**Технически университет – Варна  
Факултет: ФИТА  
Катедра: ИКТ  
Специалност: СИТ**

**К У Р С О В**

**П Р О Е К Т**

**Тема: Информационна система – склад**

**Изготвили: Проверил:……….**  
Радослав Колев, фак. №: 18621786 *доц.инж. Хр.Ненов*  
Николай Николаев, фак. №: 18621695

Условие на проекта

Да се разработи информационна система, предоставяща услуга склад. Програмата съхранява и обработва данни за складови помещения. Системата позволява множествен достъп.

Системата поддържа два вида потребители: администратор и оператори (складов агент) с различни роли за достъп до функционалностите в системата.

Операции за работа с потребители:

* Създаване на складови оператори от администратор;
* Създаване на доставчици;
* Създаване на клиенти;
* Създаване на каса (Парична наличност);

Системата поддържа операции за работа със събития.

* Създаване на номенклатури;
* Работа с фактури
  + Приемане на стока от Доставчик на доставена цена;
  + Изписване на стока на продажна цена;
* Наблюдение за наличност на стоки в склада;
* Наблюдение за наличност на пари в касата;

Системата поддържа справки по Произволен период за:

* Доставки и доставчици;
* Изписване и клиенти;
* Дейност на складовите оператори;
* За наличности в склада;
* Разходи, приходи, печалба;
* Движение на наличността в касата;

Системата поддържа Известия за събития:

* Критичен минимум и липса на стока;
* Критичен минимум и липса на парична наличност;

Анализ на проблема

1.1 Въведение

В настоящата курсова работа е разработена информационна система, предоставяща услуга склад, като основните функции на системата са да обработва поръчки идващи от и към склада, като чрез нея се следи за целия процес, от приемането на стоки в склада до тяхното пренасочване към клиентите.

1.2 Функционални изисквания и дефиниция на модулите

Системата поддържа два вида потребители: администратор и складови агенти (оператори), които имат различни роли за достъп до функционалностите на системата. Администраторът има достъп до всички функции в програмата, докато складовият оператор е ограничен спрямо добавянето на нови складови оператори и следене за тяхната дейност.

Системата дава възможност за управлението на складови наличности, проследяване на движението на стоките, тяхната инвентаризация, обработка на клиенти и поръчки и тяхното състояние. Чрез нея се генерират справки и отчети за съответно нейните клиенти, доставчици, складови оператори, разходи, приходи и печалба.

Системата извежда съобщение, при критичен минимум на дадена стока или на пари в касата.

Системата е реализирана на програмен език Java, като е използвана технологията JavaFX на Scene Builder за изграждането на потребителския интерфейс. За хранилище на данните, се използва база от данни на Oracle.

Всички събития в системата се запазват във файл, поради наличието на модул log4J.

Когато премине изпълнението на дефинираните задачи следва тестване (и корекции при откриване на грешки и несъответствия). Проверява се дали коректно функционират нововъведенията в самостоятелен режим на работа. При откриване на грешки, те се отстраняват и системата отново бива подлагана на тест.

1.3 Структура на проекта

 Кратко описание за структурата - често изборът на правилната структура прави програмата по-ефективна, тъй като се спестява памет и време за изпълнение. Данните биват групирани по определен начин, за да се улесни достъпът до тях и управлението им. За различни задачи са подходящи различни структури.

За по-сложни случаи могат да се създадат много по-сложни структури. При избор на подходяща структура е възможно по-бързо (с ползване на минимум ресурси) обработване на информацията

Курсовата работа е има следната структура, която е съставена от 4 части:

**Част първа** „Анализ на проблема” – Кратко описание на системата и нейните функционални изисквания.

**Част втора** „Проектиране на системата” – Разработване на базата от данни, потребителския интерфейс, UML диаграми, ERD диаграма на базата от данни.

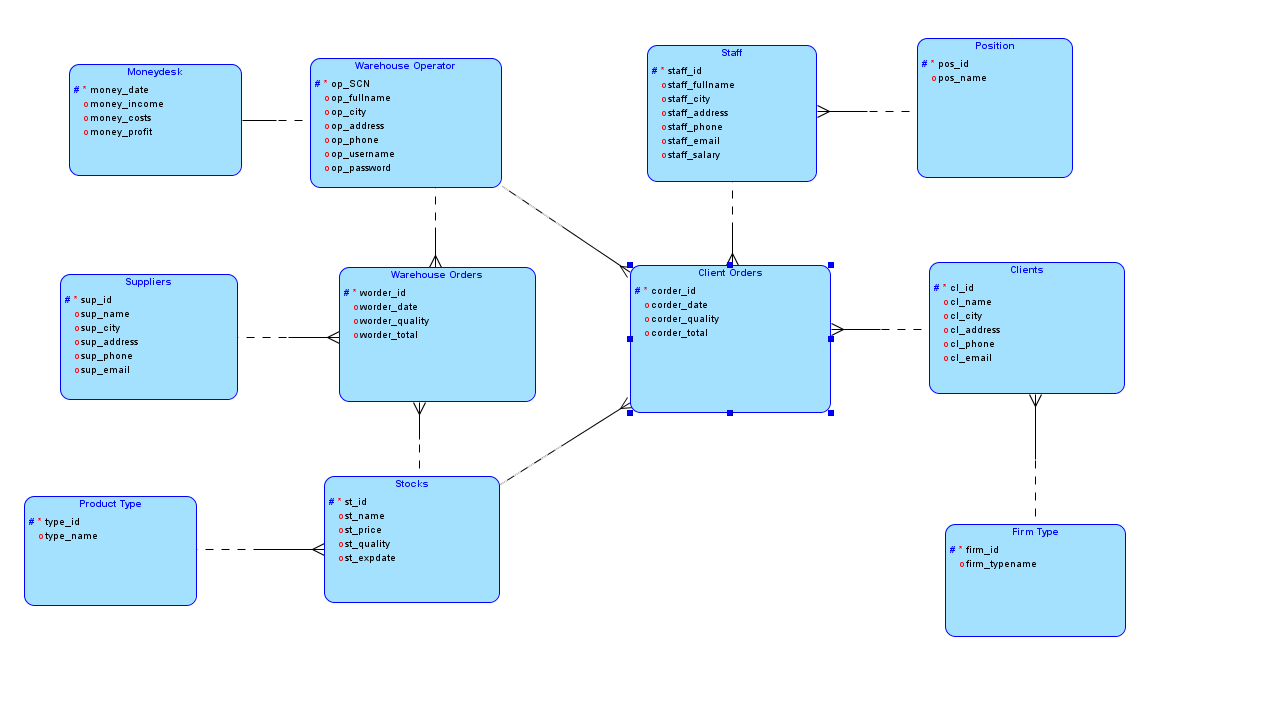
**Част трета** „Реализация на системата” – Реализация на базата от данни с Oracle и описание на таблиците. Реализация на слоя за работа с базата от данни – DAO, на бизнес логиката и графичния интерфейс както и на модула за регистриране на събития в системата -log4J.

**Част четвърта** „Тестови резултати” – Разработване на тестове за проверка на функционалността на системата.

Проектиране на системата.

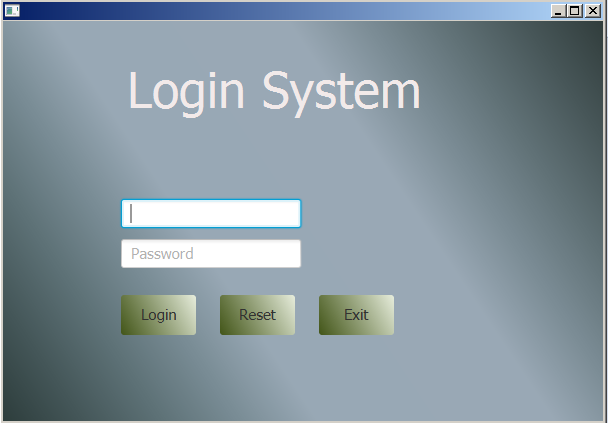
* 1. Проектиране на отделните модули

А) **База данни** - Проектиране на базата данни – Data Modeler, Oracle

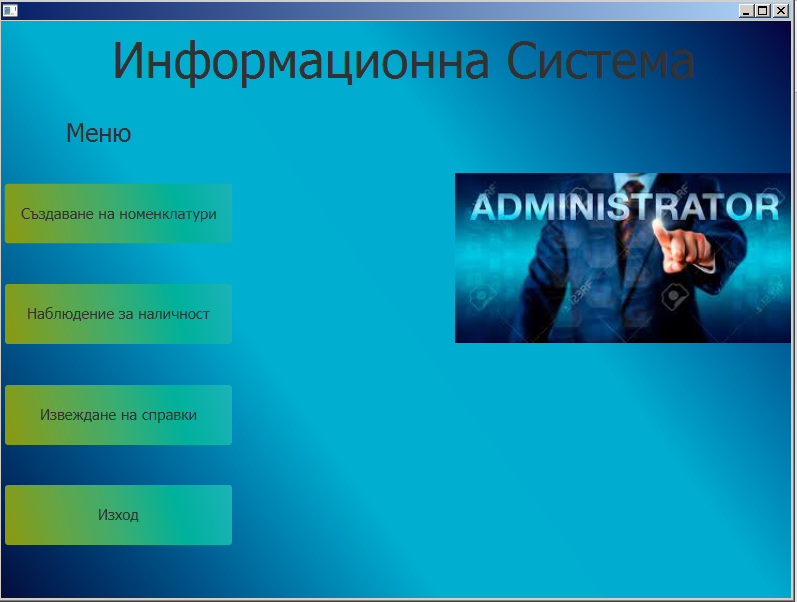
***Логически модел:***

Б) **Проектиране на GUI** – чрез Scene builder, където се добавя всеки контролер и се разработва.

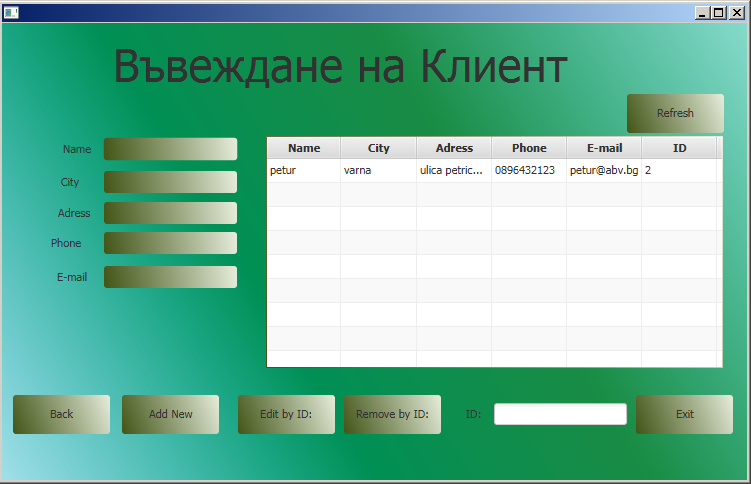
1. **Логин форма** (администратор или оператор)



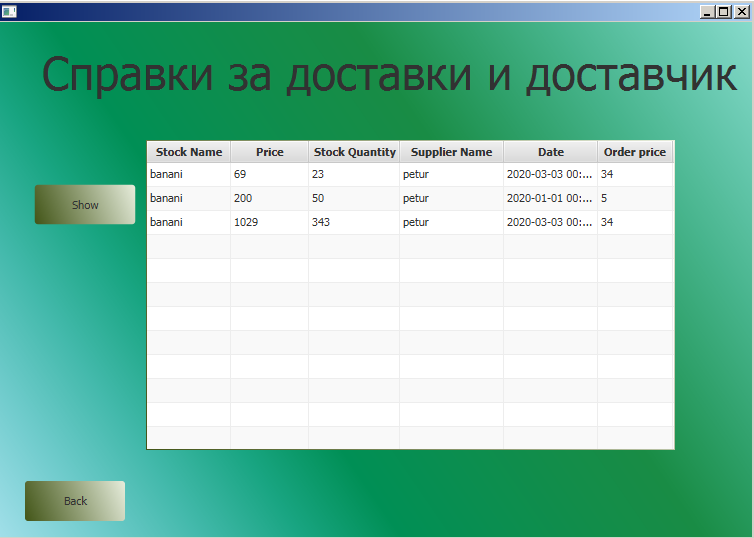
1. **Главна Форма:**



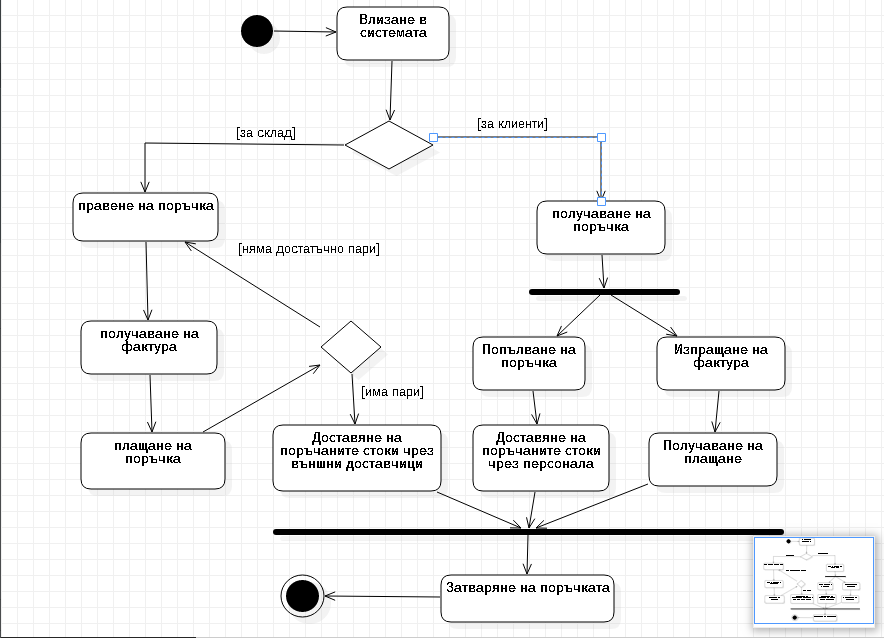
1. **Форма за Въвеждане:**



1. **Форма за справки:**



В) **Проектиране на бизнес логика** – чрез Activity Diagram на StarUML, показавайки основната роля на приложението:



* 1. UML диаграми:

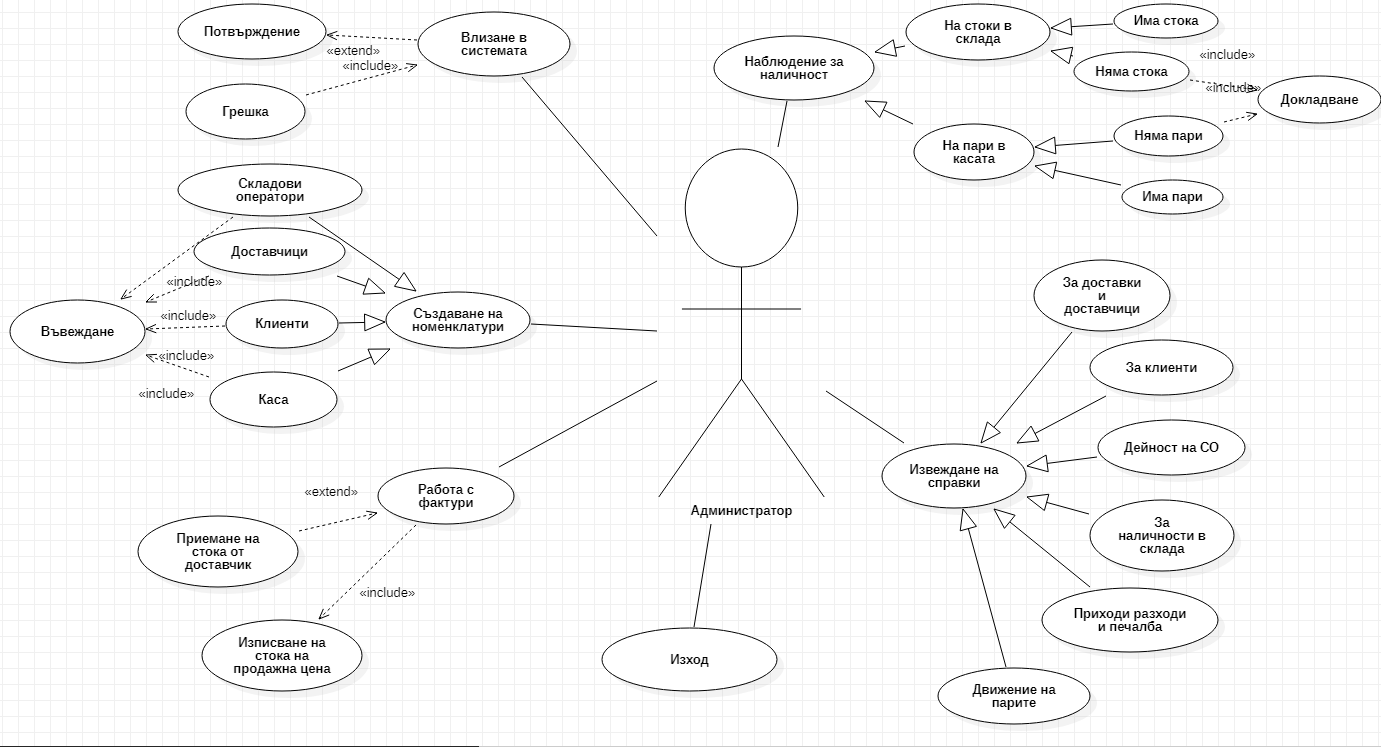
Диаграмите са създадени на **StarUML**.

Основни участници в системата са:

• **Администратор**

**• Оператор**

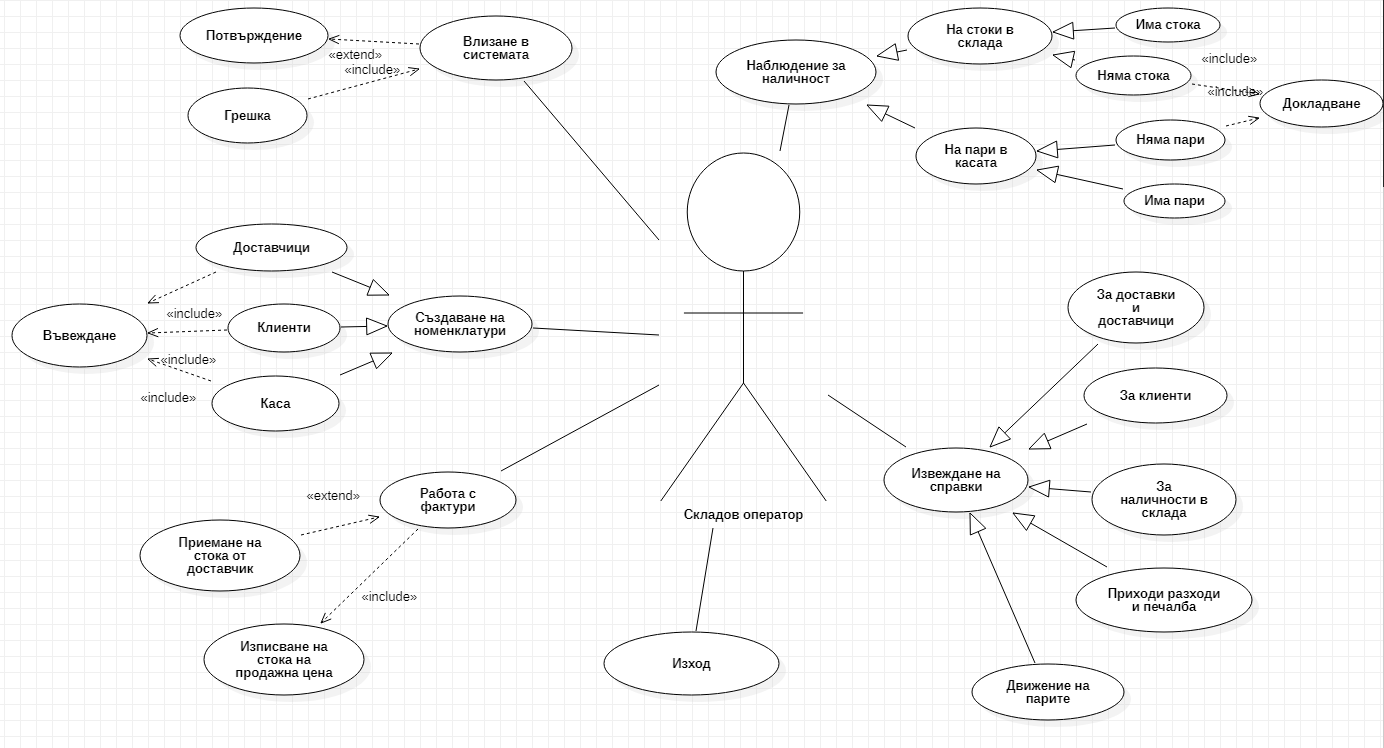
А) **Use case диаграма за администратор:**



**Администраторът** е лице отговарящо за:

1. Всички функции които използва складовият оператор;
2. Въвеждане на складов оператор;
3. Справка за следене дейността на складовия оператор

Б) **Use case диаграма за складов оператор:**

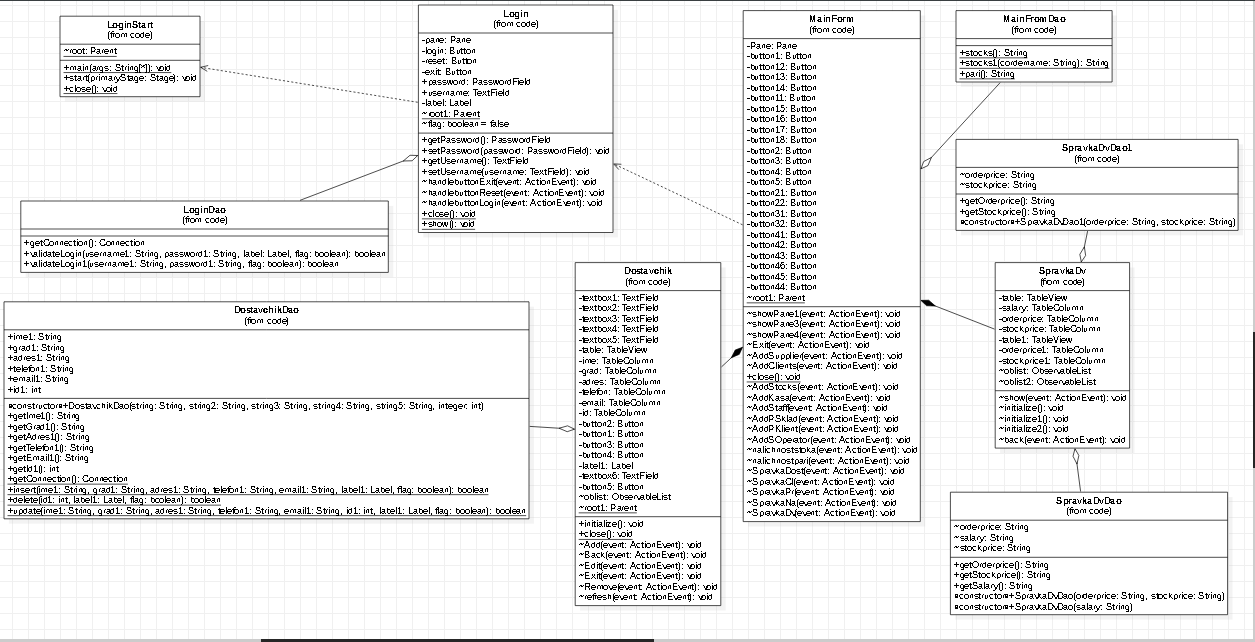


Операторът на склад е лице, което отговаря за операциите в складовете. Той следи за изчерпване на количествата в склада, вписване и изписване на стоки, инвентаризация и отчети.

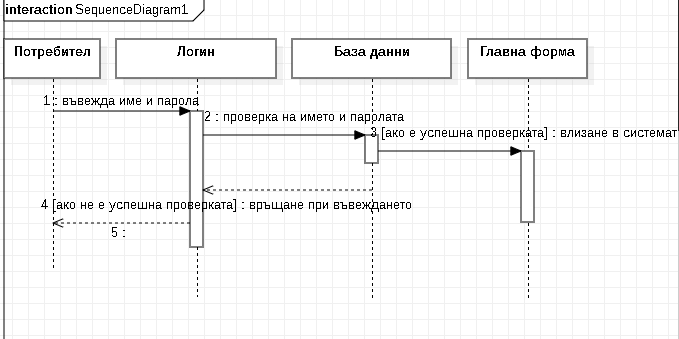
Функциите, които той използва са:

* Въвеждане на доставчици
* Въвеждане на клиенти
* Въвеждане на каса(Парична наличност)
* Създаване на номенклатури
* Приемане на стока от доставчик на доставена цена
* Изписване на стока на продажна цена
* Наблюдение за наличност на стоки в склада
* Наблюдение за наличност на пари в касата
* Изготвяне на справки за Доставки и доставчици
* Изготвяне на справки за Изписване и клиенти
* Изготвяне на справки за наличности в склада
* Разходи, приходи, печалба.
* Движение на наличността в касата.

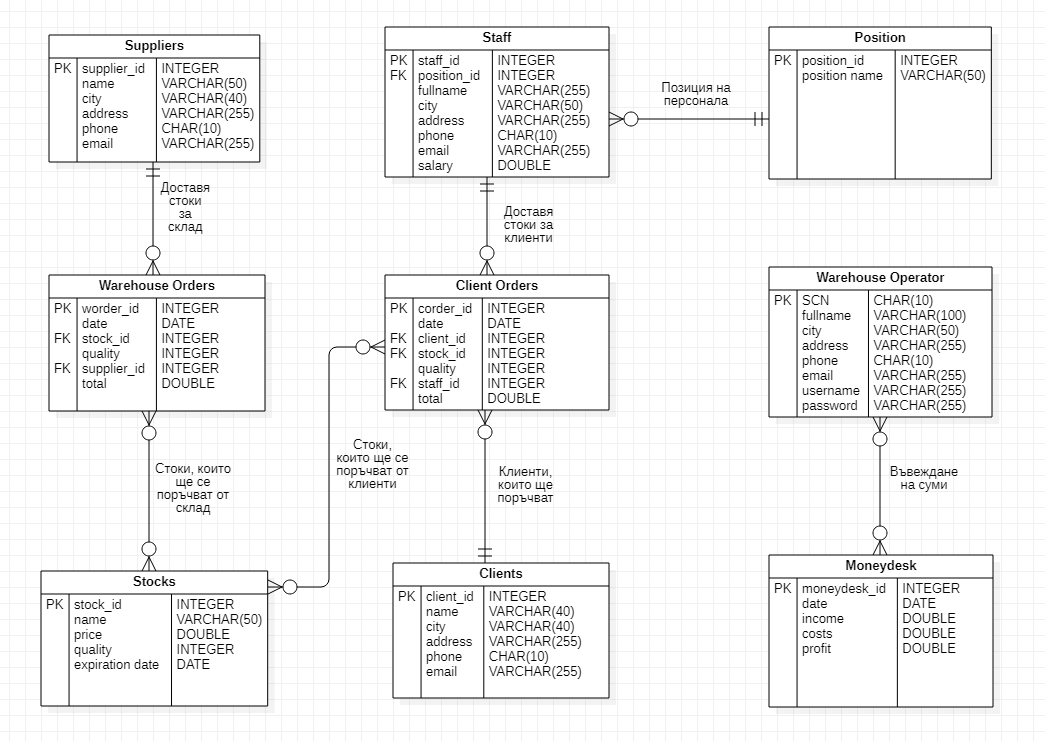
В) **Class диаграма**:



Г) **Sequence диаграма:**



**Д) Концептуален модел на базата от данни(ЕРД)**



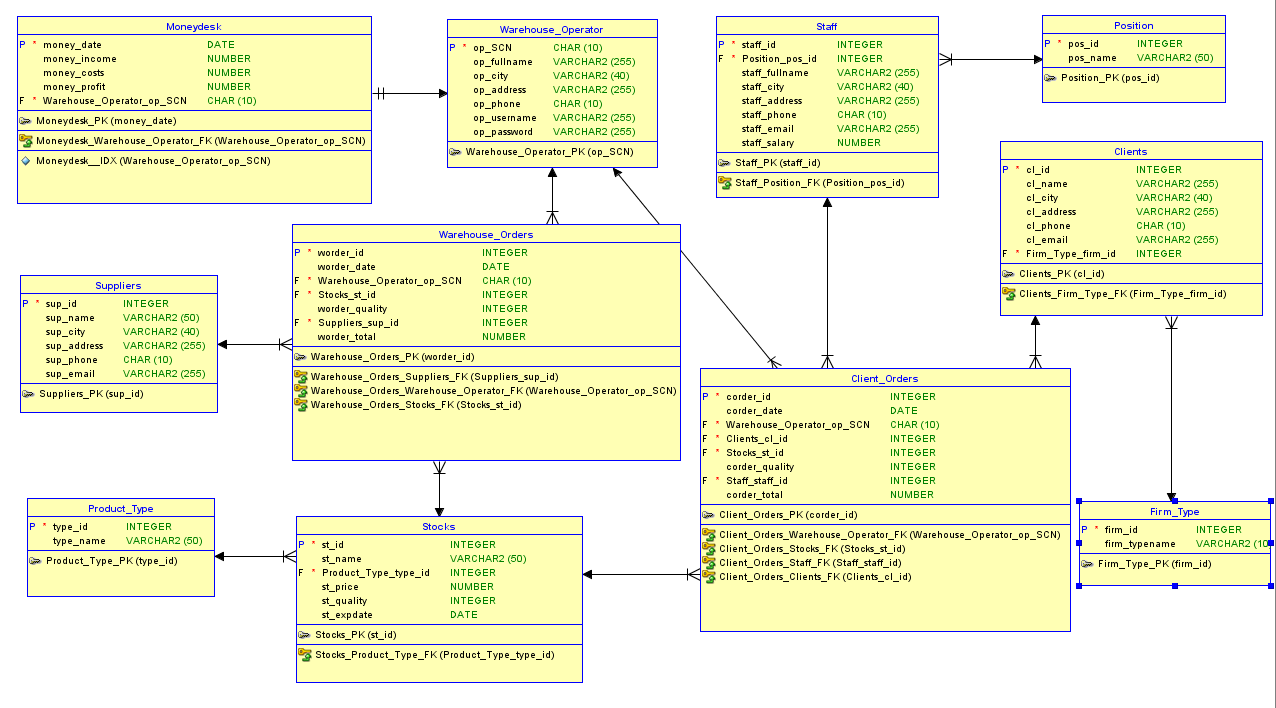
В диаграмата са описани следните таблици:

* ***Доставчици (Suppliers)*** – Това са фирми, които доставят поръчаните от нас стоки;
* ***Стоки (Stocks)*** – съхранява данните за стоките и тяхната наличност в склада;
* ***Клиенти (Clients)*** – съхранява информация за клиентите, които поръчват от склада;
* ***Каса (Moneydesk)*** – Съхранява информация за наличността в касата;
* ***Персонал (Staff)*** – Съхранява информация за служителите/работниците на склада;
* ***Позиция (Position)*** – Съхранява информация за позициите на персонала
* ***Складов оператор (Warehouse operator)*** – Съхранява информация за складовите агенти;
* ***Клиентски поръчки (Client Orders)*** – Съхранява информация за направените поръчки от клиентите;
* ***Складови поръчки (Warehouse Orders)*** – Съхранява информация за направените поръчки от склада.

Реализация на системата.

* 1. Реализация на базата от данни (Oracle)

А) **Релационен модел на базата от данни**



С помощта на **SQL Developer**, създаваме таблиците и въвеждаме данни в тях, които използваме за съхранение на информацията.

Създаване на таблица **Клиенти(Clients)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Cl\_id | Идентификационен номер на клиент |
| Cl\_name | Име на клиент |
| Cl\_city | Град, в който живее клиента |
| Cl\_address | Адрес на клиент |
| Cl\_phone | Телефонен номер на клиент |
| Cl\_email | Електронна поща на клиент |

Създаване на таблица **Позиция(Position)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Pos\_id | Идентификационен номер на позицията |
| Pos\_name | Име на позиция за персонал |

Създаване на таблица **Поръчки за клиенти(Client\_orders),** която съдържа следните данни:

|  |  |
| --- | --- |
| Corder\_id | Идентификационен номер на поръчката |
| Corder\_date | Дата на поръчката |
| Cl\_id | Идентификационен номер на клиент |
| Staff\_id | Идентификационен номер на персонал |
| Corder\_total | Цена за доставката |
| Corder\_name | Име на поръчката |

Създаване на таблица Цена **на Поръчката(Corder\_price)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Stocks\_st\_id | Идентификационен номер на стоката |
| Client\_orders\_corder\_id | Идентификационен номер на поръчката за клиент |
| quantity | количество |
| Price\_each | Единична цена |
| Price\_total | Обща цена(количество \* единична цена) |

Създаване на таблица **Каса(Moneydesk)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Money\_id | Идентификационен номер на касата |
| Money\_income | приходи |
| Money\_costs | разходи |
| Money\_Profit | Печалба(приходи – разходи) |
| Money\_Date | Дата на създаване на касата |

Създаване на таблица **Персонал(Staff),** която съдържа следните данни:

|  |  |
| --- | --- |
| Staff\_id | Идентификационен номер на персонала |
| Staff\_fullname | Име на персонала |
| Staff\_city | Град, в който живее персонала |
| Staff\_address | Адрес на персонала |
| Staff\_phone | Телефонен номер на персонала |
| Staff\_email | Електронна поща на персонала |
| Staff\_salary | Заплата на персонала |
| Pos\_id | Идентификационен номер на позиция на персонала |

Създаване на таблица **Стоки(Stocks)(Наличност на стоки)**, която съдържа следните данни:

|  |  |
| --- | --- |
| St\_id | Идентификационен номер на стоката |
| St\_name | Име на стоката |
| St\_expdate | Дата на изчерпване на стоката |
| St\_quantity | Налично количество от дадената стока |

Създаване на таблица **Доставчици(Suppliers)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Sup\_id | Идентификационен номер на доставчика |
| Sup\_name | Име на доставчика |
| Sup\_city | Град, в който живее доставчика |
| Sup\_address | Адрес на доставчика |
| Sup\_phone | Телефонен номер на доставчика |
| Sup\_email | Електронна поща на доставчика |

Създаване на таблица **Потребител(User1)**, която съдържа следните данни:

|  |  |
| --- | --- |
| username | Име на акаунт в системата |
| password | Парола на акаунт в системата |
| Op\_id | Идентификационен номер на потребител, който ще влиза с името и паролата в системата |

Създаване на таблица **Потребител(Оператор или Администратор)(Warehouse\_operator)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Op\_id | Идентификационен номер на потребител |
| Op\_fullname | Име на потребител |
| Op\_city | Град, в който живее потребителя |
| Op\_address | Адрес на потребителя |
| Op\_phone | Телефонен номер на потребител |
| Op\_email | Електронна поща на потребител |

Създаване на таблица **Поръчки за склад(Warehouse\_orders)**, която съдържа следните данни:

|  |  |
| --- | --- |
| worder\_id | Идентификационен номер на поръчка за скалд |
| Worder\_date | Дата на поръчката |
| Sup\_id | Идентификационен номер на доставчик, който е донесъл поръчката |
| Worder\_total | Цена за доставка на поръчката |
| Worder\_name | Име на поръчката |

Създаване на таблица **Цена на поръчката за склад(Warehouse\_price)**, която съдържа следните данни:

|  |  |
| --- | --- |
| Stocks\_st\_id | Идентификационен номер на стоката |
| Warehouse\_orders\_worder\_id | Идентификационен номер на поръчката за склад |
| W\_quantity | Количество от стоката |
| W\_price\_each | Единична цена |
| W\_price | Обща цена(количество \* единична цена) |

След създаването на таблици вече можем да въвеждаме данни в тях и да им зададем съответните стойности и имена. Също е много важно да се спазва размера на всяка колона от таблицата, защото въвеждане на повече символи от допустимите е невъзможно и ще доведе до неточности и неуспешни въвеждания на данни и тяхното променяне и изтриване от системата.

* 1. Реализация на слоя с базата данни (DAO)

**public** Connection getConnection() {

Connection com = **null**;

**try** {

Class.*forName*("oracle.jdbc.driver.OracleDriver");

com = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","JAVA","JAVA");

}

**catch** (Exception e)

{

System.***out***.println(e);

}

**return** com;

}

Следната функция прави връзка към сървъра на базата данни, чрез **JDBC driver** като в конекцията (връзката) се задават **hostname** – името на сървъра/хоста, **port number** – номер на порт към който сочи хоста. Съответно ако връзката има име и парола и те се въвеждат, за да може да се свърже успешно.Също така и **listener (xe)**, който е много важен, защото ако е различен от този, който е създаден в Oracle свързването ще бъде неуспешно. Това е един вид свързване с базата данни и когато това се осъществи, могат да се използват ресурсите и данните от там.

DАО позволява на системата да се свърже и да се използват различни функции и данни от базата данни, които могат да се реализират в проекта.

Следният код ще покаже един пример за свързване и проиграване на таблиците от базата данни чрез data access object (DAO)

**public** **boolean** validateLogin(String username1,String password1,Label label,**boolean** flag) {

**try**{

Connection com= getConnection();

Statement st=com.createStatement();

ResultSet rs=st.executeQuery("SELECT \*FROM user1 WHERE username='"+username1+"' AND password='"+password1+"'");

**if** (rs.next())

{

label.setText("Logged in!");

flag=**true**;

}

**else** {

Alert a = **new** Alert(AlertType.***INFORMATION***);

a.setContentText("Wrong username or password!");

a.show();

flag=**false**;

}

}

**catch** (Exception e)

{

System.***out***.println(e);

}

**return** flag;

}

Чрез тази функция описана по-горе се вижда как се използва най-пряко DAO(data access object), чрез свързване към сървъра на базата данни и използване на заявка за търсене на **username и password от таблица User1**, за да може да се потвърди дали името и паролата съществуват в базата данни и да върнат съответния резултат от изпълнението.

3.3 Реализация на бизнеслогикa и графичен интерфейс – Java, JavaFX

А)Интерфейс Java, JavaFX

Интерфейсът който използваме е разновидност на потребителски интерфейс, в който елементите, предоставени на потребителя за управление, са изпълнени във вид на графични изображения (менюта, бутони, списъци и др.).

Софтуерният интерфейс предлага достъп до ресурсите на компютъра – памет, микропроцесор и т.н. Позволява директен достъп до такива ресурси, където софтуерът може да има много разклонения – понякога катастрофални – за функционалността и стабилността.

Потребителският слой е представен от **Java, JavaFX форми**, чрез които се манипулират данните в базата.

Използван е **JavaFX Scene Builder** – инструмент за визуално оформление, който позволява на потребителите бързо да проектират потребителски интерфейси на приложението JavaFX, без кодиране. Потребителите могат да плъзгат и пускат компоненти на потребителския интерфейс в работна област, да променят свойствата си, да прилагат таблици със стилове и FXML кодът за оформлението, което създават, се генерира автоматично във фонов режим. Резултатът е **FXML файл**, който след това може да се комбинира с проект на Java чрез обвързване на потребителския интерфейс с логиката на приложението.

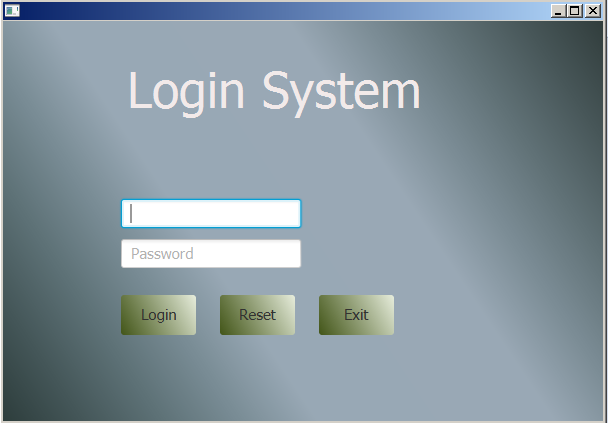
Инструментът улеснява работата с графичен интерфейс и ни дава възможност за по-добра реализация на проекта. Съдържа автоматично генериране на код и задаване на имена на контролери на класове, пакети, бутони и други компоненти, които се разработват в контролер класа в по-късен етап.

Използвани са библиотеки на Java и JavaFX, за да могат да се вкарат всички компоненти от FXML документа в кода на Java проекта и така се стартират успешно, без никакви грешки и проблеми.

Първо е създадена формата за въвеждане, която включва следните компоненти и техните обяснения:

* **Label** – това е компонент, в който могат да се пишат различни текстове и думи;
* **Username field (text field)** – това е компонент, в който могат да се пишат текстови полета като име и други;
* **Password field (text field)** – това е компонент, в който могат да се пишат текстови полета като парола и други;
* **Button** – това е компонент от тип бутон за извършване на действия – при активирането му могат да се зададат различни функционалности на избраните детайли;
* **Pane** – това е компонент от тип прозорец, в който могат да се слагат отделни компоненти.

Графично изобразяване на формата за въвеждане.

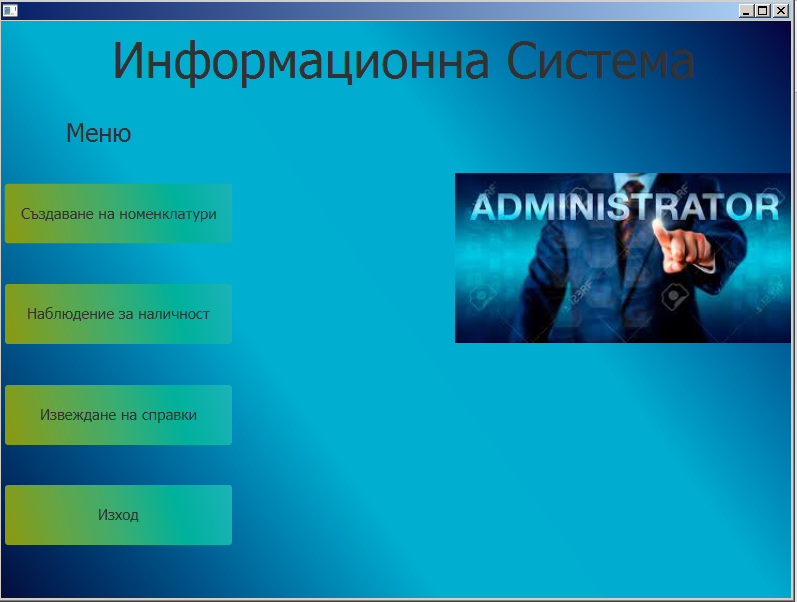


След формата за въвеждане се активира главната форма, която включва следните компоненти и тяхните обяснения:

* **Label** – това е компонент, в който могат да се пишат различни текстове и думи;
* **Button** – това е компонент от тип бутон - за извършване на действия – при активирането му могат да се зададат различни функционалности на избраните детайли;
* **Pane** – това е компонент от тип прозорец, в който могат да се слагат отделни компоненти;
* **ImageView** – илюстрация на снимка, която се зарежда чрез избор на местоположението на файла .

Тук се проиграват бутоните с различни функции за видимост, като се показват различни панели и бутони и се скриват в зависимост от изпълнението на дадения потребител и неговите решения.

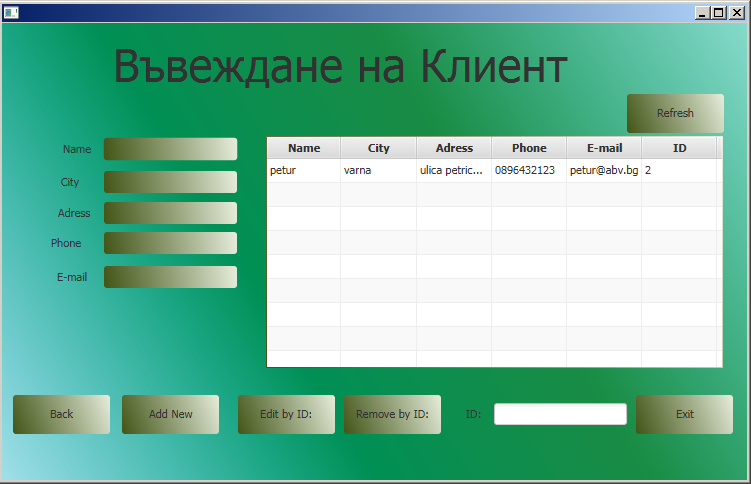
Графично изображение на главната форма:



След главната форма, в зависимост от бутоните, се зареждат различни други форми. Първо ще илюстрираме форма за въвеждане на клиент, която включва следните компоненти и техните обяснения:

* **Label** – това е компонент, в който могат да се пишат различни текстове и думи;
* **Text field** – това е компонент, в който могат да се пишат текстови полета като име и други;
* **Button** – това е компонент от тип бутон за извършване на действия – при активирането му могат да се зададат различни функционалности на избраните детайли;
* **Pane** – това е компонент от тип прозорец, в който могат да се слагат отделни компоненти;
* **TableView** – това е илюстрация на таблица, която има различни колони, в които се записва информация, съответно различни данни за всяка колона;

Форма за въвеждане на клиент:

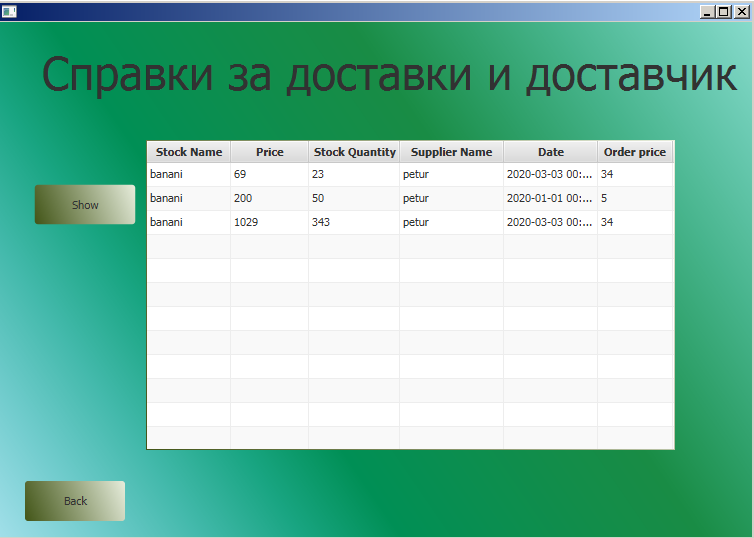


Всички форми за въвеждане са създадени на един и същ принцип, просто имат различен брой компоненти и данни.

После ще илюстрираме форма за справки за доставки и доставчици, която включва следните компоненти и техните обяснения:

* **Label** – това е компонент, в който могат да се пишат различни текстове и думи;
* **Button** – това е компонент от тип бутон за извършване на действия – при активирането му могат да се зададат различни функционалности на избраните детайли;
* **Pane** – това е компонент от тип прозорец, в който могат да се слагат отделни компоненти;
* **TableView** – това е илюстрация на таблица, която има различни колони, в които се записва информация, съответно различни данни за всяка колона;

Форма за справки за доставки и доставчици:



Всички форми за справки са създадени на един и същ принцип, просто имат различен брой компоненти и данни.

Б) Бизнес логика

Бизнес логиката е описание на функционалността на системата и действията, които се развиват.

Примерна бизнес логика включваща приложението ни за услуга – склад.

Първо се стартира Login формата, като използваме класа Login Start, чрез който зареждаме формата. След това имаме контролер Login и LoginDAO клас за влизане в системата.Чрез контролера задаваме бутоните, текстовите полета за въвеждане на данни, заглавие и други оформления, а за да влезем в системата използваме класа LoginDAO. В него първо сме разработили функция, която проверява дали даден потребител и парола съществуват в базата данни и съответно, ако съществуват – влизането в системата е успешно, а в противен случай излиза съобщение за грешна парола или потребителско име.

В системата могат да влязат 2 типа потребители, с различни права на достъп:

* + - **Администратор**
    - **Складов оператор (агент)**

При влизане в системата като **складов оператор** се зарежда главната форма за навигация към другите форми. В нея съответно всеки складов оператор може да въвежда различни данни както следва:

* + - * **въвеждане на доставчици** – фирми, които доставят стоки за склада;
      * **въвеждане на клиенти** – всички клиенти, които поръчват стоки от склада;
      * **въвеждане на каса (Парична наличност)** – обхваща паричната наличност на склада във вид на приход, разход и печалба;
      * **въвеждане на стоки (наличност)** – стоките, които са налични в склада и тяхната бройка
      * **въвеждане на персонал** – служители, които доставят направените поръчки към клиентите;
      * **въвеждане на поръчки за склада** - включва всички поръчки, които са поръчани за склада и лицата, които отговарят за тях;
      * **въвеждане на поръчки за клиенти** – включва всички поръчки, които се доставят на фирмите и лицата, отговарящи за тях.

При избор на едно от тези въвеждания излиза отделна форма за всяко едно от тях, в която се записват съответните данни и се съхраняват в базата данни на системата. При въвеждане на поръчки за и към склада, се променят данните в таблици Наличност и Каса. Добавени са и опции за промяна или изтриване на запис от таблиците при необходимост.

Освен тези действия, потребителят може да наблюдава за наличност на стоки в склада или на пари в касата. Така винаги, когато някой е извършил транзакция на стоки или пари, те се отразяват в наличност и е улеснение за потребителите, които искат да направят бърза справка.

Други функции, които може да използва складовият оператор са справки, чрез които следи различни потреби в склада. Използвани са няколко вида справки:

* + - **за доставки на стоки и доставчици** – дава пълна информация за доставките, които са доставени и доставчиците, които са ги доставили в склада;
    - **за клиенти и изписвания на стоки** – дава пълна информация за доставките, които са изписани от склада и клиентите, на които са доставени;
    - **за приходи, разходи и печалба** – дава пълна информация за паричните средствата в касата;
    - **за наличност в склада** – дава пълна информация за наличността на стоки в склада;
    - **движение на парите** – дава пълна информация за парите, които излизат от касата и които влизат в касата, като приход, разход и печалба.

Чрез заявките се следи за неточности и нередности в склада както и за паричната наличност. Показват ясна информация за случващото се в предприятието.

При влизане в системата като **администратор** се зарежда друга главна форма, подобна на тази на складовия агент, но с тази разлика, че има достъп до някои допълнителни функции, които са недостъпни за оператора. На практика, администраторът може да използва всички функции на оператора, като има и някои допълнителни екстри, които са:

* + - **въвеждане на складов оператор** – само този потребител може да създава и въвежда други складови оператори;
    - **Справка за следене дейността на складовите оператори** – дава пълна информация за направените действия от складовите оператори.

Системата поддържа 2 известия:

* + - **За критичен минимум и липса на стока** – когато стоката е по-малко от 1000 бройки, тогава излиза съобщение, което ни подканва да поръчаме от дадената стока, понеже е на изчерпване;
    - **За критичен минимум и липса на парична наличност** – когато правим поръчка, но парите в касата са по-малко от 1000 лева, тогава излиза съобщение, което ни пита дали да продължим с поръчката или се отказваме, понеже не ни достигат средства.

Навсякъде, където данните са некоректни и не съвпадат с данните, въведени в базата данни, излизат информационни или предупредителни съобщения, в зависимост от това колко е сериозен проблема.

Когато един от двата потребители свърши работата си в системата може да излезе от системата от **,,Изход”**.Така ще се прекратят всички дейности, които е предприемал и ще се затвори приложението.

* 1. Реализация на модул за регистриране на събития в системата – log4j.

**Java log4j** (логваща рамка) е комютърен пакет данни, отнасящи се за „логването“ в Java платформата.

Логването служи за следене на активността. То е основен проблем при екипите за разработване. Няколко рамки улесняват и стандартизират процеса на логване в Java платформата.

Логването е разделено на три основни части: Логърът, Форматър и Прибавачът. Логърът е отговорен за запазване на съобщенията и да бъдат вкарани с точно определена информация и изпратени към логващата рамка. След като получи съобщението, рамката извиква Форматърът със съобщението. Форматърът го форматира в приличен вид, в който ще го вижда потребителят. След това рамката занася съобщението до съответния Прибавач за визуализация. Това може да включва показване в конзолата, записване върху диск, добавяне в някаква база данни или емайл.

Пример за logger class:

**public** **class** LoggerClass {

**public** **static** **void** Logger() {

Logger logger=LogManager.*getLogger*(LoggerClass.**class**);

System.***out***.println("The Operator logged in");

logger.trace("This is trace message");

logger.info("This is information message");

logger.error("This is error message");

logger.warn("This is warn message");

logger.fatal("This is fatal message\n\n");

System.***out***.println("Completed");

}

}

Системата log4j включва няколко файла:

* **log4j.properties**;
* **pom.xml** – в него се слага dependency за log4j и logger class;
* **.log** – файл, в който се добавя информацията за всяко логване.

Log4j използва **PropertyConfigurator**, за да зареди **log4j.properties**. Ние дефинираме собствената си реализация на конфигуратора, която се простира от PropertyConfigurator и написваме код, за да сме сигурни че местоположението на файла на приложението е валидно.

Тестови резултати

* 1. JUnit tests - Компонентното тестване (JUnit test) е процес в програмирането, чрез който се тестват отделни единици (компоненти) от сорс код – един или повече програмни модула и техните контролни данни, процедури за използване, оперативни процедури с цел да се установи дали работят правилно. Под *компонент* се разбира най-малката част на едно програмно приложение, която може да бъде самостоятелно тествана. В т.нар. процедурно програмиране компонент би могъл да бъде целия програмен модул, но в по-честите случаи е отделна функция или процедура. В случая на обектно ориентирано програмиране компонент е най-често цял клас, но в някои случаи може и да е отделен метод. Компонентните тестове се създават от програмист или в някои случаи от white box тестери по време на процеса на разработка, за да се гарантира правилното поведение на софтуера и дали той покрива всички първоначално поставени изисквания.

Пример за Junit test:

**public** **class** TestFromPari {

@Test

**public** **void** test() {

MainFromDao test = **new** MainFromDao();

String output=test.*pari*();

*assertEquals*("-205",output);

}

}

Целта на компонентното тестване е да се изолира всяка част от програмата и да се покаже, че отделните части работят правилно. Компонентният тест предоставя строго определени правила, които частите от кода трябва да удовлетворяват. От това произтичат серия предимства:

* Ранно откриване на проблема
* Улесняване на промените
* Опресняване
* Добре завършена документация и дизайн.

#### 4.2 Функционални тестове (Затворена кутия (Black-Box))

[https://upload.wikimedia.org/wikipedia/commons/thumb/f/f6/Blackbox.svg/200px-Blackbox.svg.png](https://bg.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Blackbox.svg)

Black box diagram

Тестването тип затворена/черна кутия разглежда софтуера като „черна кутия“, изучавайки функционалността без каквито и да са познания за вътрешното устройство. Човекът, който извършва теста, е само наясно какво би трябвало дадения софтуер да върши, но не и как го прави. Тестването тип черна кутия включва: еквивалентно разпределение (equivalence partitioning), анализ на граничните стойности (boundary value analysis), таблици за промяна на състоянието (state transition tables), тестване на таблиците за взимане на решение (decision table testing), стрес тестване (fuzz testing), тестване базирано на модел (model-based testing), тестване за определени ценарии (use case testing), изследователско тестване (exploratory testing) и тестване основаващо се на спецификацията (specification-based testing).

Функционалните тестове проверяват дали стойността, която се въвежда е правилна и дали на изхода се принтира правилния резултат. Ако е правилен, тогава тестът е успешен. Не се разглежда целия код, а само там, където приложението извършва реализацията на процес.

### 4.3 Тестове на ниво интеграция (Integration testing)

Тестовете на ниво интеграция имат за цел да проверят взаимовръзките и съвместната работа на отделните компоненти, когато са поставени в една среда. Компонентите може да се интегрират на вълни или накуп (известно още като Големият Взрив – „big bang“). Счита се, че варианта на вълни е по-добър, защото позволява новопоявили се дефекти да се открият и отстранят по-бързо преди да се интегрира следващата вълна и дефектите са се мултиплицират.

Тестовете на ниво интеграция изваждат на преден план проблеми при свързването и съвместната работа на интегрираните компоненти (units, модули). Прогресивно нарастващи групи елементи се тестват спрямо архитектурния план на програмата и се сливат, докато целия софтуер не заработи като единна система.

Накратко интеграционните тестове се различават по това, че те служат за проверка на отделните компоненти на системата, дали са свързани правилно помежду си, било то базата данни с интерфейса или бизнес логиката на приложението и други разделени модули, които имат връзка един с друг.

Описание на пример:

Да речем ако страницата на логване при валидни данни не иска да зарежда главната страница на админа. Тогава трябва чрез интеграционен тест да тестваме първо логването и ако има грешки, да ги коригираме и после да тестваме главната страница. След това да проверим дали връзката между тях е успешна и ако е и всичко работи правилно, тогава интеграционният тест е успешен и модулите са активни.

Ако два модула са свързани последователно и първият от тях е неуспешен, то тогава и останалите модули след него ще са неуспешни. Така интеграционният тест ни позволява улесняване на проблема и намиране на бързото му решение.

Интеграционните тестове ни позволяват улесняване на принципи, като например:

* Фокус върху архитектурата на приложението;
* Идентифициране на всеки модул;
* Проверка как се предават данните;
* Разбиране на потребителските изисквания;
* Условия за многократно тестване;
* Фокусиране върху един тест за определен период.