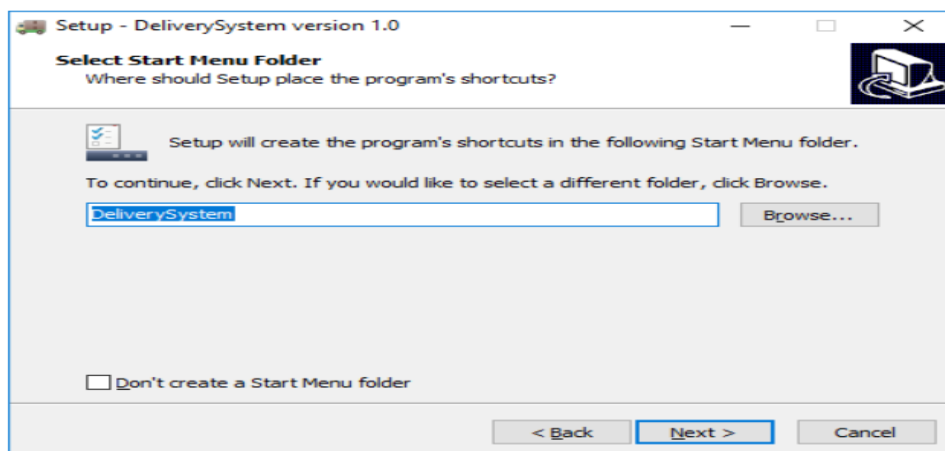
Rysunek 22: Akceptacja licencji

W następnym kroku wybieramy folder w którym program ma się zainstalować.



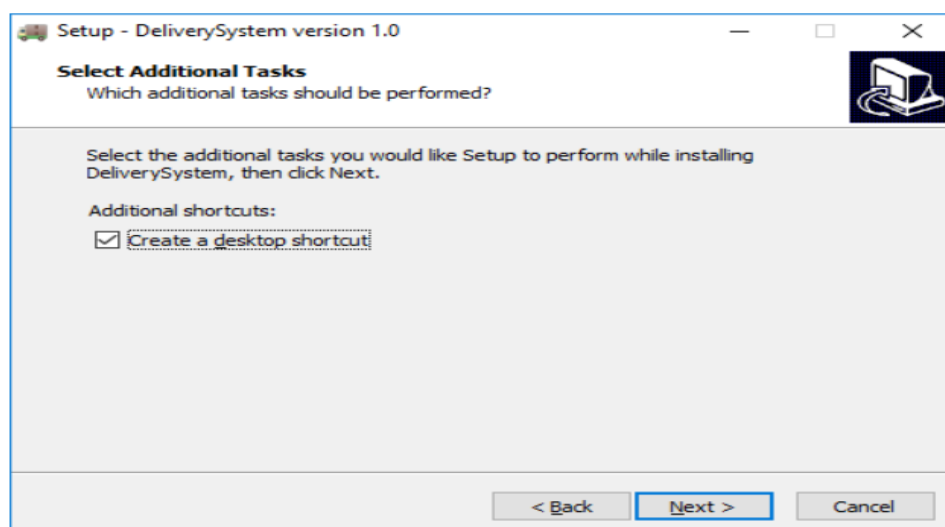
Rysunek 23: Wybór lokalizacji

Wybieramy nazwę folderu w którym program ma się zainstalować. Istnieje możliwość nie dodania skrótu do Menu Start.



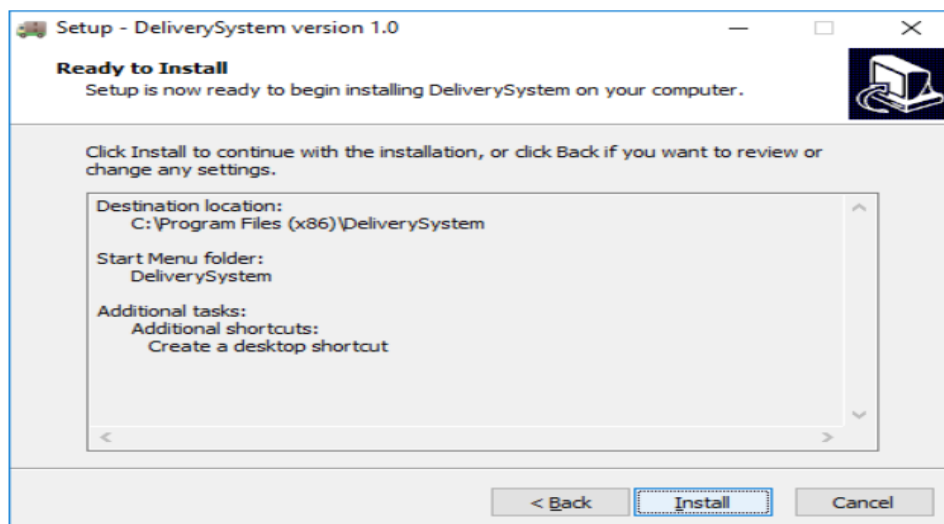
Rysunek 24: Wybór folderu, w którym ma się zainstalować

W tym oknie wybieramy czy chcemy utworzyć skrót na pulpicie.

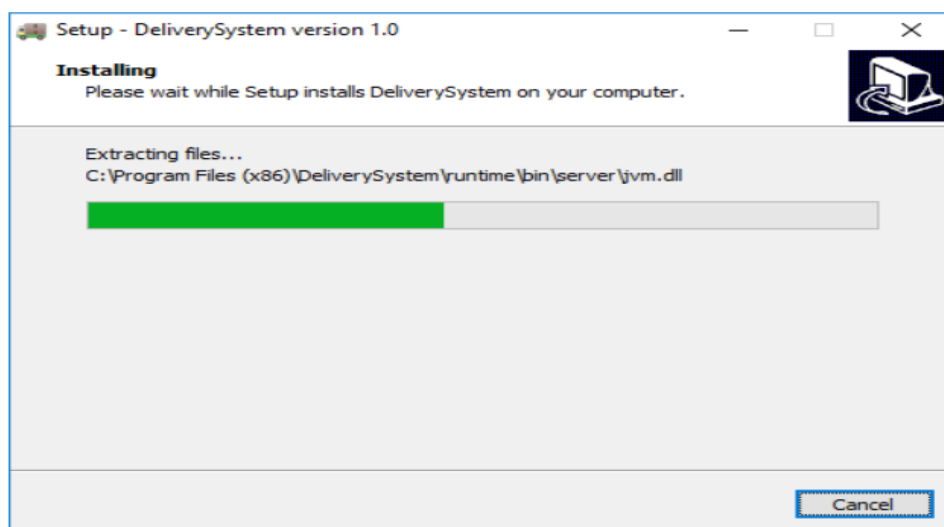


Rysunek 25: Tworzenie skrótu na pulpicie

Naciskamy Install.

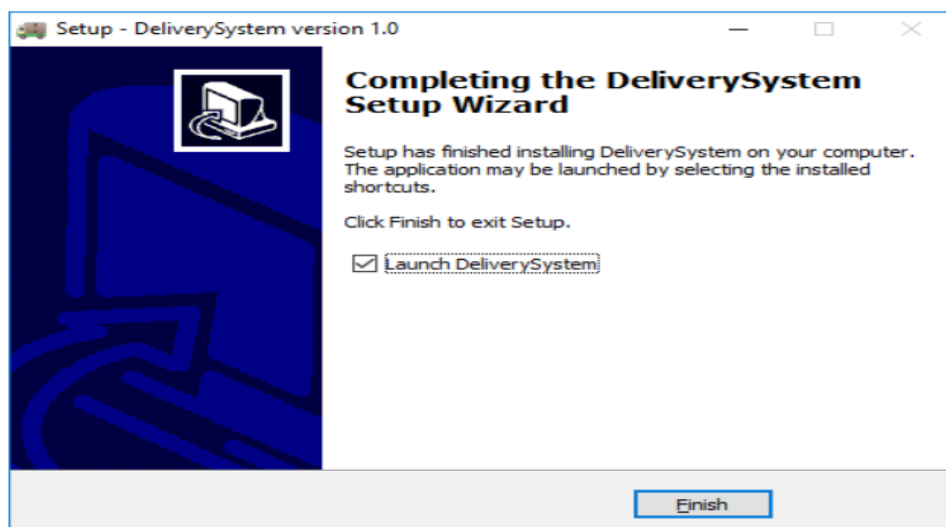


Rysunek 26: Rozpoczęcie instalacji



Rysunek 27: Instalacja w toku

Po poprawnym procesie instalacji oraz zaznaczeniu opcji “Launch DeliverySystem” program powinien się uruchomić.

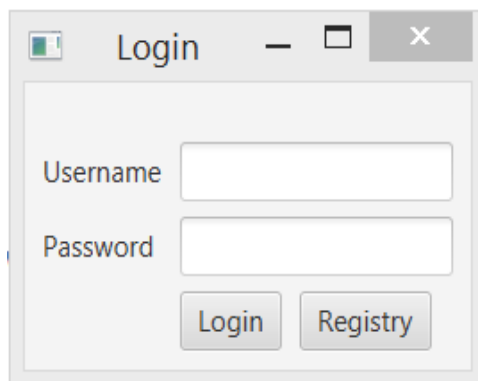


Rysunek 28: Zakończenie instalacji

5.3 Instrukcja użytkowania aplikacji

5.3.1 Logowanie i tworzenie nowego konta

Po uruchomieniu aplikacji pojawia się okienko logowania użytkownika.



Rysunek 29: Okienko startowe programu

Niezarejestrowani klienci powinni stworzyć nowe konto, klikając przycisk Registry i uzupełniając formularz rejestracyjny. W imieniu pracowników konta zakłada Administrator.

The 'Registry' window contains the following fields and controls:

- Username:** A text input field containing the text 'username'.
- Password:** A password input field represented by seven dots.
- Confirm Password:** A password input field represented by seven dots.
- Email:** A text input field containing the email address 'username@gmail.com'.
- Registry:** A button located below the email field.

Rysunek 30: Okienko tworzenia nowego konta

W procesie rejestracji klienci muszą wypełnić wszystkie powyższe pola. Username nie może już istnieć w bazie. Pole Password i Confirm Password muszą zawierać jednakowe hasła. Adres E-mail musi być poprawny.

Po zarejestrowaniu użytkownik może zalogować się na swoje konto podając swój unikalny login i poprawne hasło.

The 'Login' window contains the following fields and controls:

- Username:** A text input field containing the text 'username'.
- Password:** A password input field represented by seven dots.
- Login:** A button located below the password field.
- Registry:** A button located to the right of the Login button.

Rysunek 31: Logowanie

5.3.2 Instrukcja użytkowania aplikacji przez klienta

Po zalogowaniu na konto klienta wyświetli się okno nadawania nowej przesyłki. Wypełnienie wszystkich pól (oprócz Street i Apartment number) jest obligatoryjne. Zatwierdzenia nadania następuje przez kliknięcie przycisku Submit.

Client: daniel

New package | Follow | Sending history | My account | Log out

Sender

First name: Dawid

Last name: Greg

City: Warszawa

Post code: 33-560

Street: Uliczna

House number: 73

Apartment number: 1

Receiver

First name:

Last name:

Telephone number:

City:

Post code:

Width:

Height:

Depth:

Street:

House number:

Apartment number:

☐ Use different data

Submit

Rysunek 32: Okno nadawania przesyłki

Zakładka Follow umożliwia śledzenie lokalizacji nadanej przesyłki, poprzez wprowadzenie unikalnego numeru paczki.

Client: daniel

New package | Follow | Sending history | My account | Log out

Package number: enter number

Find

Rysunek 33: Zakładka śledzenia przesyłki

Zakładka Sending history zawiera informacje o wszystkich nadanych przesyłkach. Po zaznaczeniu wiersza oraz naciśnięciu prawego przycisku myszy pokazuje się menu z dwiema opcjami: Cancel package oraz Edit data. Opcje zrezygnowania z nadania lub edycji danych będą działać jeżeli paczka nie została jeszcze odebrana od nadawcy.

Client: daniel

New package Follow **Sending history** My account Log out

ID_client	ID	TelephoneNumber	Location	DeliveredStatus	senderFirstName	senderLastName	senderCity	SenderPos
4	12	333	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-678
4	13	324	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	14	546	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-678
4	16	3453	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	17	75675	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	19	4734	Doreczono	delivered	Adam	Greg	Warszawa	33-678
4	20	435	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	21	56	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-678
4	22	345	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	23	44	Do odebrania od nadawcy	toDelivery	Marek	Greg	Warszawa	33-678
4	24	23543	Do odebrania od nadawcy	toPickUp	Janusz	Greg	Warszawa	33-678
4	28	123456789	Do odebrania od nadawcy	toPickUp	Marek	Greg	Warszawa	33-678
4	29	123456789	Do odebrania od nadawcy	toPickUp	Marek	Greg	Warszawa	33-678

Rysunek 34: Zakładka historii przesyłek

Client: daniel

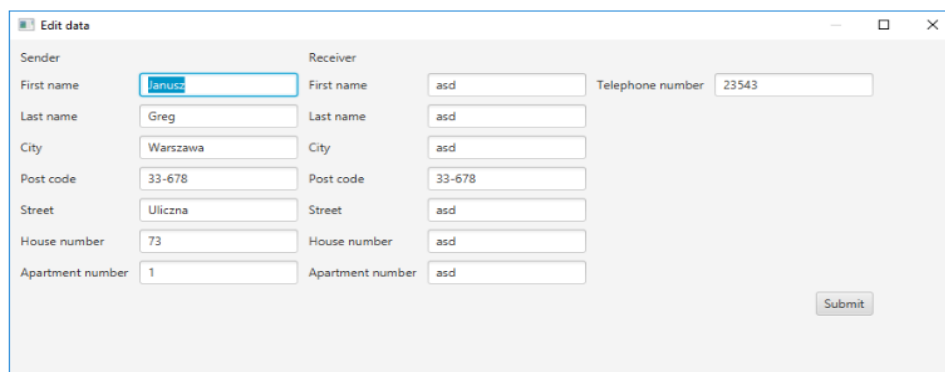
New package Follow **Sending history** My account Log out

ID_client	ID	TelephoneNumber	Location	DeliveredStatus	senderFirstName	senderLastName	senderCity	SenderPos
4	12	333	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-678
4	13	324	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	14	546	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-678
4	16	3453	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	17	75675	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	19	4734	Doreczono	delivered	Adam	Greg	Warszawa	33-678
4	20	435	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	21	56	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-678
4	22	345	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-678
4	23	44	Do odebrania od nadawcy	toDelivery	Marek	Greg	Warszawa	33-678
4	24	23543	Do odebrania od nadawcy	toPickUp	Janusz	Greg	Warszawa	33-678
4	28	123456789	Do odebrania od nadawcy	toPickUp	Marek	Greg	Warszawa	33-678
4	29	123456789	Do odebrania od nadawcy	toPickUp	Marek	Greg	Warszawa	33-678

Edit data
Cancel package

Rysunek 35: Rezultat kliknięcia prawym przyciskiem myszy na wybrany rząd tabeli

Po wciśnięciu Edit data wyświetli się okienko edycji danych. Zmianę należy zatwierdzić przyciskiem Submit.

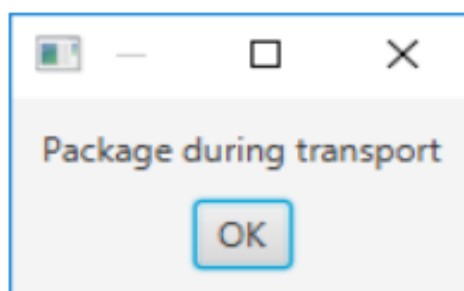


The 'Edit data' window contains two main sections: 'Sender' and 'Receiver'. The 'Sender' section has fields for First name (Janusz), Last name (Greg), City (Warszawa), Post code (33-678), Street (Uliczna), House number (73), and Apartment number (1). The 'Receiver' section has fields for First name (asd), Last name (asd), City (asd), Post code (33-678), Street (asd), House number (asd), and Apartment number (asd). A 'Telephone number' field with the value 23543 is also present. A 'Submit' button is located at the bottom right.

Sender		Receiver		Telephone number
First name	Janusz	First name	asd	23543
Last name	Greg	Last name	asd	
City	Warszawa	City	asd	
Post code	33-678	Post code	33-678	
Street	Uliczna	Street	asd	
House number	73	House number	asd	
Apartment number	1	Apartment number	asd	

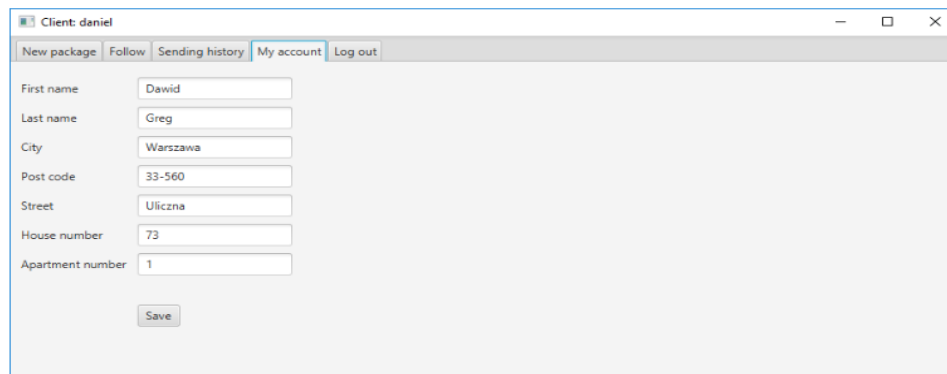
Rysunek 36: Okno edycji danych

Jeżeli przesyłka jest w trakcie transportu do odbiorcy wyświetli się komunikat informujący o tym. W takim przypadku jest już za późno na edycję danych.



Rysunek 37: Okno z komunikatem uniemożliwiającym edycję danych

W zakładce My account klient ma możliwość podania swoich danych, w szczególności adresu, z którego najczęściej nadaje przesyłki. Dzięki temu nie będzie musiał podawać go przy każdym nowym nadaniu.



Client: daniel

Navigation: New package | Follow | Sending history | **My account** | Log out

Form fields:

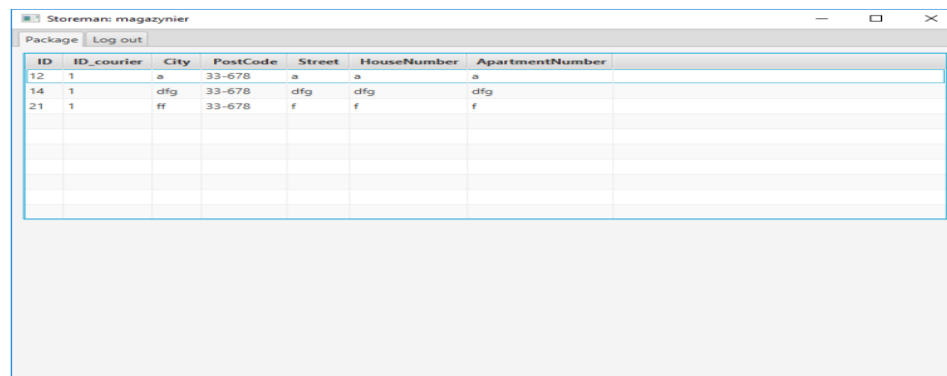
- First name: Dawid
- Last name: Greg
- City: Warszawa
- Post code: 33-560
- Street: Uliczna
- House number: 73
- Apartment number: 1

Save button

Rysunek 38: Zakładka My account

5.3.3 Instrukcja użytkowania aplikacji przez magazyniera

Po zalogowaniu na konto magazyniera wyświetla się okienko z tabelą wyświetlającą wszystkie przesyłki znajdujące się w magazynie.



ID	ID_courier	City	PostCode	Street	HouseNumber	ApartmentNumber
12	1	a	33-678	a	a	a
14	1	dfg	33-678	dfg	dfg	dfg
21	1	ff	33-678	f	f	f

Rysunek 39: Tabela z przesyłkami znajdującymi się w magazynie

Po kliknięciu na rekord konkretnej przesyłki magazynier dostaje szczegółowe informacje o niej. W celu zlecenia jej dostarczenia konkretnemu kurierowi, magazynier może edytować pole Target.

The screenshot shows a web application window titled "Storeman: magazynier". It has a "Package" tab and a "Log out" button. Below the navigation bar is a table with the following columns: ID, ID_courier, City, PostCode, Street, HouseNumber, and ApartmentNumber. The table contains three rows of data, with the second row (ID 14) highlighted in blue. Below the table is a form for editing the selected package. The form has labels and input fields for: ID (14), ID courier (1), City (dfg), Post Code (33-678), Street (dfg), House number (dfg), Apartment number (dfg), and Target (4). A "Moved" button is located at the bottom right of the form.

ID	ID_courier	City	PostCode	Street	HouseNumber	ApartmentNumber
12	1	a	33-678	a	a	a
14	1	dfg	33-678	dfg	dfg	dfg
21	1	ff	33-678	f	f	f

ID: 14
 ID courier: 1
 City: dfg
 Post Code: 33-678
 Street: dfg
 House number: dfg
 Apartment number: dfg
 Target: 4
 Moved

Rysunek 40: Przypisanie przesyłki kurierowi

5.3.4 Instrukcja użytkownika dla kuriera

Zakładka For Delivery gromadzi dane o wszystkich przesyłkach, za które aktualnie odpowiedzialny jest kurier.

The screenshot shows a web application window titled "Courier kurier1". It has a "For Delivery" tab and a "Log out" button. Below the navigation bar is a table with the following columns: ID, TelephoneNumber, DeliveryStatus, ReceiverFirstName, ReceiverLastName, ReceiverCity, ReceiverPostCode, ReceiverStreet, ReceiverHouseNumber, and ReceiverApartmentNumber. The table contains ten rows of data. Below the table is a large empty space.

ID	TelephoneNumber	DeliveryStatus	ReceiverFirstName	ReceiverLastName	ReceiverCity	ReceiverPostCode	ReceiverStreet	ReceiverHouseNumber	ReceiverApartmentNumber
24	23543	toPickUp	asd	asd	asd	33-678	asd	asd	asd
25	234234234	toPickUp	ja	kow	hls	56-785	ulw	4	23
26	123456768	toPickUp	Maciel	złtani	Fuzuu	34-534	polil	4	54
27	888234579	toPickUp	Janusz	Nowak	Poznan	12-342	Brzydka	14	6
28	123456789	toPickUp	h	h	h	77-777	h	h	h
30	234	toPickUp	g	g	g	43-432	g	g	g
34	123456789	toPickUp	Andrzej	Kowalski	Wroclaw	50-343	Drukarska	32	1
35	123	toPickUp	Jan	Nowak	Wroclaw	12-345	Grunwaldzka	23	1
36	756351212	toPickUp	David	Jakubiak	Zielona Gora	34-756	Warszawska	23	1
37	234345678	toPickUp	Jakub	Jakubowski	Gdansk	67-345	Mazowiecka	54	2

Rysunek 41: Zakładka For Delivery

Po naciśnięciu na rekord przesyłki, kurier może wybrać z listy status przesyłki. Jeśli z jakiegoś powodu kurier nie może dostarczyć przesyłki, powinien on ustawić status przesyłki na Undelivered. Po dostarczeniu paczki, status zmienia się na Delivered.

The screenshot shows the 'Courier kurier1' application window with the 'Undelivered' tab selected. The window has a menu bar with 'For Delivery', 'Undelivered', and 'Log out'. Below the menu is a table with columns: ID, TelephoneNumber, DeliveryStatus, ReceiverFirstName, ReceiverLastName, ReceiverCity, ReceiverPostCode, ReceiverStreet, ReceiverHouseNumber, and ReceiverApartmentNumber. The table contains several records, with the one for ID 30 (Receiver: g, 43-432) highlighted. Below the table is a form with fields for First Name, Last Name, City, Post Code, Street, House number, Apartment number, and Telephone, along with a 'Delivered status' dropdown menu and a 'Confirm' button.

ID	TelephoneNumber	DeliveryStatus	ReceiverFirstName	ReceiverLastName	ReceiverCity	ReceiverPostCode	ReceiverStreet	ReceiverHouseNumber	ReceiverApartmentNumber
24	23543	toPickUp	and	and	and	33-678	and	and	and
25	234234234	toPickUp	ja	kow	hhi	56-785	uliv	4	23
26	123456768	toPickUp	Maciel	zklani	Fuuuu	34-534	polil	4	54
27	888234579	toPickUp	Janusz	Nowak	Poznan	12-342	Brzydka	14	6
28	123456789	toPickUp	h	h	h	77-777	h	h	h
30	234	toPickUp	g	g	g	43-432	g	g	g
34	123456789	toPickUp	Andrzej	Kowalski	Wroclaw	50-343	Drukarska	32	1
35	123	toPickUp	Jan	Nowak	Wroclaw	12-345	Grunwaldzka	23	1
36	756351212	toPickUp	Dawid	Jakubiak	Zielona Gora	34-756	Warszawska	23	1
37	234345678	toPickUp	Jakub	Jakubowski	Gdansk	67-345	Mazowiecka	54	2

Rysunek 42: Zmiana statusów przesyłki

Po zatwierdzeniu statusu przesyłki z wybraną opcją Undelivered przesyłka zostanie przeniesiona do zakładki Undelivered, a kurier ma obowiązek zwrócić ją do magazynu.

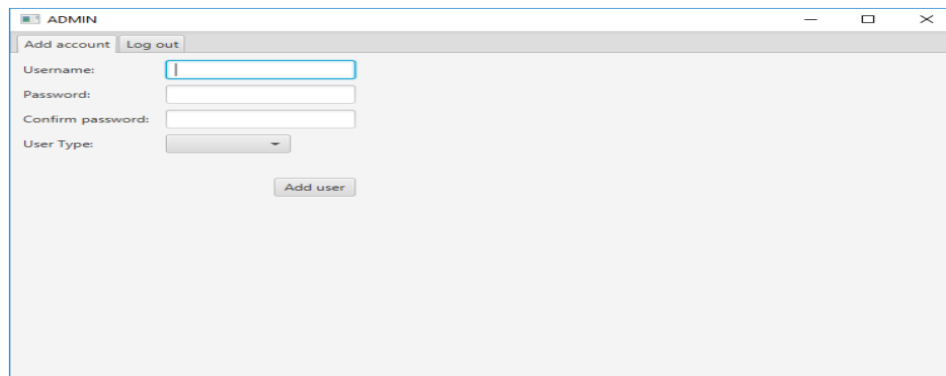
The screenshot shows the 'Courier kurier1' application window with the 'Undelivered' tab selected. The window has a menu bar with 'For Delivery', 'Undelivered', and 'Log out'. Below the menu is a table with columns: ID, TelephoneNumber, DeliveryStatus, ReceiverFirstName, ReceiverLastName, ReceiverCity, ReceiverPostCode, ReceiverStreet, ReceiverHouseNumber, and ReceiverApartmentNumber. The table contains three records, all with 'undelivered' status. The rest of the table is empty.

ID	TelephoneNumber	DeliveryStatus	ReceiverFirstName	ReceiverLastName	ReceiverCity	ReceiverPostCode	ReceiverStreet	ReceiverHouseNumber	ReceiverApartmentNumber
1	787878787	undelivered	Adam	Greg	Warszawa	33-678	Uliczna	73	1
7	123456789	undelivered	Anna	Brat	Lodz	33-678	Wroclawska	34	6
10	123456789	undelivered	Anna	Brat	Lodz	33-678	Wroclawska	34	6

Rysunek 43: Zakładka Undelivered

5.3.5 Instrukcja obsługi aplikacji przez administratora

Administrator ma uprawnienia do tworzenia nowych kont użytkowników dla kurierów oraz magazynierów. Po zatwierdzeniu przyciskiem Add user, wyświetlony zostanie komunikat o powodzeniu tworzenia użytkownika.



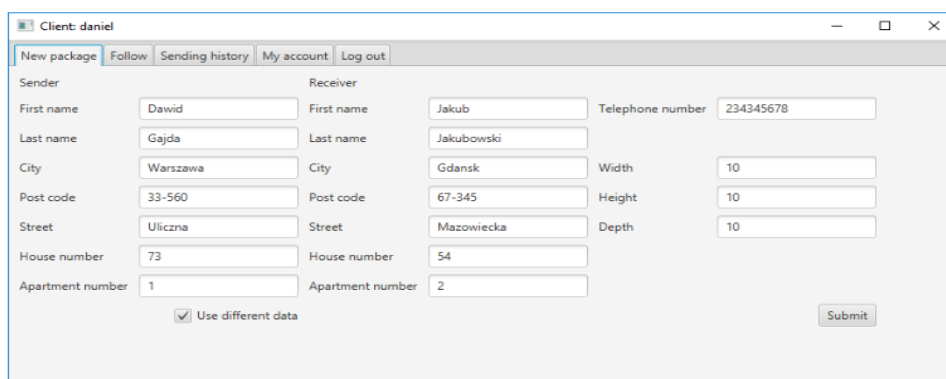
The screenshot shows a web application window titled 'ADMIN'. It has two tabs: 'Add account' (selected) and 'Log out'. The form contains the following fields: 'Username:' with a text input, 'Password:' with a text input, 'Confirm password:' with a text input, and 'User Type:' with a dropdown menu. Below these fields is a button labeled 'Add user'.

Rysunek 44: Zakładka Add user

5.4 Testowanie opracowanych funkcji systemu

5.4.1 Testy manualne aplikacji

Nadawanie nowej przesyłki.



The screenshot shows a web application window titled 'Client: daniel'. It has four tabs: 'New package' (selected), 'Follow', 'Sending history', and 'Log out'. The form is divided into two main sections: 'Sender' and 'Receiver'. The 'Sender' section includes fields for First name (Dawid), Last name (Gajda), City (Warszawa), Post code (33-560), Street (Uliczna), House number (73), and Apartment number (1). The 'Receiver' section includes fields for First name (Jakub), Last name (Jakubowski), Telephone number (234345678), City (Gdansk), Post code (67-345), Street (Mazowiecka), House number (54), and Apartment number (2). There are also fields for Width (10), Height (10), and Depth (10). A checkbox labeled 'Use different data' is checked. A 'Submit' button is located at the bottom right.

Rysunek 45: Nadawanie przesyłki

Po zatwierdzeniu przyciskiem submit, wyświetli się napis z wygenerowanym numerem przesyłki.

Client: daniel

New package | Follow | Sending history | My account | Log out

Sender

First name: Dawid

Last name: Greg

City: Warszawa

Post code: 33-560

Street: Uliczna

House number: 73

Apartment number: 1

Receiver

First name:

Last name:

City:

Post code:

House number:

Apartment number:

Telephone number:

Width:

Height:

Depth:

Package number: 37

☐ Use different data

Submit

Rysunek 46: Nadawanie przesyłki zakończone sukcesem

W zakładce Follow po wpisaniu numeru przesyłki wyświetli się jego aktualna lokalizacja.

Client: daniel

New package | Follow | Sending history | My account | Log out

Package number: 37 Find

Package location: Do odebrania od nadawcy

Rysunek 47: Śledzenie przesyłki

Jeżeli przesyłka nie została odebrana od nadawcy istnieje możliwość anulowania przesyłki lub zmiany danych.

ID_client	ID	TelephoneNumber	Location	DeliveredStatus	senderFirstName	senderLastName	senderCity	Sender
4	20	435	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-67
4	21	56	Do odebrania od nadawcy	inWarehouse	Adam	Greg	Warszawa	33-67
4	22	345	Do odebrania od nadawcy	toDelivery	Adam	Greg	Warszawa	33-67
4	23	44	Do odebrania od nadawcy	toDelivery	Marek	Greg	Warszawa	33-67
4	24	23543	Do odebrania od nadawcy	toPickUp	Janusz	Greg	Warszawa	33-67
4	28	123456789	Do odebrania od nadawcy	toPickUp	Marek	Greg	Warszawa	33-67
4	29	123456789	Odebrano od nadawcy	pickedUp	DROP ALL TABLES;	DROP ALL TABLES;	DROP ALL TABLES;	33-67
4	30	234	Do odebrania od nadawcy	toPickUp	Marek	Greg	Warszawa	33-67
4	31	123456781	Do odebrania od nadawcy		Daniel	Greg	Warszawa	33-56
4	32	789123615	Do odebrania od nadawcy		Daniel	Greg	Warszawa	33-56
4	36	756351212	Do odebrania od nadawcy	toPickUp	Daniel	Greg	Warszawa	33-56
4	37	234345678	Do odebrania od nadawcy	toPickUp		Gajda	Warszawa	33-56

Rysunek 48: Edytowanie danych

Sender		Receiver		Telephone number
First name	Dawid	First name	Dawid	756351212
Last name	Greg	Last name	Jakubiak	
City	Warszawa	City	Zielona Gora	
Post code	33-560	Post code	34-756	
Street	Uliczna	Street	Warszawska	
House number	73	House number	23	
Apartment number	1	Apartment number	1	

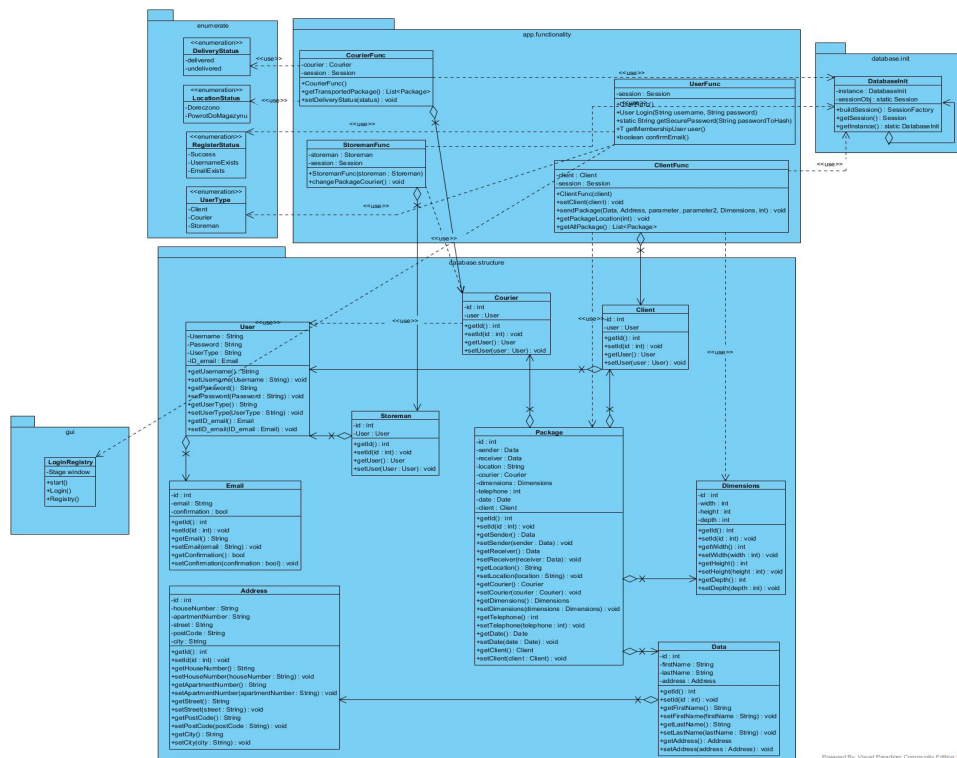
Submit

Rysunek 49: Wprowadzanie nowych danych

5.4.2 Testy jednostkowe

Funkcjonalności naszej aplikacji, zdefiniowane w klasach UserFunc, CourierFunc, StoremanFunc i ClientFunc (diagram klas zaprezentowany został poniżej) zostały poddane testom jednostkowym z wykorzystaniem narzędzi JUnit i Mockito. W celu przetestowania ich zgodnie z definicją testu jednostkowego, która nie zakłada łączenia się z bazą danych zawierającą dane logowania użytkowników, stworzone zostały klasy DataBase i simQuery, które imitują jej działanie. Dla klasy ClientFunc wykonane zostały testy nadawania przesyłki dla nowych danych klienta i adresu, nadania przesyłki dla danych

istniejących w bazie oraz nadania przesyłki dla nowych danych użytkownika i istniejącego w bazie adresu. W przypadku klasy UserFunc, funkcjonalności prowadzą się do logowania i rejestracji użytkownika, które przetestowane zostały jako logowanie na konta różnego typu, logowanie, używając nieistniejącej nazwy użytkownika, rejestrację na istniejące w bazie adres e-mail lub nazwę użytkownika. Przetestowane funkcjonalności klienta to ustawianie statusu przesyłek na Delivered, Undelivered, PickedUp, NotPickedUp i UnknownDeliveredStatus.

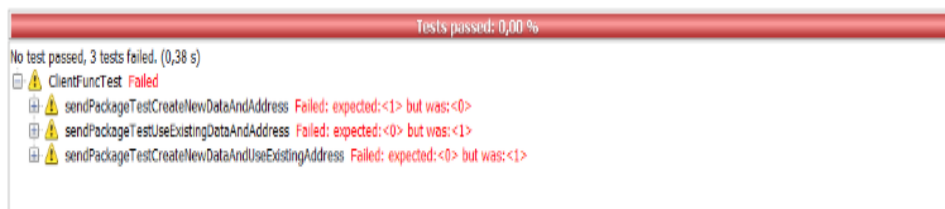


Rysunek 50: Diagram klas aplikacji

Przykładowo, dla klasy ClientFunc, kiedy dane testowe były zgodne z tym, czego oczekiwał program, testy wychodziły pozytywnie, w przeciwnym razie negatywnie.



Rysunek 51: Pozytywne testy jednostkowe



Rysunek 52: Negatywne testy jednostkowe

5.5 Omówienie wybranych rozwiązań programistycznych

Łączenie z bazą zostało zrealizowane za pomocą systemu ORM Hibernate. Do stworzenia aplikacji zastosowaliśmy podejście Database First, a następnie na podstawie istniejącej już bazy zostały stworzone klasy reprezentujące strukturę danych. Pobieranie danych zostało zrealizowane za pomocą obiektu Query który jako parametr przyjmuje składnię języka zapytań SQL. Łączenie aplikacji z bazą zrealizowane zostało na pięciu poziomach dostępu. Podczas włączenia aplikacji dostęp do bazy odbywa się za pomocą użytkownika DeliveryDbUserLogin. Następnie po prawidłowym zalogowaniu na odpowiedni typ użytkownika sesja jest zamykana i otwierane jest nowe połączenie za pomocą odpowiedniego użytkownika. Po zalogowaniu użytkownika sesja jest zamykana dopiero po zamknięciu aplikacji lub wylogowaniu. Takie podejście łączenia z bazą zwiększa bezpieczeństwo w przypadku dostępu do bazy osób trzecich, ponieważ każdy poziom dostępu do bazy ma minimalne prawa do operacji na danych.

5.5.1 Implementacja interfejsu dostępu do bazy danych

Kod przedstawiony w Listingu 1 odpowiada za utworzenie połączenia z bazą danych przy użyciu hibernate. Jako argument funkcji w linii 6 przekazujemy nazwę pliku konfiguracyjnego Hibernate, w którym znajdują się dane dotyczące połączenia z bazą oraz zasady mapowania tabel bazy danych na klasy aplikacji. Po poprawnym nawiązaniu połączenia z bazą metoda zwraca obiekt klasy Session.

Metoda w linii 22 zwraca odpowiedni obiekt Session zależnie od danych logowania do bazy. Takie podejście do sposobu nawiązywania połączenia z bazą zwiększa jego bezpieczeństwo, ponieważ każdy typ użytkownika w aplikacji posiada swoje konto w systemie zarządzania bazą danych posiadając tylko niezbędne uprawnienia. Do metody przekazujemy odpowiedni typ wyliczeniowy w którym znajdują się dostępne poziomy logowania do bazy. Dzięki przekazaniu tej zmiennej, funkcja rozpoznaje typ użytkownika oraz przekazuje odpowiedni plik konfiguracyjny do metody buildSession. Po poprawnym utworzeniu sesji jest ona zwracana.

```
1  /**
2   * konfiguruje i nawiązuje połączenia z baza danych o wskazanej konfiguracji
3   * @param String conf plik z konfiguracja
4   * @return zwraca utworzona sesje
5   */
6  private SessionFactory buildSession(String conf)
7  {
8      Configuration configObj = new Configuration();
9      configObj.configure(conf);
10
11      ServiceRegistry serviceRegistryObj =
12  newStandardServiceRegistryBuilder().applySettings(configObj.getProperties()).build();
13
14      SessionFactoryObj = configObj.buildSessionFactory(serviceRegistryObj);
15      return SessionFactoryObj;
16  }
17
18  /**
19   * odpowiada za utworzenie odpowiedniej sesji
20   * (zależnie od typu użytkownika) oraz otwarcie jej
21   * @param type okresla użytkownika który ma polaczyc sie z baza
22   * @return zwraca sesje
23   */
24  public Session getSession(SessionType type)
25  {
26      switch(type)
```

```

27     {
28         case Admin:
29         {
30             closeSessionsWithout(SessionType.Admin); //zamyka pozostale sesje
31                                                         //z pominięciem sesji dla admina
32             if(sessionObjAdmin == null || !sessionObjAdmin.isOpen())
33                 sessionObjAdmin = buildSession("hibernate_Admin.cfg.xml").openSession();
34             return sessionObjAdmin;
35         }
36         case Client:
37         {
38             closeSessionsWithout(SessionType.Client);
39             if(sessionObjClient == null || !sessionObjClient.isOpen())
40                 sessionObjClient = buildSession("hibernate_Client.cfg.xml").openSession();
41             return sessionObjClient;
42         }
43         case Courier:
44         {
45             closeSessionsWithout(SessionType.Courier);
46             if(sessionObjCourier == null || !sessionObjCourier.isOpen())
47                 sessionObjCourier = buildSession("hibernate_Courier.cfg.xml").openSession();
48             return sessionObjCourier;
49         }
50         case Login:
51         {
52             closeSessionsWithout(SessionType.Login);
53             if(sessionObjLogin == null || !sessionObjLogin.isOpen())
54                 sessionObjLogin = buildSession("hibernate_Login.cfg.xml").openSession();
55             return sessionObjLogin;
56         }
57         case Storeman:
58         {
59             closeSessionsWithout(SessionType.Storeman);
60             if(sessionObjStoreman == null || !sessionObjStoreman.isOpen())
61                 sessionObjStoreman = buildSession("hibernate_Storeman.cfg.xml").openSession();
62             return sessionObjStoreman;
63         }
64         default:
65         {
66             return null;
67         }
68     }
69 }

```

5.5.2 Implementacja wybranych funkcjonalności systemu

Metoda przedstawiona w Listingu 2 odpowiada za logowanie użytkownika do aplikacji. Linie 6-26 odpowiadają za sprawdzenie poprawności wprowadzonych danych. W procesie walidacji sprawdza się czy pola tekstowe nie są puste. W lini 28 tworzony jest obiekt klasy UserFunc, odpowiadającej za logowanie oraz rejestrację. Od linii 39 przedstawiony kod, za pomocą switch() rozpoznaje typ użytkownika oraz otwiera odpowiednie okno aplikacji.

```
1  /**
2   funkcja wywoływana przez przycisk login w oknie logowania,
3   sprawdza poprawność wprowadzonych danych i otwiera
4   odpowiednie okno zależnie od typu użytkownika
5   */
6  private void loginSubmit()
7      {
8      if(inputUsername.getText().equals(""))
9          {
10             labelUserError.setText("Empty username");
11             return;
12          }
13      else
14          {
15             labelUserError.setText("");
16          }
17
18      if(inputPassword.getText().equals(""))
19          {
20             labelUserError.setText("Empty password");
21             return;
22          }
23      else
24          {
25             labelUserError.setText("");
26          }
27
28      try
29      {
30          UserFunc userFunc = new UserFunc();
31
32          User user = userFunc.Login(inputUsername.getText(), inputPassword.getText());
33          if(user == null)
34          {
35             labelUserError.setText("Incorrect username or password");
36          }
37      }
```

```

37         else
38         {
39             labelUserError.setText("success");
40
41             switch(UserType.valueOf(user.getUserType()))
42             {
43                 case Client:
44                 {
45                     ClientGUI clientGUI = new ClientGUI((Client)userFunc.getMembership(user));
46                     window.close();
47                     clientGUI.Display();
48                     break;
49                 }
50                 case Courier:
51                 {
52                     CourierGUI courierGUI = new
53                     CourierGUI((Courier)userFunc.getMembership(user));
54                     window.close();
55                     courierGUI.Display();
56                     break;
57                 }
58                 case Storeman:
59                 {
60                     StoremanGUI storemanGUI = new
61                     StoremanGUI((Storeman)userFunc.getMembership(user));
62                     window.close();
63                     storemanGUI.Display();
64                     break;
65                 }
66                 case Admin:
67                 {
68                     AdminGUI adminGUI = new AdminGUI();
69                     window.close();
70                     adminGUI.Display();
71                     break;
72                 }
73             }
74         }
75     }
76     catch (NoSuchAlgorithmException ex)
77     {
78         System.err.println(ex.getStackTrace());
79     }
80 }

```

Poniższy listing przedstawia metodę nadawania przesyłek oraz sposób dobierania bądź tworzenia odpowiednich danych adresatów. W liniach 86-117 kolejno zostaje stworzony obiekt Package. Następnie do obiektu dodajemy dane nadawcy, odbiorcy, klienta, wymiary, status przesyłki (linie 92- 101). Podczas dodawania nadawcy oraz odbiorcy, zostaje wywołana metoda setData (ciało metody znajduje się w liniach 58 - 78). Ta metoda zwraca obiekt Data zależnie od danych w bazie. Za pomocą if-a sprawdza czy dane adresowe oraz personalne istnieją w bazie. Jeżeli nie istnieją tworzy nowy obiekt i dodaje go do bazy. Za sprawdzenie czy podane dane istnieją w bazie odpowiada metoda findData znajdująca się w liniach 27-50. Metoda sprawdza w bazie danych czy podane istnieją oraz je zwraca. Jeżeli okaże się że dane nie istnieją zwraca null. Następnie za pomocą metody findAddress (linie 6-21) sprawdza czy podany adres istnieje. Tak samo jak w metodzie findData zwracamy null jeżeli nie istnieje.

```

1  /**
2  wyszukuje przekazany adres w bazie
3  @param Address address adres ktory bedzie wyszukiwac w bazie
4  @return jezeli podany adres znaleziono to go zwraca, jezeli nie zwraca null
5  */
6  private Address findAddress(Address address)
7  {
8      session.beginTransaction();
9
10     Query q = session.createQuery("FROM Address WHERE houseNumber = :hn
11 AND apartmentNumber = :an AND street = :s AND postCode = :pc AND city = :c ");
12     q.setParameter("hn", address.getHouseNumber());
13     q.setParameter("an", address.getApartmentNumber());
14     q.setParameter("s", address.getStreet());
15     q.setParameter("pc", address.getPostCode());
16     q.setParameter("c", address.getCity());
17
18     Address addr = (Address)q.uniqueResult();
19
20     session.getTransaction().commit();
21     return addr;
22 }
23
24 /**
25 odpowiada za szukanie w tabeli Data w bazie danych
26 @param Data data, Address address przyjmuje 2 argumenty, pierwszy okresla dane do
27 znalezienia natomiast drugi adres @return zwraca znalezione dane, jezeli dane nie
28 istnieja w bazie zwraca null
29 */

```



```

30     private Data findData(Data data, Address address)
31     {
32         session.beginTransaction();
33
34         Query q = session.createQuery("FROM Data WHERE firstName = :fn
35         AND lastName = :ln");
36         q.setParameter("fn", data.getFirstName());
37         q.setParameter("ln", data.getLastName());
38         @SuppressWarnings("unchecked")
39         List<Data> dataOut = q.list();
40         session.getTransaction().commit();
41
42         if(dataOut != null)
43         {
44             for(Data current : dataOut)
45             {
46                 if (current.getAddress().equals(address))
47                 {
48                     return current;
49                 }
50             }
51         }
52         return null;
53     }
54
55     /**
56     tworzy odpowiedni obiekt Data zaleznie od danych w bazie danych,
57     jezeli adres i dane sa takie same jak podane w parametrach funkcji
58     zwraca istniejacy obiekt w bazie danych, jezeli adres istnieje w bazie,
59     tworzy tylko nowy obiekt Data i przypisuje do niego istniejacy adres,
60     jezeli zarowno adres jak i dane nie istnieja w bazie tworzone sa oba obiekty
61     @param Data data, Address address przyjmuje 2 argumenty, pierwszy okresla dane
62     do znalezienia natomiast drugi adres @return zwraca odpowiedni obiekt Data
63     */
64
65     private Data setData(Data data, Address address)
66     {
67         Data dataSender = findData(data, address);
68         if(dataSender == null)
69         {
70             Address addrSender = findAddress(address);
71             if(addrSender == null)
72             {
73                 data.setAddress(address);

```

```

74         }
75         else
76         {
77             data.setAddress(addrSender);
78         }
79         return data;
80     }
81     else
82     {
83         return dataSender;
84     }
85 }
86 /**
87  odpowiada za zapisanie nadanej paczki przez klienta w bazie danych
88  @param Data sender, Address addressSender, Data receiver,
89  Address addressReceiver, Dimensions dimension, int telephone odpowiednio
90  dane i adres adresata, dane i adres odbiorcy, wymiary, nr telefonu
91  @return zwraca wygenerowany przez baze unikalny numer przesyłki,
92  jezeli wystapi blad zwraca -1
93  */
94 public long SendPackage(Data sender, Address addressSender, Data receiver,
95 Address addressReceiver, Dimensions dimension, int telephone)
96 {
97     long id;
98     Date utilDate = new Date();
99     Package pack = new Package();
100
101     pack.setSender(setData(sender, addressSender));
102     pack.setReceiver(setData(receiver, addressReceiver));
103     pack.setClient(client);
104     pack.setDeliveredStatus(DeliveryStatus.toPickUp.toString());
105     pack.setDimensions(dimension);
106     pack.setLocation(LocationStatus.DoOdebraniaOdNadawcy.toString());
107     pack.setTelephone(telephone);
108     pack.setDate(utilDate);
109     pack.setCurier(chooseCourier());
110
111     try
112     {
113         session.beginTransaction();
114         id = Long.parseLong(session.save(pack).toString());
115         session.getTransaction().commit();
116         //session.close();
117         //session = DatabaseInit.getInstance().getSession(SessionType.Client);

```

```

118         return id;
119     }
120     catch(Exception ex)
121     {
122         System.err.println(ex.getMessage());
123         return -1;
124     }
125 }

```

5.5.3 Implementacja mechanizmów bezpieczeństwa

W aplikacji zostały zaimplementowane podstawowe mechanizmy bezpieczeństwa tj. logowanie oraz rejestracja. Listing 4 przedstawia ciało metody obsługującej logowanie. Cała funkcjonalność logowania polega na zapytaniu bazy (linie 10-16) czy dany użytkownik istnieje, jeżeli tak system ORM zwróci odpowiedni obiekt typu User, w przeciwnym przypadku zwróci null. Funkcjonalność rejestracji przedstawiona w liniach 26-69 jest bardziej złożona. Aby dodać nowego użytkownika do systemu muszą być spełnione dwa warunki, użytkownik oraz adres email nie mogą już istnieć w bazie. Metoda odpytuje bazę danych, następnie w linii 46 sprawdza czy adres email istnieje w bazie oraz w linii 49 czy nazwa użytkownika nie będzie duplikatem. Jeżeli któreś z założeń nie będzie poprawne metoda zwróci odpowiedni status rejestracji (linie 47 oraz 50). W przeciwnym wypadku wprowadzone dane zostaną zapisane w bazie, a użytkownik poprawnie zarejestrowany.

```

1  /**
2   odpowiada za logowanie, sprawdzenie czy podane parametry istnieja
3   w bazie @param String username, String password @return jezeli uzytkownik
4   istnieje w bazie zwraca obiekt uzytkownika, w przeciwnym razie zwraca null
5   */
6  public User Login(String username, String password) throws NoSuchAlgorithmException
7  {
8      session.beginTransaction();
9
10     Query q = session.createQuery("FROM User WHERE Username = :un
11     AND Password = :pa");
12     q.setParameter("un", username);
13     q.setParameter("pa", getSecurePassword(password)); //haszowanie hasla sha-256
14
15     User user = (User)q.uniqueResult();
16
17     session.getTransaction().commit();
18
19     return user;

```

```

20     }
21
22     /**
23     odpowiada za rejestracje nowego uzytkownika w bazie, sprawdza
24     czy nazwa uzytkownika jest w bazie, nastepnie czy podany adres
25     email jest w bazie, jezeli nie tworzy obiekt User i dodaje do do bazy
26     @param String username, String password, String email, UserType type
27     @return zwraca RegisterStatus.UsernameExist jezeli nazwa uzytkownika
28     istnieje w bazie, RegisterStatus.EmailExists jezeli adres email istnieje
29     w bazie, RegisterStatus.Success jezeli pomyslnie dodano uzytkownika do bazy
30     */
31     public RegisterStatus Registry(String username, String password, String email,
32     UserType type)// throws Exception
33     {
34         RegisterStatus status = RegisterStatus.Success;
35
36         session.beginTransaction();
37         Query q;
38         Email emailExist = null;
39         if(type == UserType.Client)
40         {
41             q = session.createQuery("FROM Email WHERE email = :e");
42             q.setParameter("e", email);
43             emailExist = (Email)q.uniqueResult();
44         }
45         q = session.createQuery("FROM User WHERE Username = :u");
46         q.setParameter("u", username);
47         User usernameExist = (User)q.uniqueResult();
48
49         session.getTransaction().commit();
50
51         if(emailExist != null && type == UserType.Client)
52             return RegisterStatus.EmailExists;
53         if(usernameExist != null)
54             return RegisterStatus.UsernameExists;
55
56         Email emailObj = new Email();
57         emailObj.setEmail(email);
58
59         User userObj = new User();
60
61         if(type == UserType.Client)
62             userObj.setID_email(emailObj);
63

```

```
64         userObj.setUsername(username);
65         userObj.setPassword(getSecurePassword(password));
66         userObj.setUserType(type.toString());
67
68         session.beginTransaction();
69         session.save(userObj);
70         session.getTransaction().commit();
71
72         return RegisterStatus.Success;
73     }
```

6 Podsumowanie i wnioski

Wybrany temat projektu przysporzył nam kilku problemów, a część początkowych założeń musiała zostać zmodyfikowana. Jednak rozwiązywanie tych problemów spowodowało, że wiele się nauczyliśmy. Ostatecznie jesteśmy zadowoleni z wyglądu i funkcjonalności aplikacji.

Literatura

- [1] MySQL: *MySQL Documentation* Dokumentacja, stan na dzień 26.01.2019r.
- [2] Hibernate: *Hibernate ORM Documentation* Dokumentacja, stan na dzień 26.01.2019r.
- [3] Oracle JavaFX 8: *Packages* Pakiety, stan na dzień 26.01.2019r.
- [4] phpMyAdmin: *Welcome to phpMyAdmin's documentation!* Dokumentacja, stan na dzień 26.01.2019r.
- [5] JUnit: *JUnit 5 User Guide* Instrukcja użytkowania, stan na dzień 26.01.2019r.
- [6] javadoc: *Mockito 2.23.4 API* Dokumentacja, stan na dzień 26.01.2019r.

Spis rysunków

1	Model konceptualny bazy danych	6
2	Model logiczny bazy danych, będący również modelem fizycznym	7
3	Diagram przypadków użycia dla trzech aktorów modelu	10
4	Okienko rejestracji	11
5	Okienko logowania	11
6	Tabela uprawnień dostępu do bazy danych	14
7	Próba aktualizacji adresu E-mail przez nieupoważnioną osobę	15
8	Brak dostępu użytkownika DeliveryDbUserLogin do aktualizacji danych w tabeli Email	16
9	Użytkownik DeliveryDbUserLogin ma dostęp do wyświetlania danych z tabeli Email	16
10	Wprowadzenie istniejących danych do kolumny z ograniczeniem unique	17
11	Wychwycenie próby dodania istniejących danych	17
12	Tabela Storeman przed dodaniem użytkownika	18
13	Dodawanie użytkownika "Janusz" do tabeli User	19
14	Tabela Storeman po dodaniu użytkownika i poprawnym wykonaniu triggera	20
15	Próba wyświetlania danych z widoku CourierData bez uprawnień	21
16	Efekt wyświetlania danych z widoku ClientHistory z poziomu użytkownika DeliveryDbUserClient	22
17	Próba usunięcia rekordu z tabeli User bez posiadania uprawnień	22
18	Pakowanie projektu jako obrazu	23
19	Komunikat potwierdzający poprawne wyeksportowanie	24
20	Zawartość folderu projektu	24
21	Instalator aplikacji w katalogu Output	26
22	Akceptacja licencji	27
23	Wybór lokalizacji	27
24	Wybór folderu, w którym ma się zainstalować	28
25	Tworzenie skrótu na pulpicie	28
26	Rozpoczęcie instalacji	29
27	Instalacja w toku	29
28	Zakończenie instalacji	30
29	Okienko startowe programu	30
30	Okienko tworzenia nowego konta	31
31	Logowanie	31
32	Okno nadawania przesyłki	32
33	Zakładka śledzenia przesyłki	32
34	Zakładka historii przesyłek	33

35	Rezultat kliknięcia prawym przyciskiem myszy na wybrany rząd tabeli	33
36	Okno edycji danych	34
37	Okno z komunikatem uniemożliwiającym edycję danych . . .	34
38	Zakładka My account	35
39	Tabela z przesyłkami znajdującymi się w magazynie	35
40	Przypisanie przesyłki kurierowi	36
41	Zakładka For Delivery	36
42	Zmiana statusów przesyłki	37
43	Zakładka Undelivered	37
44	Zakładka Add user	38
45	Nadawanie przesyłki	38
46	Nadawanie przesyłki zakończone sukcesem	39
47	Śledzenie przesyłki	39
48	Edytowanie danych	40
49	Wprowadzanie nowych danych	40
50	Diagram klas aplikacji	41
51	Pozytywne testy jednostkowe	42
52	Negatywne testy jednostkowe	42