



Технически университет - Варна

Факултет: ФИТА
Катедра: КНТ §§ СИТ

КУРСОВ ПРОЕКТ

Цел на проекта:
ДА СЕ ИЗГРАДИ КУРИЕРСКА ИНФОРМАЦИОННА
СИСТЕМА ЗА СЪХРАНЕНИЕ И ОБРАБОТКА НА
ИНФОРМАЦИЯ.

Изготвили:

Десислава Георгиева Паунова
Ф.номер: 18621405
Спец.: КСТ

Илиян Станчев Станчев
Ф.номер: 18621750
Спец.: СИТ

Проверил:

доц. д-р. инж. Хр. Ненов

/ /

1. ИЗИСКВАНИЯ КЪМ ПРОДУКТА

Да се разработи куриерска информационна система, предоставяща услуга за изпращане и получаване на пратки по куриер. Програмата съхранява и обработва данни за създадени пратки, обработени от куриер и доставени до адрес.

Системата поддържа три вида потребители – администратор, куриер и клиент (с различни роли за достъп до функционалностите в системата).

Операции за работа с потребители:

- Създаване на куриерска(и) фирма(и) от администратор(име, офис в градове и др.);
- Създаване на куриери към фирмата от администратор;
- Регистриране на клиент към куриерска фирма.

Системата поддържа операции за работа с изпращане и получаване на пратки:

- Регистриране на пратка в офис на фирмата. Различните категории пратки (плик,колет,голям пакет, товар) имат различни цени;
- Транзит на пратката до местоназначението (извършва се автоматично от системата на базата на времезаделяща техника);
- Статус на получаването – получена, неполучена, отказана;

Системата поддържа справки по произволен период за:

- Пратки и тяхното състояние;
- Възможност за проследяване на статуса на дадена пратка;
- Статистика на куриерската фирма;
- Работа на куриерите;
- Статистика на клиентите на куриерската фирма.

Системата поддържа известия за:

- Неполучени пратки;
- Отказани пратки;

2. АНАЛИЗ НА ПРОДУКТА

2.1 Функционални изисквания

- Анализ на единиците в системата: Системата поддържа три типа потребители, които получават съответните права в системата. Изгражда се работна среда за всеки тип потребител (GUI). В основата на функционалността стои пратката, от нея произлизат още компания по която да се изпраща, офис/адрес и известие за нейното получаване или отказване.
- Анализ на ролята на клиента: Главна роля на клиента е заявяването на пратки чрез попълване на необходимата информация за създаване на една пратка (получател, тип, цена, цена за сметка на, компания, офис или адрес на доставка...). Още всеки клиент използва системата, за да проследява своите заявени и предстоящи пратки, да бъде известяван при предстояща доставка или върната, заявена от него, пратка.
- Анализ на ролята на куриера: Куриерът обработва заявените пратки като ги поема. Чрез достъпа в системата той още може да проверява своите доставени пратки, да прави справки за свършената от него работа, развитието на компанията в която работи и клиентска статистика.
- Анализ на ролята на администратора: Администраторът е сърцето на системата. Той добавя нови ползватели на софтуера в лицето на куриерските компании, нови куриери и офиси към съответната компания, нови клиенти. Осигурява техния достъп и съответно се грижи за тяхното премахване от системата. Още следи състоянието на пратки, процеса на работа на куриерите и развитието на компаниите.

- Анализ на трансфера на пратката: Пратката се характеризира функционално със своя статус, който се изменя при преминаване на всеки етап на доставка. Пратката се доставя паралелно с работата на потребителя като състоянията се сменят през определено време (определено спрямо разстоянието между двата адреса) или при съответни действия от страна на потребител (например при поемане от куриер, пратката ще променя статуса си от „чакаща“ в „в процес на доставка“).
- Анализ на известяване на клиент: При достигнат желан статус на пратката, клиентът бива уведомен за това чрез изкачащо известие, принадлежащо само и единствено на неговата работна област. Известието подлежи на обработка от страна на клиента, ако пратката е за доставяне, за да се предприемат следващи действие по нейната обработка.

2.2 Структура на проекта

Проектът изисква структура позволяваща комуникация между базата данни и потребителя.



2.3 Дефиниция на модулите на системата

- GUI слой(Controllers): реализира функционалностите на елементите на fxml формите;
Осигурява валидациите като така защитава по-долните нива от грешки;
- Business Logic слой (Service): реализира връзката между GUI и DAO слоя и едновременно с това следва някаква логика свързана със същността на системата;
- Слой за връзка с базата данни:
 - DAO (Data Access Object): осъществява комуникация между програмния код и базата данни. Бизнес логиката използва DAO класовете само за искане или връщане на данни от или към основното хранилище на данни;
 - Entity: моделира базата данни, свързва програмен клас с таблица от базата данни посредством анотации.

3. ПРОЕКТИРАНЕ НА СИСТЕМАТА

Използва се подход Модулно програмиране, който се характеризира с това, че писането на софтуера се разделя на набор от компоненти, наречени модули, всеки от които е управляем по размер, има точно определена цел и има добре дефиниран интерфейс за използване от други модули.

Някои свойства на модула са функционална цялост (реализира цялостно задачата, предназначена за него), логически независим е (резултатът от неговата работа зависи само от входните данни и не зависи от работата на другите модули) и слаба външна и висока вътрешна свързаност (обменът между модулите трябва да е така реализиран, че той да е минимизиран до максимална степен).

Впредвид размера на приложението, за модулна единица се избира пакетът.

3.1 Планиране на изграждането на продукта

- *На ниво GUI:*

Да бъде изградена среда за работа за всеки тип потребител.

- За администратор;
- За клиент;
- За куриер.

Всяка подфункция на средата да бъде реализиране в отделна форма и да бъде обмислена възможността за използването и от различни типове потребители (ако е необходимо).

Пример: администратор и клиент правят Заявяване на пратка, добре е това да става от една форма, която да разпознава с какви права се използва и да функционира спрямо тях .

Разбира се, да бъде изградена и log-in форма, която да идентифицира потребителя и да му даде достъп до съответния ресурс.

Всяка форма да поддържа приблизително еднакъв стил и да е подходяща за нивото на работа (подходяща за съответния потребител), т.е да поддържа характеристики на добре структуриран и лесен за използване интерфейс.

За всяка форма да се поддържа валидация на данни като например при работа на клиент тези валидации да са синхронни с работата на потребителя.

Функционалности на средата за работа на Администратор:

- Създаване/Изтриване на Пратка;
- Добавяне/Премахване на Куриер към/от определена компания;
- Добавяне/Премахване на Клиент;
- Създаване/Изтриване на Компания;
- Добавяне/Премахване на офис към/от определена компания;
- Справка за Куриер;
- Справка за Пратка;
- Справка за Компания.
- Изход от системата.

Функционалности на средата за работа на Клиент:

- Заявяване на пратка;
- Отказване на пратка (ако вече не пътува към съответния клиент);
- Преглед на заявени пратки;
- Преглед на очаквани пратки;
- Проследяване на пратки;
- Преглед и редактиране на профил;
- Преглед на известия;
- Изход от системата.

Функционалности на средата за работа на Куриера:

- Поемане на пратка (доставяне);
- Преглед на доставени пратки;
- Статистика на Клиент;
- Статистика на Компания;
- Проследяване на личен прогрес;
- Преглед и редактиране на профил;
- Изход от системата.

• На ниво бизнес логика:

Да бъдат изградени интерфейси гарантиращи функционалностите на системата, като правят достъпа до базата данни през осигурените от слоя за връзка с базата данни функции. Тези интерфейси да бъдат разпределени като всеки да отговаря за част от пълната функционалност.

- Работа с Потребители;
- Работа с Пратки;
- Работа с Компании;
- Работа при влизане в системата;
- Работа с Известия.

Поддръжка на работа с потребители:

- създаването на потребител;
- изтриване на потребител;
- търсене по определни изисквания;
- редактиране на данни (парола, ел.поща, адрес, телефонен номер т.н);
- получаване на списък от всички клиенти/куриери.

Поддръжка на работа с пратки:

- Създаване на пратка;
- Изтриване на пратка;
- Промяна статуса на пратка;
- Търсене по определени изисквания;
- Получаване на списък от заявени пратки на даден клиент;
- Получаване на списък от очаквани пратки на даден клиент;
- Получаване на списък от пратки чакащи да бъдат изпратени;
- Получаване на списък от доставени пратки от даден куриер;
- Възлагане на пратка на куриер.

Поддръжка на работа с компании:

- Създаване на компания;
- Изтриване на компания;
- Търсене на компания;
- Създаване на офис към компания;
- Изтриване на офис към компания;
- Получаване на списък от всички офиси за дадена компания;
- Получаване на списък от всички офиси;
- Търсене на офис.

Поддръжка на работа при влизане в системата:

- Проверка на въведената информация за вход.

Поддръжка на работа с известия:

- Изпращане на известия до клиент;
- Получаване на списък от всички известия за текущ клиент;
- Промяна състоянието на известие;
- Изтриване на известие;
- Търсене на известие;
- Обработка на известие.

Бизнес логиката се грижи и за доставяне на пратката от адрес до адрес или от адрес до офис:

Класът имплементира интерфейса Runnable, който стартира алгоритъм в background режим, отговарящ за доставяне на пратката:

- изчисляване на времето за доставка,
- изпращането на известие след приключване на доставяне.

- *На ниво връзка с база данни:*

DAO:

Да бъде изграден клас реализиращ връзката с базата данни при инициализирането му със стартирането на системата.

Той да бъде използван от интерфейси гарантиращи възможността за опериране с данни от базата данни.

Тези интерфейси да бъдат разпределени като всеки да отговаря за част от пълната функционалност.

- Достъп до данните, отнасящи се до потребители;
- Достъп до данните, отнасящи се до компании;
- Достъп до данните, отнасящи се до офиси;
- Достъп до данните, отнасящи се до поръчки;
- Достъп до данните, отнасящи се до известия.

Достъп до данните, отнасящи се до потребители:

- Запазване на потребител;
- Изтриване на потребител;
- Актуализиране на потребител;
- Получаване на потребител по определени изисквания;
- Получаване списък от всички клиенти/куриери.

Достъп до данните, отнасящи се до компании:

- Запазване на компания;
- Изтриване на компания;
- Получаване на компания по определени изисквания;
- Актуализиране на компания;
- Получаване списък от всички компании.

Достъп до данните, отнасящи се до офиси:

- Запазване на офис;
- Изтриване на офис;
- Получаване на офис по определени изисквания;
- Актуализиране на офис;
- Получаване списък от всички компании.

Достъп до данните, отнасящи се до поръчки:

- Запазване на поръчка;
- Изтриване на поръчка;
- Получаване на поръчка по определени изисквания;
- Актуализиране на поръчка;
- Получаване списъци от всички компании в зависимост от тяхното състояние;
- Получаване броя на успешните/неуспешните пратки.

Достъп до данните, отнасящи се до известия:

- Запазване на известие;
- Изтриване на известие;
- Получаване на известие по ID;
- Актуализиране на известие;
- Получаване списъци от всички известия на даден клиент.

ENTITY:

Изграждане на класове представлящи основните единици в системата, които са обекти на съхранение в хранилището:

- Клас за клиент;
- Клас за куриер;
- Клас за администратор;
- Клас за пратка;
- Клас за компания;
- Клас за офис;
- Клас за известие;

Изграждане на други класове:

- Клас за адрес;
- Клас за лице за контакт;
- Клас за статус на поръчка;
- Клас за тип на потъчка.

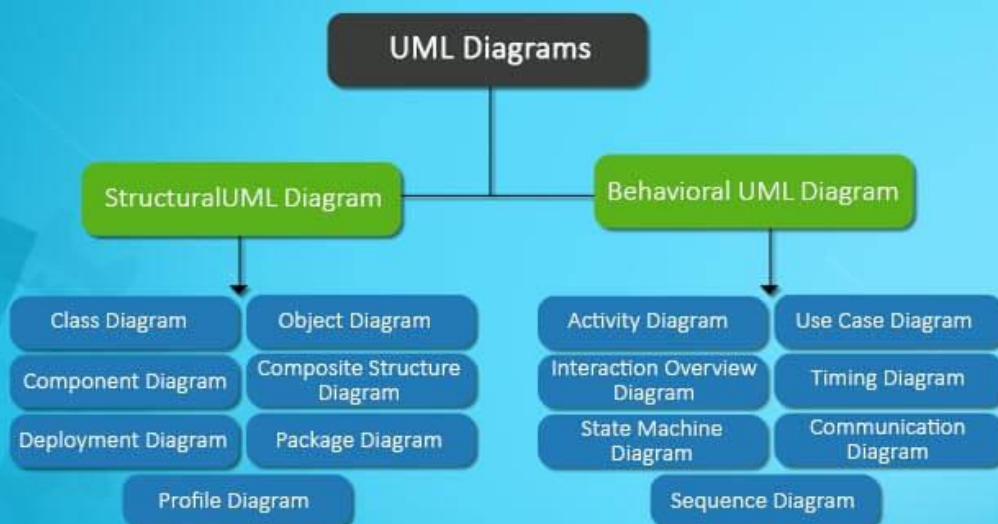
3.2 UML (Unified Modeling Language)

Унифицираният език за моделиране е създаден с цел даване визуална представа за софтуера.

Чрез него се моделира поведение на системата, а това от своя страна намира приложение в етапите на планиране и документация на продукта.



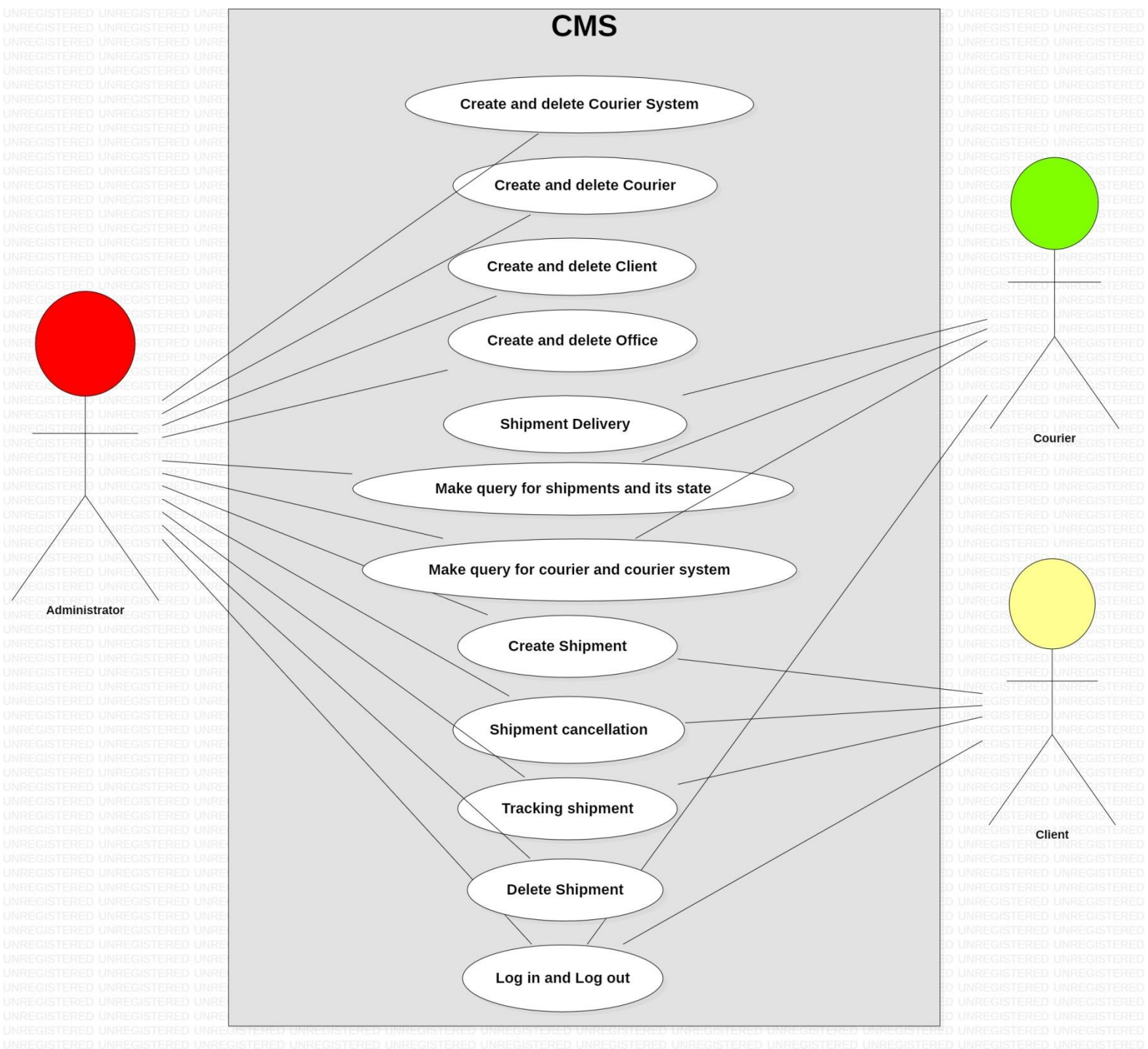
Types of UML Diagrams



- **Use case диаграма:**

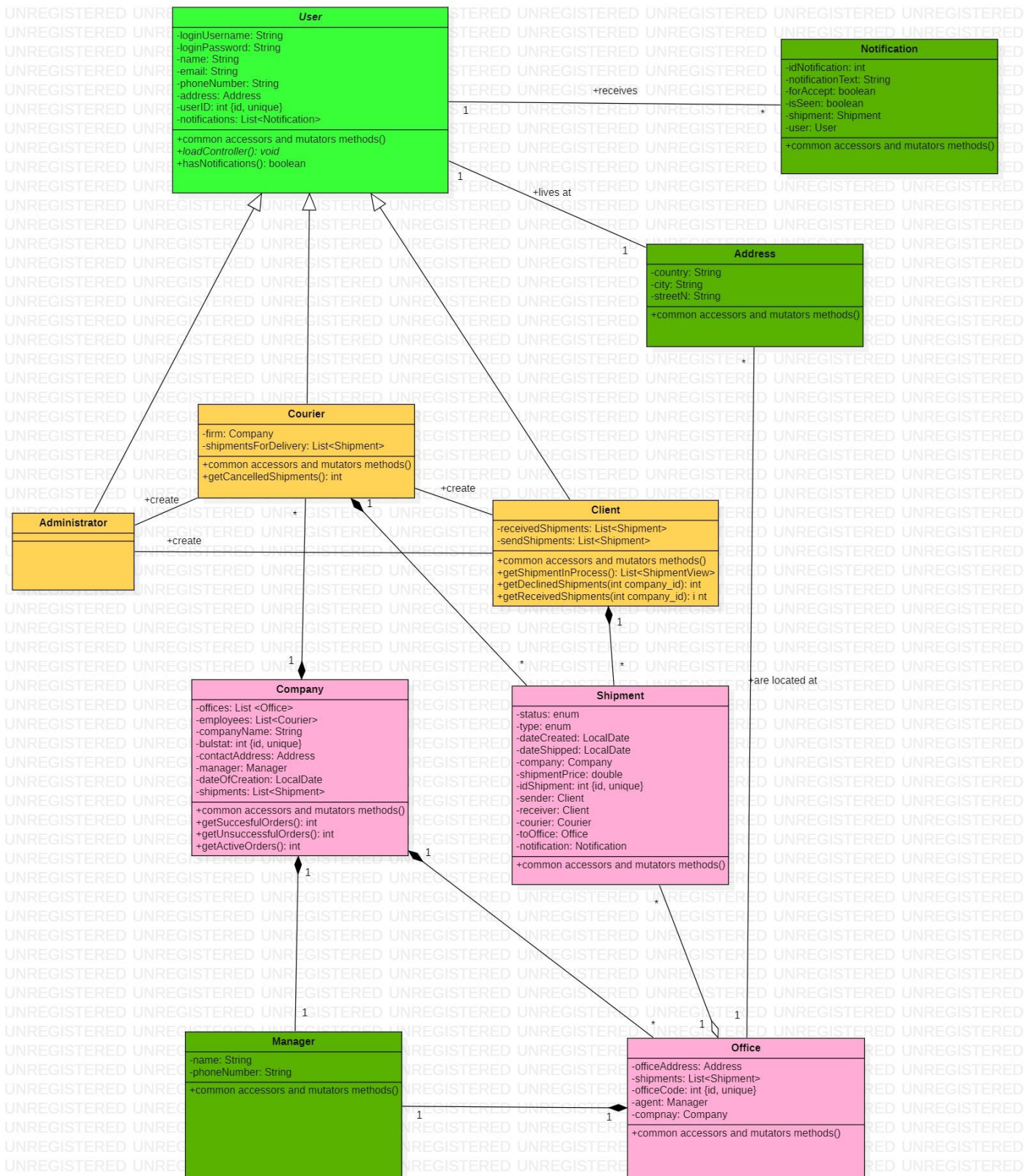
Проектиране на ролите в системата и тяхното взаимодействие.

Осигурява се добър анализ на ниво извън системата – указва се как системата си взаимодейства с актьорите, без да се притеснява за това как тази функционалност ще бъде приложена.



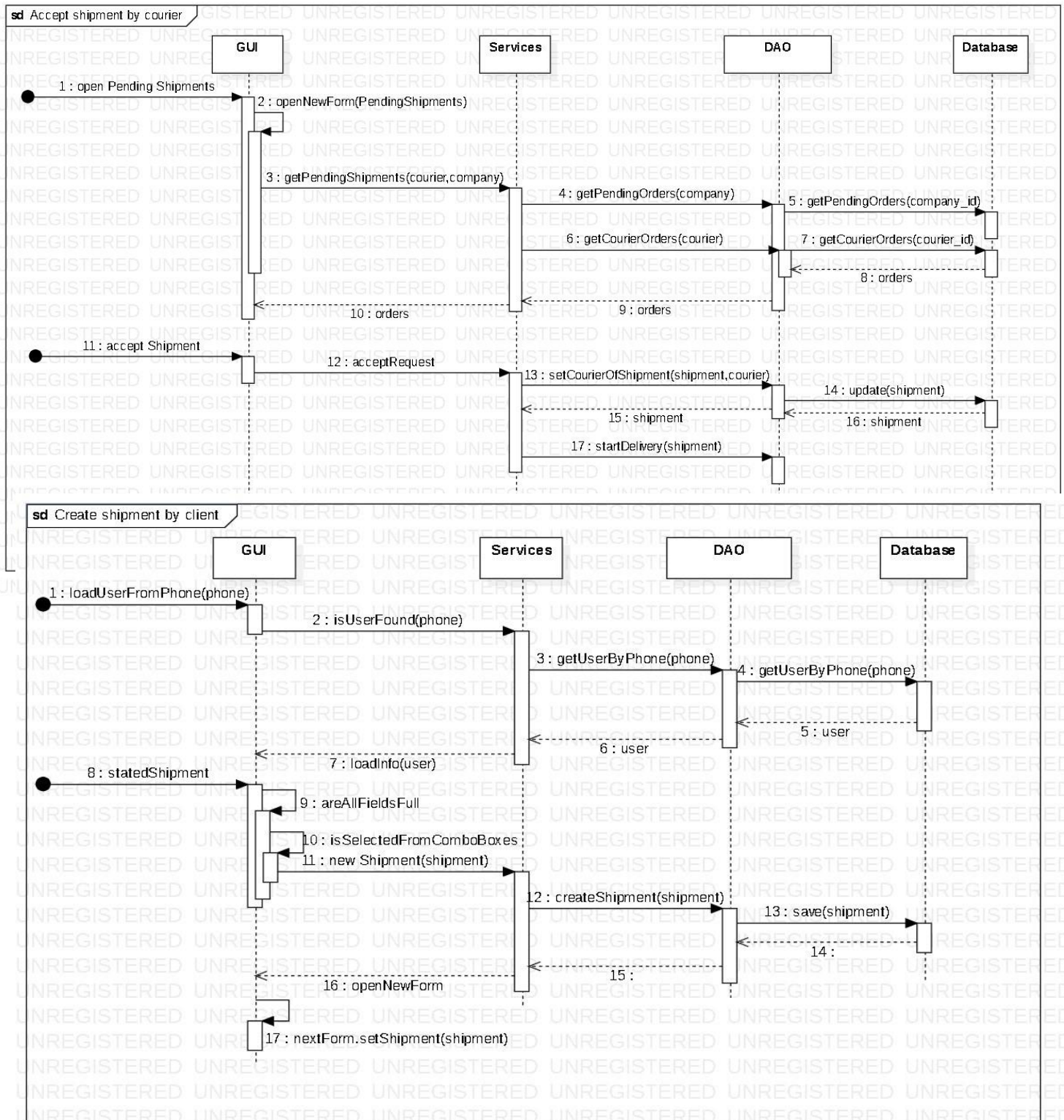
- **Class диаграма:**

Class диаграмата описва структурата на системата като показва нейните елементи и общите връзки (основните за същността на системата) между тях.



- **Sequence диаграми:**

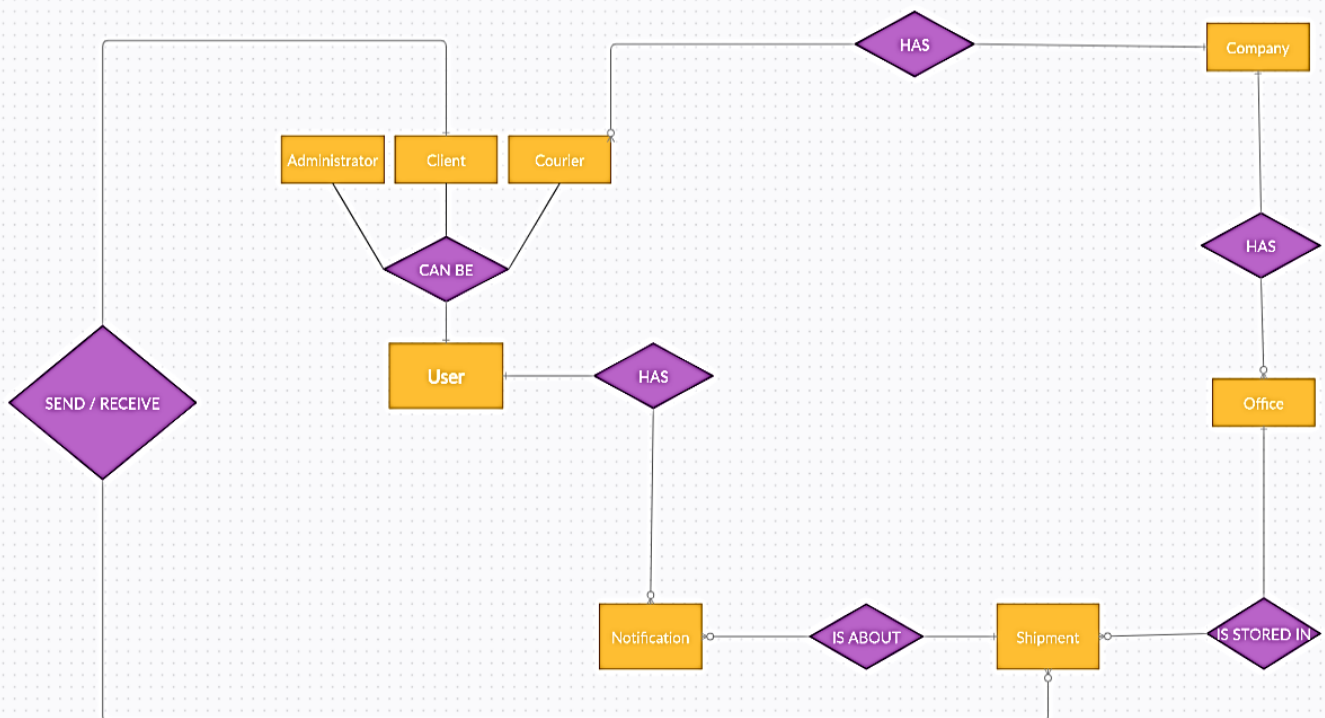
Наричани още диаграми на събития, sequence диаграмите подробно описват как се извършват операциите – как се изпълнява даден метод например. Тези диаграми са подходящи за визуализиране и валидация на различни сценарии на изпълнение.



3.3 ER диаграма – Entity Relationship diagram

ER диаграмата се използва за скициране на дизайна на базата данни. За разлика от други обозначения на диаграмата на ER, тази на Чен показва атрибутите като самостоятелни полета, а не като част от обекти.

CMSoftware

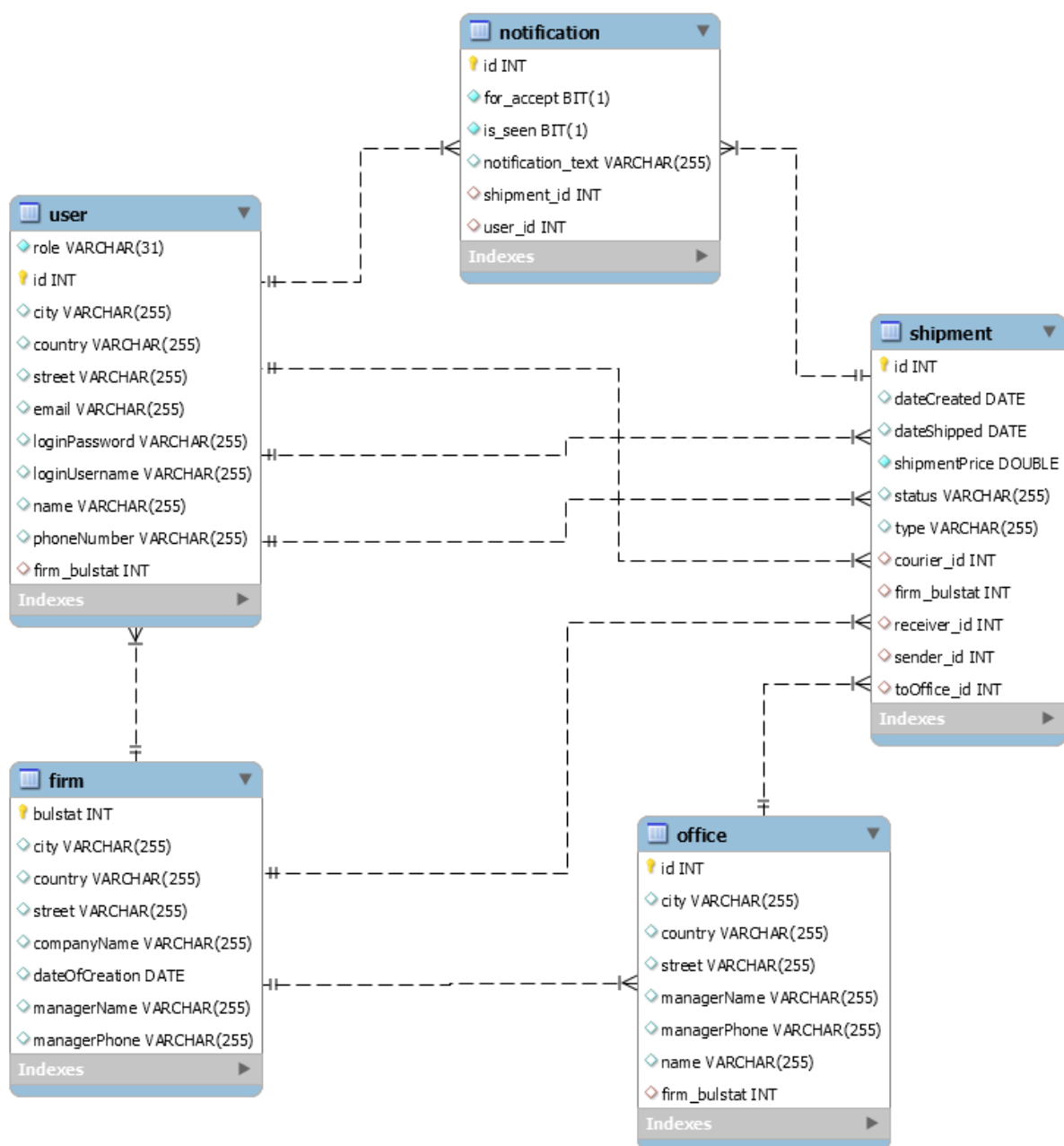


4. РЕАЛИЗАЦИЯ НА СИСТЕМАТА

4.1 Реализиране на базата от данни.

За реализирането на базата данни се използва MySQL и MySQL Workbench – инструмент за достъп до сървара.

Релационен модел:



- User – съхранява данни за потребител:
 - Име - name: VARCHAR;
 - Име за вход в системата - loginUsername: VARCHAR;
 - Парола за вход в системата - loginPassword: VARCHAR;
 - Адрес - полетата country, city, street: VARCHAR;
 - Ел. поща - email: VARCHAR;
 - Тел. номер - phoneNumber: VARCHAR.

Уникалността на всеки потребител се задава с ID полето.

Тази таблица има 3 подтаблицы - Admin, Courier и Client.

Те „разширяват“ User таблицата (решение на Java наследяване в релационен модел) и допълват функционалностите на различните роли в системата. (Като например: клиентът има получени пратки, куриерът се характеризира още с фирма, в която работи).

- Firm – съхранява данни за компания:
 - Име - companyName: VARCHAR;
 - Телефон и лице за контакти - полетата managerPhone и managerName: VARCHAR;
 - Дата на създаване - dateOfCreation: DATE;
 - Адрес – полетата country, city, street: VARCHAR.

Уникалността на всяка компания се задава с bulstat полето.

- Office Table – съхранява данни за офис:
 - Име – комбинация от името на фирмата и адреса на офиса name: VARCHAR;
 - Адрес – полетата country, city, street: VARCHAR;
 - Телефон и лице за контакти – полетата managerPhone и managerName: VARCHAR;
 - Компания към която е офисът – firm_bulstat: INT.

Уникалността на всеки офис се задава с ID полето.

- Shipment – съхранява данни за пратка:
 - Дата на създаване - dateCreated: DATE;
 - Дата на доставяне - dateShipped: DATE;
 - Цена на пратка – включва цената на пратката и цената на доставката - shipmentPrice: DOUBLE;
 - Състояние – на какъв етап от процеса на доставяне е пратката в даден момент - status: VARCHAR;
 - Тип - type: VARCHAR;
 - Куриер изпълняващ доставката - courier_id: INT;
 - Компания изпълняваща доставката - firm_bulstat: INT;
 - Получател на пратката - receiver_id: INT;
 - Изпращач на пратката - sender_id: INT;
 - Офис към който е изпратена пратката (ако е към офис) - toOffice_id: INT.

Уникалността на всяка пратка се задава с ID полето.

- Notification – съхранява данни за известията в системата:
 - Тип на известието(за обработка или не) - for_accept: BIT;
 - Състояние на известието(видно или не) - is_seen: BIT;
 - Текст на известието - notification_text: VARCHAR;
 - Пратка, за която се отнася известието - shipment_id: INT;
 - Получател на известието - user_id: INT.

Уникалността на всяко известие се задава с ID полето.

4.2 Реализиране на слоя за работа с базата данни.

За ORM инструмент се използва Hibernate: имплементация на JPA(**Java Persistence API**).

Hibernate предлага обекти SessionFactory, Session, но при реализацията се използват тези на JPA EntityManagerFactory, EntityManager. Идеята на това е, че смяната на ORM tool ще се случи по-лесно, тъй като всеки друг такъв би работил с обектите от JPA.

За свързване с база данни - persistence.xml.

```
▼ > src/main/resources
  ▼ > META-INF
    > persistence.xml
```

Установява връзка с:

- Основната за приложението база данни;
- Тестова база данни.

Клас entityManager:

```
▼ > tu_varna.project.courier_system.dao.em
  > > entityManager.java
```

Атрибути -JPA обекти :

```
private static EntityManagerFactory emf;
private static EntityManager entityManager;
```

За достъпване на базата данни - Аксесор за променлива `entityManager`.

За достъпване на базата данни чрез отваряне на транзакция и изпълнение на `EntityManager` операции (`persist`, `find`, `remove`, `merge`). –














```
public static void executeInsideTransaction(Consumer<EntityManager> action) {
    EntityTransaction tx = entityManager.getTransaction();
    try {
        tx.begin();
        action.accept(entityManager);
        tx.commit();
    } catch (RuntimeException e) {
        tx.rollback();
        throw e;
    }
}
```

Функция за инициализация на атрибутите-

```
public static void initEntityManager(String persistence_unit)
{
    emf = Persistence.createEntityManagerFactory(persistence_unit);
    entityManager = emf.createEntityManager();
}
```

където `persistence_unit` може да е връзката с главната база данни или с базата данни за тестове.

Пакет entity:

- ▼  > tu.varna.project.courier_system.entity
 - >  Address.java
 - >  > Admin.java
 - >  Client.java
 - >  Company.java
 - >  Courier.java
 - >  Manager.java
 - >  Notification.java
 - >  Office.java
 - >  Shipment.java
 - >  Status.java
 - >  Type.java
 - >  User.java

Класовете *Manager* и *Address* са с анотация `@Embeddable` (Данните им са колони в таблицата на entity-то, което има атрибут *Manager* или *Address*).

Класовете *Status* и *Type* съдържат enum променлива съответно **за статус:**

```
public class Status {  
    public enum status {  
        pending, delivered, in_proccess_of_delivery,  
        declined, accepted, taken_by_courier, in_proccess_of_return;  
    }  
}
```

и тип със съответната цена на доставка за различните типове пратки:

```
public class Type {  
    public enum type {  
        bag(5.50), packet(7.50), parcel(10.50), cargo(12.50), document(2.50);  
    }  
}
```

Те се изобразяват в базата данни като низове, благодарение на:

```
@Enumerated(EnumType.STRING)  
private Status.status status;
```

Останалите класове (същинските Entity класове) отговарят на таблици в базата данни. Всеки клас съдържа основните аксесори и мутатори, конструктори и методи, даващи възможността при прочитане на обект, да се прочетат сигурно и често ползваните данни (снижава се комуникация с базата данни).

Всеки клас има основни анотация @Entity и @Id:

```
@Entity  
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)  
@DiscriminatorColumn(name="role")  
public abstract class User {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private int id;  
  
    @Column(unique = true)  
    private String loginUsername;  
  
    private String loginPassword;  
  
    private String name;  
  
    private String email;  
  
    @Column(unique = true)  
    private String phoneNumber;  
  
    private Address address;
```


Анотации:

@GeneratedValue за автоматично генериране на id при следващ INSERT в базата данни.

@Column е пропусната, където има тази възможност - Hibernate създава колона от всеки един атрибут.

За връзките между таблиците използваме асоциации-

@OneToOne и **@OneToMany**. Това генерира съответните foreign key-ове в таблиците при създаване на базата данни от самите entity класове.

Използваме **FetchType.LAZY** за да избегнем ненужните заявки към базата данни.

CascadeType.DETACH - в случай че изтрием ред в таблицата не се изтриват редовете в другите таблици, които имат връзка с този ред. – например при изтриване на Notification.

CascadeType.REMOVE - когато изтрием ред от таблица изтриваме и другите свързани с него редове от другите таблици – например при изтриване на Company.

MappedBy, за да избегнем създаването на допълнителни таблици, които да правят връзката между entity-тата.

```
@OneToMany(cascade = CascadeType.REMOVE, mappedBy = "user")  
private List<Notification> notifications;
```

```
@ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.DETACH)  
private User user;
```

Пакет dao:

```
▼ 📁 > tu_varna.project.courier_system.dao
  > 📄 CompanyDao.java
  > 📄 NotificationDao.java
  > 📄 OfficeDao.java
  > 📄 ShipmentDao.java
  > 📄 UserDao.java
```

Съдържат базовите CRUD операции и допълнителни, свързани с изискванията на софтуера.

- **CompanyDao** - опериране с данни, отнасящи се до компании.

Запазване на компания-

```
boolean save(Company t);
```

Изтриване на компания-

```
void delete(Company t);
```

Получаване на компания по определени изисквания-

```
Company get(int id);
```

Актуализиране на компания-

```
void update(Company t);
```

Получаване списък от всички компании-

```
List<Object[]> getAllCompanies();
```

- **NotificationDao** - опериране с данни, отнасящи се до известия.

Запазване на известие-

```
boolean save(Notification t);
```

Изтриване на известие-

```
void delete(Notification t);
```

Получаване на известие по ID-

```
Notification get(int id);
```

Актуализиране на известие-

```
void update(Notification t);
```

Получаване списъци от всички известия на даден клиент-

```
List<Notification> getUserNotifications(int id);
```

- **OfficeDao** - опериране с данни, отнасящи се до офиси.

Запазване на офис-

```
boolean save(Office t);
```

Изтриване на офис-

```
void delete(Office t);
```

Получаване на офис по определени изисквания-

```
Office get(int id);
```

Актуализиране на офис-

```
void update(Office t);
```

Получаване списък от всички офиси/ офиси в компания-

```
List<Object[]> getAllOffices();
```

```
List<Object[]> getOfficesByFirm(int bulstat);
```

- **ShipmentDao** - опериране с данни, отнасящи се до поръчки.

Запазване на поръчка-

```
boolean save(Shipment t);
```

Изтриване на поръчка-

```
void delete(Shipment t);
```

Получаване на поръчка по определени изисквания-

```
Shipment get(int id);
```

Актуализиране на поръчка-

```
void update(Shipment t);
```

Получаване списъци от всички поръчки в зависимост от тяхното състояние-

```
List<Object[]> getExpectedShipments(int client_id);
```

```
List<Object[]> getRequestedShipments(int client_id);
```

```
List<Object[]> getCourierDeliveredShipments(int courier_id);
```

```
List<Object[]> getCourierActiveShipments(int courier_id);
```

```
List<Object[]> getShipmentsByCompany(int bulstat);
```

```
List<Object[]> getPendingOrdersByCompany(int bulstat);
```

Получаване броя на успешните/неуспешните пратки-

```
long getSuccessfulOrders(int bulstat);
```

```
long getUnsuccessfulOrders(int bulstat);
```

- **UserDao** - опериране с данни, отнасящи се до потребители.

Запазване на потребител-

```
boolean save(User t);
```

Изтриване на потребител-

```
void delete(User t);
```

Актуализиране на потребител-

```
void update(User t);
```

Получаване на потребител по определени изисквания-

```
User get(int id);
```

```
User getUserByPhone(String phoneNmb);
```

```
User getUserByUsername(String name);
```

Получаване списък от всички клиенти/куриери-

```
List<Object[]> getAllClients();
```

```
List<Object[]> getAllCouriers();
```

Dao класовете използват статичните методи на entityManager executeInsideTransaction или getEntityManager за да правят достъп до базата данни.

4.3 Реализиране на бизнес логика.

```
▼ > tu_varna.project.courier_system.services
  > > CompanyService.java
  > > LoginService.java
  > > NotificationService.java
  > > ShipmentDeliveryService.java
  > > ShipmentService.java
  > > UserService.java
```

- **CompanyService** – поддържа работата с компании.

Създаване на компания –

```
public boolean createCompany(Company company)
```

Изтриване на компания –

```
public void deleteCompany(Company company)
```

Получаване на списък от всички компании –

```
public List<CompanyView> getAllCompanies()
```

Търсене на компания –

```
public Company getCompanyByID(int id)
```

Създаване на офис към компания –

```
public boolean createOffice(Office office)
```

Изтриване на офис към компания –

```
public void deleteOffice(Office office)
```

Получаване на списък от всички офиси за дадена компания –

```
public List<OfficeView> getOfficesList(int bulstat)
```

Получаване на списък от всички офиси –

```
public List<OfficeView> getAllOffices()
```

Търсене на офис -

```
public Office getOfficeById(int id)
```

- **UserService** - поддържа работа с потребители.

Създаване на потребител-

```
public boolean createClient(Client client)
```

```
public boolean createCourier(Courier courier)
```

Изтриване на потребител-

```
public void deleteUser(User usr)
```

Търсене на потребител –

```
public User getUserByPhone(String phoneNmb)
```

```
public User getUserByID(int id)
```


Редактиране на данни (парола, ел.поща, адрес, телефонен номер т.н)-

```
public void changeUserAddress(User user, String country, String city, String street)
```

```
public void changeUserEmail(User user, String email)
```

```
public void changeUserPassword(User user, String password)
```

```
public void changeUserPhone(User user, String phone)
```

Получаване на списък от всички потребител –

```
public List<ClientView> getAllClients()
```

```
public List<CourierView> getAllCouriers()
```

- **ShipmentService** - поддържа работа с пратки.

Създаване на пратка-

```
public boolean createShipment(Shipment shipment)
```

Изтриване на пратка-

```
public void deleteShipment(Shipment shipment)
```

Промяна състоянието на пратка-

```
public void changeShipmentStatus(Shipment shipment, status status)
```

Търсене -

```
public Shipment getShipmentByID(int id)
```

Получаване на списък от заявени пратки на даден клиент Impl-

```
@Override
public List<ShipmentView> getRequestedShipments(Client client)
{
    List<ShipmentView> toReturn = new
    ArrayList<ShipmentView>();
    List<Object[]> list =
shipmentDao.getRequestedShipments(client.getId());
    for (Object[] column : list) {
        int id = (Integer) column[0];
        User receiver = (User) column[1];
        Company company = (Company) column[2];
        Status.status status = (Status.status) column[3];

        toReturn.add(new ShipmentView(id,
receiver.getName(), company.getCompanyName(), status));
    }
    return toReturn;
}
```

Получаване на списък от очаквани пратки на даден клиент-

```
public List<ShipmentView> getExpectedShipments(Client client)
```

Получаване на списък от пратки чакащи да бъдат изпратени-

```
public List<ShipmentView> getPendingOrders(Company company)
```

Получаване на списък от доставени пратки от даден куриер-

```
public List<ShipmentView> getCourierDeliveredShipments(Courier
courier)
```

Получаване на списък от активни пратки на даден куриер-

```
public List<ShipmentView> getCourierActiveShipments(Courier courier)
```

Получаване броя на успешно/неуспешно изпратените пратки-

```
public long getOrders(Company company, boolean isSuccessful)
```

Възлагане пратка на куриер-

```
public void setCourierOfShipment(Shipment shipment, Courier courier)
```

- **LoginService** – поддържа работа при влизане в системата.

Проверка на въведената информация за вход Imp-

```
public User authenticateUserLogin(String username, String password) {  
    User user = userDao.getUserByUsername(username);  
    if (user != null) {  
        if (user.getLoginPassword().equals(password)) {  
            return user;  
        }  
        return null;  
    }  
    return null;  
}
```

- **NotificationService** – поддържа работа с известия в системата.

Изпращане на известия до клиент-

```
@Override
    public void sendNotification(String text, Client client) {
        ClientWorkspaceFormController next = new
        ClientWorkspaceFormController();
        Image img = new
        Image("tu_varna/project/courier_system/img/notification.png");
        Notifications not = Notifications.create();
        not.title("Notification").text(text).hideAfter(new
        Duration(5000)).hideCloseButton().owner(next.getPane())
        .graphic(new ImageView(img)).onAction(new
        EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                FXMLLoader loader =
                OpenNewForm.openNewForm("NotificationsForm.fxml",
                "Notifications");
                NotificationsFormController next =
                loader.getController();

                next.setNotifications(getListNotifications(client));
            }
        });
        not.show();
    }
```

Получаване на списък от всички известия за текущ клиент-

```
public List<Notification> getListNotifications(Client client)
```

Промяна състоянието на известие-

```
public void setSeenStatus(Notification notification)
```

Изтриване на известие-

```
public void deleteNotification(Notification notification)
```

Търсене на известие-

```
public Notification getNotificationByID(int id)
```

Обработка на известие-

```
@Override
    public void handleUserNotificationAnswer(boolean answer,
Notification notification) {
        Shipment shipment = notification.getShipment();
        if (answer == true) {
            shipment.setStatus(status.accepted);
            shipmentDao.update(shipment);

        } else if (answer == false) {
            shipment.setStatus(status.declined);
            shipmentDao.update(shipment);
            Shipment new_shipment = new
Shipment(status.in_process_of_return, shipment.getType(),
LocalDate.now(),
                0.00, shipment.getReceiver(),
shipment.getSender(), shipment.getFirm());
            shipmentDao.save(new_shipment);
            notificationDao.save(new
Notification(shipment.getSender(),
                shipment.getReceiver().getName() + "
declined your shipment! A courier will bring it back to you!",
                false));
        }
        notification.setIsSeen(true);
        notificationDao.update(notification);
    }
}
```

- **ShipmentDeliveryService** - доставяне на пратката от адрес до адрес или от адрес до офис.

Изчисляване на времето за доставка-

```
private int calculateDeliveryTime()
{
    Address senderAddress=
this.shipment.getSender().getAddress();
    Address receiverAddress=
this.shipment.getReceiver().getAddress();

    if(senderAddress.getCountry().equalsIgnoreCase(receiverAddress.
getCountry()))
    {

        if(senderAddress.getCity().equalsIgnoreCase(receiverAddress.get
City()))
        {
            return 5000;
        }
        return 10000;
    }
    return 15000;
}
```

Функция за стартиране на нишката-

```
public void startDelivery()
{
    Thread t = new Thread(this);
    t.start();
}
```

Имплементация на run() -

```
@Override
public void run() {
    boolean isAcceptable = true;
    if (shipment.getStatus() == status.in_process_of_return)
    {
        isAcceptable = false;
    }
}
```

```

    }
    try {
        int deliveryTime= calculateDeliveryTime();
        shipment.setStatus(status.taken_by_courier);
        shipmentDao.update(shipment);
        Thread.sleep(deliveryTime);
        shipment.setStatus(status.in_proccess_of_delivery);
        shipmentDao.update(shipment);
        Thread.sleep(deliveryTime);
        shipment.setStatus(status.delivered);
        shipment.setDateShipped(LocalDate.now());
        shipmentDao.update(shipment);

    } catch (InterruptedException e) {

        e.printStackTrace();
    }
    if (isAcceptable == true) {
        sendNotification("A " +
this.shipment.getType().toString() + " from " +
this.shipment.getSender().getName()
        + " has been send to you. Due amount: " +
this.shipment.getShipmentPrice(), true);
    } else {
        sendNotification("A " +
this.shipment.getType().toString() + " from " +
this.shipment.getSender().getName()
        + " has been returned back to you. ",
false);

        shipment.setStatus(status.accepted);
        shipmentDao.update(shipment);

    }

}

```

Изпращането на известие след приключване на доставяне-

```

private void sendNotification(String text, boolean
isAcceptable) {

    Notification notification = new
Notification(this.shipment.getReceiver(), text, isAcceptable,
this.shipment);
    notificationDao.save(notification);
    notification = new
Notification(this.shipment.getSender(),

```

```

        "Your shipment to " +
shipment.getReceiver().getName() + " was successfully
delivered!", false);
        notificationDao.save(notification);
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                if (ClientWorkspaceFormController.getID()
== userDao.get(shipment.getReceiver().getId()).getId()) {

                    notificationService.sendNotification("A shipment was
delivered to you right now!",
                                                        (Client)
userDao.get(shipment.getReceiver().getId()));
                }
                if (ClientWorkspaceFormController.getID()
== userDao.get(shipment.getSender().getId()).getId()) {

                    notificationService.sendNotification(
                                                        "Your shipment to " +
shipment.getReceiver().getName() + " was successfully
delivered!",
                                                        (Client)
userDao.get(shipment.getSender().getId()));
                }
            }
        });
    }
}

```


4.4 Реализиране на графичен интерфейс.

За изграждане на GUI се използва JavaFX – набор от GUI инструменти за Java приложения и Scene Builder-инструмент за визуално оформление на приложението.

- View

- ▼ > src/main/resources
 - > META-INF
 - ▼ tu_varna
 - ▼ project
 - ▼ courier_system
 - > img
 - ▼ view
 - AboutCourierCompanyForm.fxml
 - AboutCourierForm.fxml
 - AboutShipmentForm.fxml
 - administratorStyle.css
 - AdministratorWorkspaceForm.fxml
 - BillOfLoadingForm.fxml
 - ClientProfileForm.fxml
 - ClientShipmentsForm.fxml
 - ClientStatisticsForm.fxml
 - ClientWorkspaceForm.fxml
 - CompanyStatisticsForm.fxml
 - ControlClientForm.fxml
 - ControlCompanyForm.fxml
 - CourierControlForm.fxml
 - CourierHomeForm.fxml
 - CourierProfileForm.fxml
 - CourierWorkspaceForm.fxml
 - CreateClientForm.fxml
 - CreateCompanyForm.fxml
 - CreateCourierForm.fxml
 - CreateOfficeForm.fxml
 - DeliveredShipmentsForm.fxml
 - DoYouAcceptShipmentForm.fxml
 - ExpectedShipmentsForm.fxml
 - NotificationsForm.fxml
 - OfficesControlForm.fxml
 - PendingShipmentsForm.fxml
 - RequestedShipmentsForm.fxml
 - RequestShipmentForm.fxml
 - ReturnedShipmentForm.fxml
 - ShipmentControlForm.fxml
 - ShipmentDetailsForm.fxml
 - ShipmentStateForm.fxml
 - tableView.css
 - TrackShipmentForm.fxml
 - ViewShipmentsForm.fxml
 - WelcomeForm.fxml
 - YourProgressForm.fxml



- Controllers

Контролер с най-голяма логическа тежест е **RequestShipmentFormController**:



Инициализация (някой контроли при зареждане на формата)-

```
public void initialize(URL arg0,  
ResourceBundle arg1)
```

Валидации на полетата –

```
private void countryValidation(KeyEvent  
event)
```

```
private void emailValidation(KeyEvent  
event)
```

```
private void nameValidation(KeyEvent  
event)
```

```
private void priceValidation(KeyEvent  
event)
```

```
private void streetNValidation(KeyEvent  
event)
```

```
private boolean numberValidation()
```

```
tu_varna.project.courier_system.controllers  
> AboutCourierCompanyFormController.java  
> AboutCourierFormController.java  
> AboutShipmentFormController.java  
> AdminFormController.java  
> BillOfLoadingFormController.java  
> ClientControlFormController.java  
> ClientProfileFormController.java  
> ClientStatisticsFormController.java  
> ClientWorkspaceFormController.java  
> CompanyControlFormController.java  
> CompanyStatisticsFormController.java  
> CourierControlFormController.java  
> CourierHomeFormController.java  
> CourierProfileFormController.java  
> CourierWorkspaceFormController.java  
> CreateClientFormController.java  
> CreateCompanyFormController.java  
> CreateCourierFormController.java  
> CreateOfficeFormController.java  
> DeliveredShipmentsFormController.java  
> DoYouAcceptShipmentFormController.java  
> ExpectedShipmentsFormController.java  
> NotificationsFormController.java  
> OfficesControlFormController.java  
> PendingShipmentsFormController.java  
> RequestedShipmentsFormController.java  
> RequestShipmentFormController.java  
> ReturnedShipmentFormController.java  
> ShipmentControlFormController.java  
> ShipmentDetailFormController.java  
> ShipmentsViewFormController.java  
> TrackShipmentFormController.java  
> WelcomeFormController.java  
> YourProgressFormController.java
```

Запълване на combo boxes-

```
public void fillCompanyCombo(CompanyView company)
```

```
public void fillOfficeCombo(OfficeView office)
```

Активиране на полета-

```
private void activatePersonalInfoFields()
```

```
private void activateShipmentInfoFields()
```

Активиране/деактивиране на office combo box –

```
private void disabledOfficeCombo(ActionEvent event)
```

```
private void activateOfficeCombo(ActionEvent event)
```

Зареждане на компания чрез която е избрано да се осъществи поръчката
– в случай, че формата е отворена с правата на клиент се активира
ComboBox даващ право на избор, в противен случай този избор не се
предоставя, тъй като с права администратор
е ясно към коя компания се изпраща
(предварително избрана от TableView).

```
public void getCompanyForAdmin(Company choosedCompany)
```

```
public void getCompanyForClient(User sender)
```

Валидация на всички данни необходими за регистране на пратка-

```
private boolean areAllFieldsFull()
```

```
private boolean isSelectedFromComboBoxes()
```

```
private boolean isSelectedFromRadioButtons()
```

Запълване на полетата с данни за получател-

```
public void loadInfo(String name, String email, String country,  
String city, String streetN)
```

Проверка за съществуващ клиент(потребител) с този тел. номер -

```
private boolean isUserFound()
```

Зареждане на данни на вече съществуващ клиент-

public void loadClientInfo(ActionEvent event) – използва:

- **private boolean** isUserFound()
- **public void** loadInfo(String name, String email, String country, String city, String streetN)
- **private void** activatePersonalInfoFields()
- **private void** activateShipmentInfoFields()

Заявяване на пратката-

@FXML

```
private void statedShipment(ActionEvent event) {
    this.phoneValidationLabel.setText("");
    this.searchResultLabel.setText("");
    this.resultLabel.setText("");

    if (areAllFieldsFull() && isSelectedFromComboBoxes() &&
        isSelectedFromRadioButtons()) {

        RadioButton selectedRadioButtonfromSendTo = (RadioButton)
sendTo.getSelectedToggle();
        RadioButton selectedRadioButtonfromExpenseOf =
(RadioButton) expenseOf.getSelectedToggle();
        String name = this.name.getText();
        String phoneNmb = this.phoneNmb.getText();
        String email = this.email.getText();
        String country = this.country.getText();
        String city = this.city.getText();
        String streetN = this.streetN.getText();
        Double price = Double.parseDouble(this.price.getText());
        OfficeView officeView;
        CompanyView companyView =
this.companyCombo.getSelectionModel().getSelectedItem();
        Company company =
companyService.getCompanyByID(companyView.getBulstat());
        type type =
this.typeCombo.getSelectionModel().getSelectedItem();
        Shipment shipment = null;
        Office office = null;

        if
(selectedRadioButtonfromSendTo.getText().equals("address")) {
            officeView = null;
        }

        else {
```

```

        officeView =
this.officeCombo.getSelectionModel().getSelectedItem();
        office =
companyService.getOfficeById(officeView.getCode());
    }

    if
(selectedRadioButtonfromExpenseOf.getText().equals("receiver")) {
        dueAmount.setText("0.00");
        price = price + type.showPrice();
    } else {

        dueAmount.setText(Double.toString(type.showPrice()));
    }
    if (receiver != null && officeView != null) {

        shipment = new Shipment(type,
dateCreation.getValue(), price, sender, receiver, company, office);
        shipmentService.createShipment(shipment);
    }
    if (receiver != null && officeView == null) {
        shipment = new Shipment(type,
dateCreation.getValue(), price, sender, receiver, company);
        shipmentService.createShipment(shipment);
    }
    if (receiver == null && officeView != null) {
        receiver = new Client(phoneNmb, phoneNmb, name,
email, phoneNmb, country, city, streetN);
        userService.createClient((Client) receiver);
        shipment = new Shipment(type,
dateCreation.getValue(), price, sender, receiver, company, office);
        shipmentService.createShipment(shipment);
    }
    if (receiver == null && officeView == null) {
        receiver = new Client(phoneNmb, phoneNmb, name,
email, phoneNmb, country, city, streetN);
        userService.createClient((Client) receiver);
        shipment = new Shipment(type,
dateCreation.getValue(), price, sender, receiver, company);
        shipmentService.createShipment(shipment);
    }

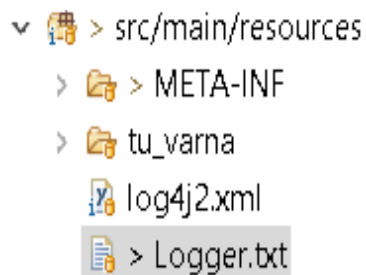
    LOGGER.info("Client with id: " + sender.getId() + "
requested a new shipment!");
    resultLabel.setText("You requested shipment
succsefully!");
    FXMLLoader loader =
OpenNewForm.openNewForm("BillofLoadingForm.fxml", "Successful request");
    BillofLoadingFormController next =
loader.getController();
    next.setCreatedShipment(shipment);
    }
    }

```

4.5 Реализиране на модул за регистране на събития в системата.

За регистриране на събитията се използва log4j 2.

За целта - **log4j2.xml** настройващ файл и **Logger.txt** – съхраняващ събитията файл.



Чрез xml файла се ограничени конзолните съобщения от hibernate и е създаден нов файл -Logger.txt в който се добавя INFO информация със структура :

<дата><контролер на събитието><информация за събитието>

Пример:

```
130-11-2020 23:16:49 tu_varna.project.courier_system.controllers.ClientWorkspaceFormController - Client with id: 3 successfully logged in!
230-11-2020 23:17:05 tu_varna.project.courier_system.controllers.RequestShipmentFormController - Client with id: 3 requested a new shipment!
330-11-2020 23:17:08 tu_varna.project.courier_system.controllers.ClientWorkspaceFormController - Client with id: 3 successfully logged out!
430-11-2020 23:17:14 tu_varna.project.courier_system.controllers.CourierWorkspaceFormController - Courier with id: 4 successfully logged in!
530-11-2020 23:17:18 tu_varna.project.courier_system.controllers.PendingShipmentsFormController - Shipment with id: 3 has been taken by courier with id: 4
```

Контролерите, които правят промени, имат обект от тип Logger, което дава възможността за отразяване на желаните събития.








```
private static final Logger logger = LogManager.getLogger(AdminFormController.class);
```

Съхранява се, освен хронология на промяна в базата данни, още и хронология на влизанията и излизанията от системата на различните потребители:







```
@FXML
private void logOut(ActionEvent event) throws IOException {
    CloseForm.closeForm(event);
    OpenNewForm.openNewForm("WelcomeForm.fxml", "Welcome");
    logger.info("Administrator successfully logged out!");
}
```

4.6 Допълнителни пакети в системата.

- **tableView**– осигурява изглед на обекти в таблици.

```
✓  > tu_varna.project.courier_system.tableView
  >  > ClientView.java
  >  > CompanyView.java
  >  > CourierView.java
  >  > NotificationsView.java
  >  > OfficeView.java
  >  > ShipmentView.java
```

- **helper** – помощни класове.

```
✓  tu_varna.project.courier_system.helper
  >  BuiltInForm.java
  >  CloseForm.java
  >  DataValidation.java
  >  FieldValidation.java
  >  OpenNewForm.java
```

BuiltInForm – използва се за вграждане на една fxml форма в друга.

OpenNewForm – използва се за отваряне на fxml форма в нов прозорец.

CloseForm – използва се за затваряне на форма при натискане на бутон.

DataValidation – съдържа функции за валидации на данни. Използва regex.

Проверка на дължината на данната-

```
public static boolean dataLength(TextField inputTextField, Label  
inputLabel, String validationText, int requiredLength)
```

Проверка дали данната е с формат на ел.адрес -

```
public static boolean emailFormat(TextField inputTextField, Label  
inputLabel, String validationText)
```

Проверка дали данната е с формат на потребителско име -

```
public static boolean isUsername(TextField inputTextField, Label  
inputLabel, String validationText)
```

Проверка дали данната е с формат цена -

```
public static boolean priceFormat(TextField inputTextField, Label  
inputLabel, String validationText)
```

Проверка дали данната е с формат улица + адресен номер -

```
public static boolean streetNFormat(TextField inputTextField, Label  
inputLabel, String validationText)
```

Проверка дали данната е само с позволените буквени символи от латиницата -

```
public static boolean textAlphabet(TextField inputTextField, Label  
inputLabel, String validationText)
```


Проверка дали данната носи информация -

```
public static boolean textFieldIsEmpty(TextField inputTextField,  
Label inputLabel, String validationText)
```

Проверка дали данната е с числов формат -

```
public static boolean textNumeric(TextField inputTextField,  
Label inputLabel, String validationText)
```

FieldValidation – всяка една функция валидира полета в системата, предназначени да приемат определени данни, като комбинира няколко проверки от класа `DataValidation` и на определени места (основни за работата на потребителя) тази валидация е синхронна с работата на потребителя.

Пример:

Валидация на поле, предназначено да приема ел.поща-

```
public static boolean emailValidation(TextField field,  
Label validationLabel) {  
    boolean isEmpty =  
DataValidation.textFieldIsEmpty(field, validationLabel,  
"The field is empty.");  
    if (!isEmpty) {  
        boolean isEmail =  
DataValidation.emailFormat(field, validationLabel, "Wrong  
email format.");  
        return isEmail;  
    }  
    return false;  
}
```

5. ТЕСТВАНЕ НА СИСТЕМАТА

Unit тестване

Тестван е слойт реализиращ бизнес логиката.
Използва се Junit и Mockito за тестването.

За проверка на резултатите от тестовете –

assertEquals(), assertNotNull() на връщаните от функциите стойности.

```
▼ > tu_varna.project.courier_system.services
  > > CompanyServiceTest.java
  > > LoginServiceTest.java
  > > NotificationServiceTest.java
  > > ShipmentServiceTest.java
  > > UserServiceTest.java
```

За проверка дали самият Services метод извиква правилния DAO метод с правилните данни, без да се интересуваме дали DAO методът работи коректно (DAO методите са тествани чрез функционални тестове), използваме анотация @Mock за да mock-нем обект и да зададем връщан от него желан резултат:

```
@Mock
CompanyDaoImpl companyDao;
```

```
@Mock
OfficeDaoImpl officeDao;
```

Чрез анотация @BeforeEach към функция Init() във всеки тест инициализираме обектите, съответно от Services и Dao класовете, отново при всеки следващ тест:

```
@BeforeEach
void init() {
    companyService = new CompanyServiceImpl();
    companyService.setDaos(companyDao, officeDao);
}
```

Mockito команди:

when()- когато метод от DAO клас се извика с посочен параметър да върне желан от нас резултат с **thenReturn()**:

```
when(companyDao.get(1)).thenReturn(company);
```

verify()- проверява дали метод от DAO клас с зададен параметър се извиква от Service метод:

```
verify(companyDao).get(1);
```

Примерен тест - тества се функцията за зареждане на компания по id:

```
@Test
void testGetCompanyByID() {

    final Company company = new Company();
    company.setId(1);
    when(companyDao.get(1)).thenReturn(company);
    Company returnedCompany = companyService.getCompanyByID(1);
    assertNotNull(returnedCompany);
    verify(companyDao).get(1);

}
```

Тестове на UserService

Създаване на клиент-

```
void testCreateClient()
```

Създаване на куриер-

```
void testCreateCourier()
```

Изтриване на потребител-

```
void testDeleteUser()
```

Зареждане на потребител по Id-

```
void testGetUserById()
```

Промяна на телефон на потребител-

```
void testChangeUserPhone()
```

Тестове на CompanyService

Зареждане на компания по ид-

```
void testGetCompanyById()
```

Изтриване на компания-

```
void testDeleteCompany()
```

Създаване на компания-

```
void testCreateCompany()
```

Създаване на офис-

```
void testCreateOffice()
```

Изтриване на офис-

```
void testDeleteOffice()
```

Зареждане на офис по ид-

```
void testGetOfficeById()
```

Тестове на LoginService

Проверка за правилно потребителско име и парола-

```
void testAuthenticateUserLogin()
```

Тестове на NotificationService

Зареждане на известие по id-

```
void testGetNotificationByID()
```

Изтриване на известие-

```
void testDeleteNotification()
```

Зареждане списък от известията на клиент-

```
void testGetListNotifications()
```

Тестове на ShipmentService:

Зареждане на пратка по ид-

```
void testGetShipmentByID()
```

Задаване куриер на пратка-

```
void testSetCourierOfShipment()
```

Изтриване на пратка-

```
void testDeleteShipment()
```

Създаване на пратка-

```
void testCreateShipment()
```

Промяна статуса на пратка-






```
void testChangeShipmentStatus()
```

Изпълнение на тестовете чрез Run:

Runs: 20/20

✖ Errors: 0

✖ Failures: 0

- >  CompanyServiceTest [Runner: JUnit 5] (0.467 s)
- >  NotificationServiceTest [Runner: JUnit 5] (0.029 s)
- >  ShipmentServiceTest [Runner: JUnit 5] (0.030 s)
- >  LoginServiceTest [Runner: JUnit 5] (0.017 s)
- >  UserServiceTest [Runner: JUnit 5] (0.011 s)

Функционално тестване

Тества се функционалността на DAO слоя.

```
▼ > tu_varna.project.courier_system.dao
  > > CompanyDaoTest.java
  > > NotificationDaoTest.java
  > > OfficeDaoTest.java
  > > ShipmentDaoTest.java
  > > UserDaoTest.java
```

Проверява се дали заявките към базата данни, правени от методите на DAO класовете, връщат желания резултат. За тестването се създава допълнителна “вградена” (in memory) база данни – H2, която се генерира преди всеки тест, като това позволява да се тества върху празна база данни.

„persistence_test” е името на unit-а за H2 база данни, която е конфигурирана да се генерира при всяко инициализиране на EntityManager за нея.

Чрез анотация @BeforeEach се инициализира базата данни и обекта от клас DAO, който ще прави промени в базата:

```
@BeforeEach
void init() {
    companyDao=new CompanyDaoImpl();
    entityManager.initEntityManager("persistence_test");
}
```

Тестовите на DAO layer са за CRUD операциите и някои допълнителни, които задоволяват изискванията на софтуера.

Примерен тест – запазване на компания и връщане на id като резултат:

```
@Test
void testSaveAndThenGet() {
    Company company = new Company();
    company.setId(123);
    companyDao.save(company);
    Company dbCompany = companyDao.get(123);
    assertNotNull(dbCompany);
    assertEquals(company.getId(), dbCompany.getId());
}
```

Тестване на UserDao:

Записване и зареждане на потребител-

```
void testSaveAndGetAfter()
```

Записване и промяна на потребител-

```
void testSaveAndThenUpdate()
```

Изтриване на потребител-

```
void testDelete()
```

Промяна телефона на потребител-

```
void testGetUserByPhone()
```

Зареждане на потребител по username-

```
void testGetUserByUsername()
```


Зареждане на списък от клиенти-

```
void testGetAllClients()
```

Зареждане на списък от куриери-

```
void testGetAllCouriers()
```

Тестване на CompanyDao:

Записване и зареждане на компания-

```
void testSaveAndThenGet()
```

Записване и промяна на компания-

```
void testSaveAndThenUpdate()
```

Изтриване на компания-

```
void testDelete()
```

Зареждане на списък от всички компании-

```
void testGetAllCompanies()
```

Тестване на NotificationDao:

Записване и зареждане на известие-

```
void testSaveAndThenGet()
```

Промяна и зареждане на известие-

```
void testUpdateAndThenGet()
```

Изтриване на известие-

```
void testDelete()
```

Зареждане списък от известията на потребител-

```
void testGetUserNotifications()
```

Тестване на OfficeDao:

Записване и зареждане на офис-

```
void testSaveAndThenGet()
```

Записване и промяна на офис-

```
void testSaveAndThenUpdate()
```

Изтриване на офис-

```
void testDelete()
```

Зареждане списък от всички офиси-

```
void testGetAllOffices()
```

Зареждане списък от офисите на фирма-

```
void testGetOfficesByFirm()
```

Тестване на ShipmentDao:

Записване и зареждане на пратка-

```
void testSaveAndThenGet()
```

Записване и промяна на пратка-

```
void testUpdateAndThenGet()
```

Изтриване на пратка-

```
void testDelete()
```

Зареждане на броя успешни пратки-

```
void testGetSuccessfulOrders()
```

Зареждане на броя неуспешни пратки-

```
void testGetUnsuccessfulOrders()
```

Изпълнение на тестовете чрез Maven build:

```
Results :
```

```
Tests run: 44, Failures: 0, Errors: 0, Skipped: 0
```

6. ДЕМОНСТРАЦИЯ

1. Заявяване на пратка:

с.л.1

с.л.2

с.л.3

С права на администратор, тъй като предварително е избрано в коя компания се правят промени

С права на клиент

RECEIVER:

087878788

Load

No client with this phone number.

Personal information:

The field is empty.

email

Address information:

country city

street and №

0896145600

Load

Personal information:

Desislava Georgieva

desi@gmail.com

Address information:

Bulgaria Kotel

Ivan Ivanov №84

RECEIVER:

phone number

Load

Not a clients number!

Personal information:

name

email

Address information:

country city

street and №

Successful request

Econt

12314

Sender: Vladimir Kolev
Bulgaria Burgas Rilska №5

Receiver: Petra Petrova
Bulgaria Kotel Georgi Genov #7

Type: document

Price: 0.0

Sender: //

Receiver: //

20

CASH ON DELIVERY

Courier: //

2020-12-05

RECEIVER:

087878788

Load

Personal information:

Petra Petrova

P_P788@gmail.com

Address information:

Bulgaria Kotel

Georgi Genov #7

DELIVERY INFORMATION:

Courier Company: Econt

Send to: address office

Office: Econt Georgi Genov №7

Type: document

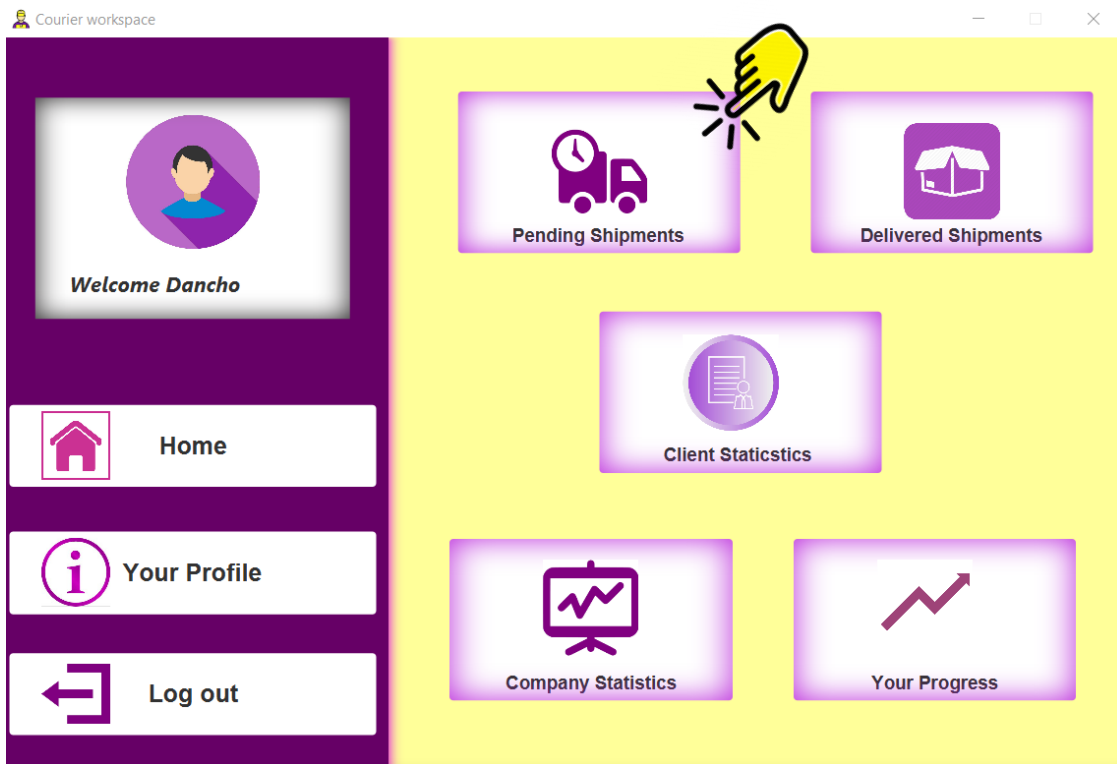
Shipment price: 0

Due amount: 2.5

Stated

You requested shipment successfully!

2. Поемане на пратката от куриер:



Pending Shipments

Shipment №	From	To
20	Bulgaria Burgas Rilska №5	Bulgaria Kotel Georgi Genov #7
19	address address address	Bulgaria Kotel Georgi Genov #7
18	address address address	Bulgaria Varna Dobrovnik #13

Accept

Shipment №	From	To
------------	------	----

No content in table

Details

Pending Shipments

Shipment №	From	To
21	Bulgaria Burgas Rilska №5	Bulgaria Kotel Georgi Genov #7
19	address address address	Bulgaria Kotel Georgi Genov #7
18	address address address	Bulgaria Varna Dobrovnik #13

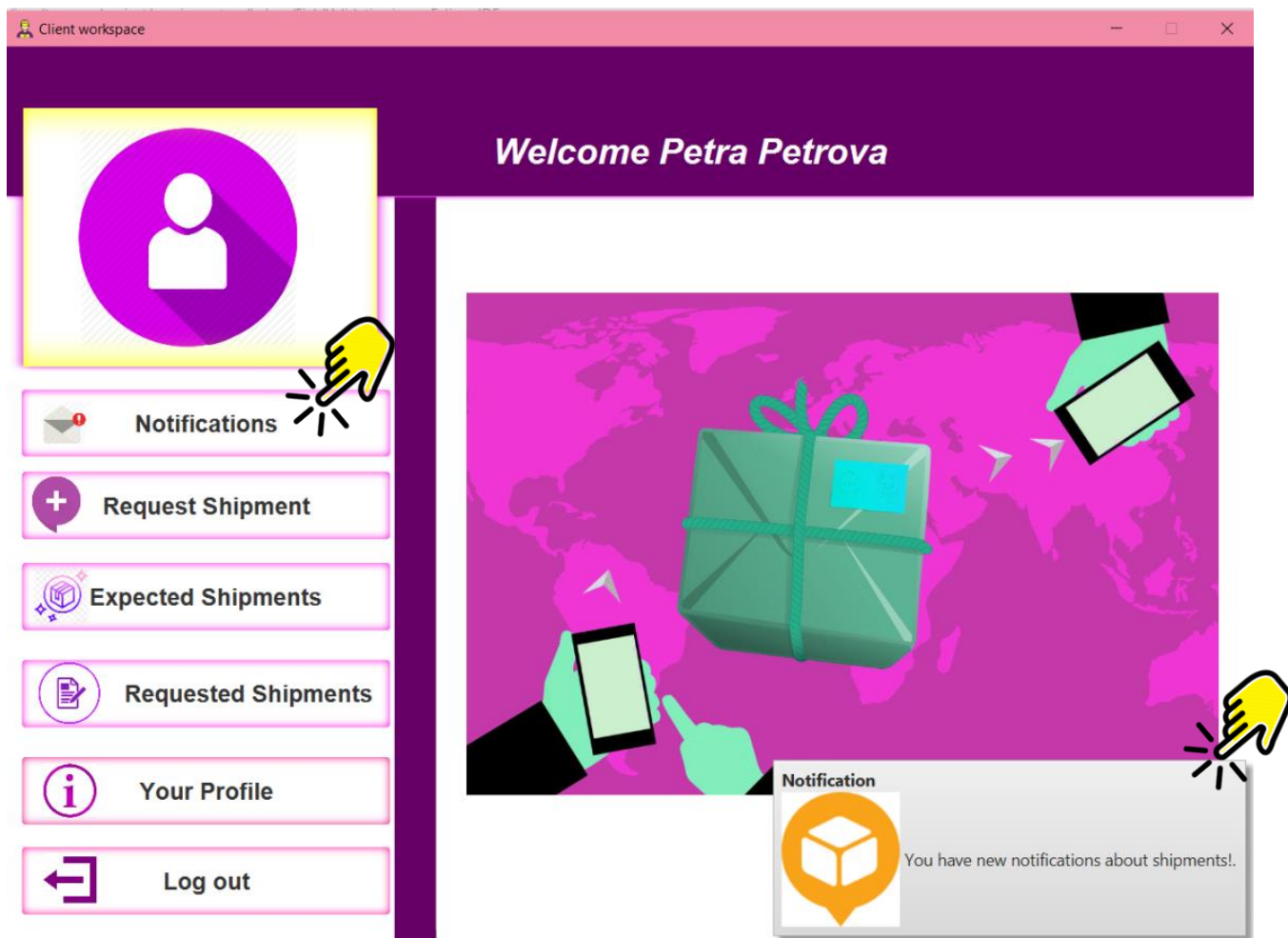
Accept

Shipment №	From	To
20	Bulgaria Burgas Rilska №5	Bulgaria Kotel Georgi Genov #7

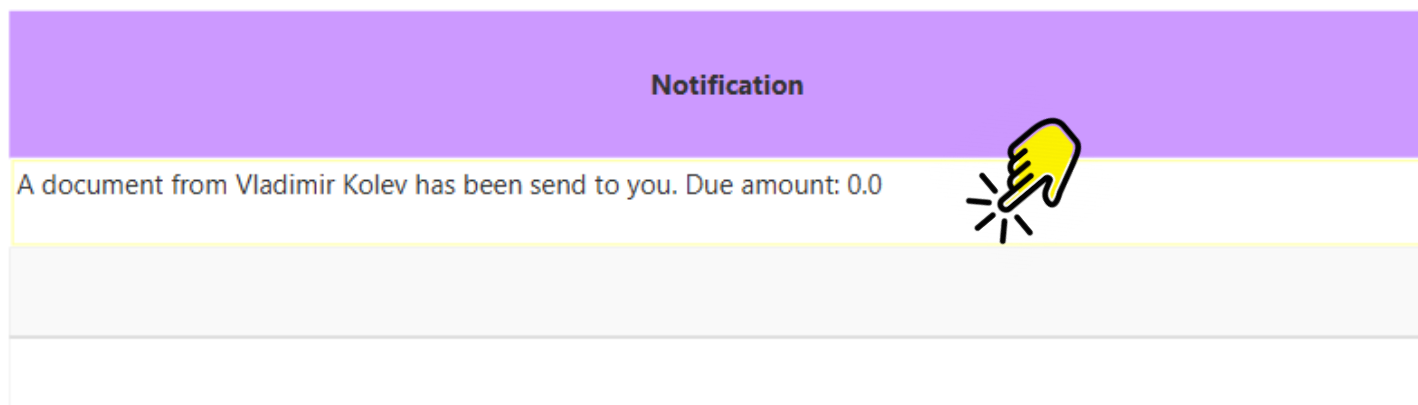
Details



3. Доставка на пратка, известяване и обработка:



b



A document from Vladimir Kolev has been send to you. Due amount: 0.0



Accept

Decline

сл.1

Notification

Your shipment to Petra Petrova was successfullly delivered!

сл.2



Отново в състояние на изчакваща за поемане от куриер – връща се при изпращача

Notification

Petra Petrova declined your shipment! A courier will bring it back to you!

7. ПРОБЛЕМИ

7.1 Създаване на поръчки от администратор:

Тъй като само клас Client има връзка с клас Shipment (receiver и sender) ако обект от клас Admin е зададен за изпращач на пратка, се получава *ClassCastException*.

Затова в базата данни е създаден допълнителен Client- системен клиент, чрез който админ да изпраща пратки:

```
private final String SYSTEM_CLIENT = "1111111111";

public void getCompanyForAdmin(Company choosedCompany) {
    CompanyView company = new CompanyView(choosedCompany.getId(), choosedCompany.getCompanyName());
    companyList.add(company);
    company_name.setText(choosedCompany.getCompanyName());
    companyCombo.setItems(companyList);
    companyCombo.getSelectionModel().select(company);
    this.isCompanyComboDisabled = true;
    this.sender = userService.getUserByPhone(SYSTEM_CLIENT);
}
```

7.2 Изпращане на известие до точния клиент:

Когато пратката се достави до клиент трябва да се покаже известие на екрана, но само ако екрана е на клиента, до когото е стигнала пратката.

Затова, за проверка кой клиент е отворил приложението в момента се използва:

```
private static int id;
```

променлива, която получава id-то на клиента, който логва в системата и която при напускане на системата, от клиента, става равна на 0.


```
@FXML
```

```
private void logOut(ActionEvent event) throws IOException {  
    CloseForm.closeForm(event);  
    Logger.info("Client with id: " + client.getId() + " successfully logged out!");  
    id = 0;  
    OpenNewForm.openNewForm("WelcomeForm.fxml", "Welcome");  
}
```

Ако тази променлива не се преинициализира, е възможно да се покаже известие на грешен клиент.

```
private void sendNotification(String text, boolean isAcceptable) {  
  
    Notification notification = new Notification(this.shipment.getReceiver(), text, isAcceptable, this.shipment);  
    notificationDao.save(notification);  
    notification = new Notification(this.shipment.getSender(),  
        "Your shipment to " + shipment.getReceiver().getName() + " was successfully delivered!", false);  
    notificationDao.save(notification);  
    Platform.runLater(new Runnable() {  
        @Override  
        public void run() {  
            if (ClientWorkspaceFormController.getID() == userDao.get(shipment.getReceiver().getId()).getId()) {  
                notificationService.sendNotification("A shipment was delivered to you right now!",  
                    (Client) userDao.get(shipment.getReceiver().getId()));  
            }  
            if (ClientWorkspaceFormController.getID() == userDao.get(shipment.getSender().getId()).getId()) {  
                notificationService.sendNotification(  
                    "Your shipment to " + shipment.getReceiver().getName() + " was successfully delivered!",  
                    (Client) userDao.get(shipment.getSender().getId()));  
            }  
        }  
    });  
}
```

Функция sendNotification се извиква след доставяне на пратката. Ако статичната променлива id на контролера е равна на тази на получателя или на изпращача на пратката се показва съобщение в неговата работна среда.

Използва се:

```
Platform.runLater(new Runnable() {  
    @Override  
    public void run() {
```

За изпращане на известията в background режим, за да не се получават javaFX runtime errors.

7.3 Неактуална информация:

За да се избегне извеждане на **неактуална информация** при всяко натискане на бутон за извеждане на справка **се прави заявка към базата данни и се извеждат актуалните моментни записи и техния статус.**

8. ВЪЗМОЖНОСТИ ЗА ПОДОБРЕНИЕ

- **Разширяване на алгоритъма за доставка на пратка** при поемане на няколко пратки от един куриер наведнъж, той ще ги достави до всички получатели по почти едно и също време. Това може да се подобри като се добави и автоматично разпределение на пратките по райони и дори по тип пратка;
- **Поддръжка на известяване и за останалите роли в системата** – админ (системни известия, проблеми на потребители..) и куриер(известия за заявена пратка принадлежата на негов работен район..);
- **Добавяне на допълнителни роли на потребител** – агент на офис, собственик на фирма и осигуряване на съответните им функционалности;
- **Login да се отдели в отделен защитен модул.** Да се криптират паролите.
- **Добавяне на различни начини на доставка.** (Стандартна, Експресна доставка с различни цени)
- **Добавяне на различни методи на плащане.**