Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерного проектирования

Дисциплина «Программирование сетевых приложений»

|  |
| --- |
| *К защите допустить*: |
| Руководитель курсовой работы  ассистент |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Богданова Е.А.  \_\_\_.\_\_\_. 20\_\_\_ |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

**СИСТЕМА УЧЕТА ПРОДАЖ КОМПЬЮТЕРНОЙ ТЕХНИКИ**

БГУИР КР 1-40 05 01-10 №\_28

|  |  |  |
| --- | --- | --- |
| Студент | (подпись студента) | Макаренко А.Д. |
|  |  | Курсовая работа представлена на проверку \_\_\_.\_\_\_. 20\_\_\_ |
|  |  | (подпись студента) |

Минск 2018

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ](https://docs.google.com/document/d/1WBqXxUhVZz7dtOZvxzYtb6YJs37XjKYfd0L5g0CLkMw/edit#heading=h.gjdgxs) 2

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 3

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ 5

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0 6

4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ 8

5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ КРАТКОЕ ОПИСАНИЕ 9

5.1 Диаграмма вариантов использования 9

5.2 Диаграмма последовательности 10

5.3 Диаграмма классов 11

5.4 Диаграмма компонентов 13

5.5 Диаграмма развертывания 14

5.6 Диаграмма состояний 15

6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ 16

[2.1 Сервер](https://docs.google.com/document/d/1WBqXxUhVZz7dtOZvxzYtb6YJs37XjKYfd0L5g0CLkMw/edit#heading=h.4d34og8) 16

[2.2 Клиент](https://docs.google.com/document/d/1WBqXxUhVZz7dtOZvxzYtb6YJs37XjKYfd0L5g0CLkMw/edit#heading=h.2s8eyo1) 16

[ЗАКЛЮЧЕНИЕ](https://docs.google.com/document/d/1WBqXxUhVZz7dtOZvxzYtb6YJs37XjKYfd0L5g0CLkMw/edit#heading=h.49x2ik5) 20

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](https://docs.google.com/document/d/1WBqXxUhVZz7dtOZvxzYtb6YJs37XjKYfd0L5g0CLkMw/edit#heading=h.147n2zr) 21

ПРИЛОЖЕНИЕ22

**ВВЕДЕНИЕ**

Автоматизация - известное слово в 21 веке. Век технологий. Компьютер помогает человеку, а иногда и полностью заменяет его. Идея данного проекта, как раз состоит в этом. Автоматизировать систему продаж компьютерной техники и сделать ее удобной для пользования. Что позволит сэкономить и время и деньги.

**1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

В жизни, как и на просторах интернета существую магазины и интернет магазины, которые продают товары. У каждого есть статистика продаж, информация и доходах и т.д. Также существуют целые сети таких магазинов. И чтобы как-то совместить информацию и представить общий отчет, придется неплохо потрудиться в бухгалтерии, провести много расчетов и потратить кучу времени. То есть для этого нужен сотрудник или сотрудники. И чем больше такая сеть магазинов, тем больше сотрудников.

Идея данной системы, выкинуть всех сотрудников и заставить работать технологии на тебя. Зачем платить больше?

Система будет представлять и себя общий сервер, который будет обрабатывать информацию. Также будет сервер будет взаимодействовать с одной общей базой данных. И вся информация со всех магазинов будет храниться в одном месте. Как листы бумаги можно взять и просмотреть каждую. Но для этого уже не нужно собирать информацию о всех. Она уже собрана!

Так же система будет иметь пользовательский интерфейс, для того чтобы сотрудники магазина, смогли вносить информацию о товарах. Или же если это интернет магазин, то сами покупатели будут вносить эту информацию, а система будет контролировать и распределять ее в нужные места.

Необходимые методы сортировки, вывода статистики, рисование графиков, будет выполнять путем нажатия одной кнопки и все это автономно и удобно.

Моя система - это система учет продаж именно компьютерной техники. Компьютеры можно продавать целыми или частями. Также некоторые магазины предоставляют возможность покупателю выбрать необходимое железо(компьютерные составляющее: видеокарта, процессор и т.д.). А что если и это система сделает сама? Почему бы не дать пользователю сделать свой компьютер самому соединяя устройства, которыми он бы его хотел оснастить или просто мог себе позволить? А это уже экономия времени продавца. В случае интернет магазина, то он там и вообще не нужен.

Теперь обобщим вышеуказанную информацию:

Цель: написать систему которая объединит информацию и данные по каждому магазину и будет хранить ее в месте удобном для использования в нуждах администрации.

Какие проблемы решает:

* данные хранятся в одном месте их легко обрабатывать, сортировать, делать необходимые операции и подсчеты над ними.
* проста в использовании как для клиента, так и владельца, а значит трата времени на покупку нужно компьютера станет меньше.
* наличие лишних сотрудников.

**2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ**

Для создания системы сервер-клиент будет использован UDP - протокол.

UDP (англ. User Datagram Protocol — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

В качестве языка программирования использовался ООП язык JAVA.

Java — сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Используемая база данных: MySQL.

MySQL — свободная реляционная система управления базами данных.

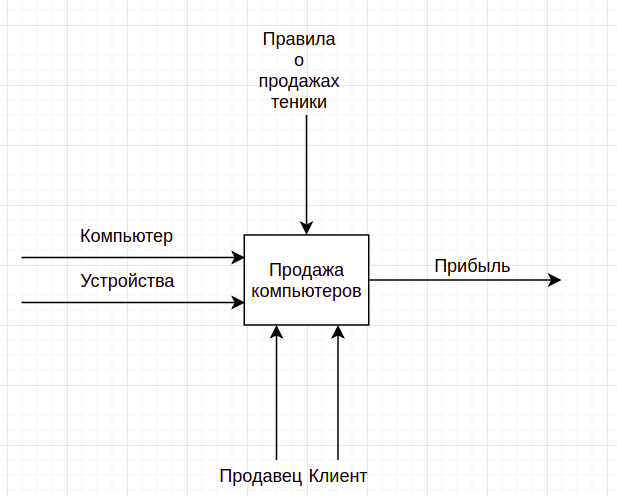
Для реализации GUI использовались библиотеки SWING.

Swing — библиотека для создания графического интерфейса для программ на языке Java.

Для написания был использован редактор Visual Studio Code и ОС Linux(Ubuntu 18.04).

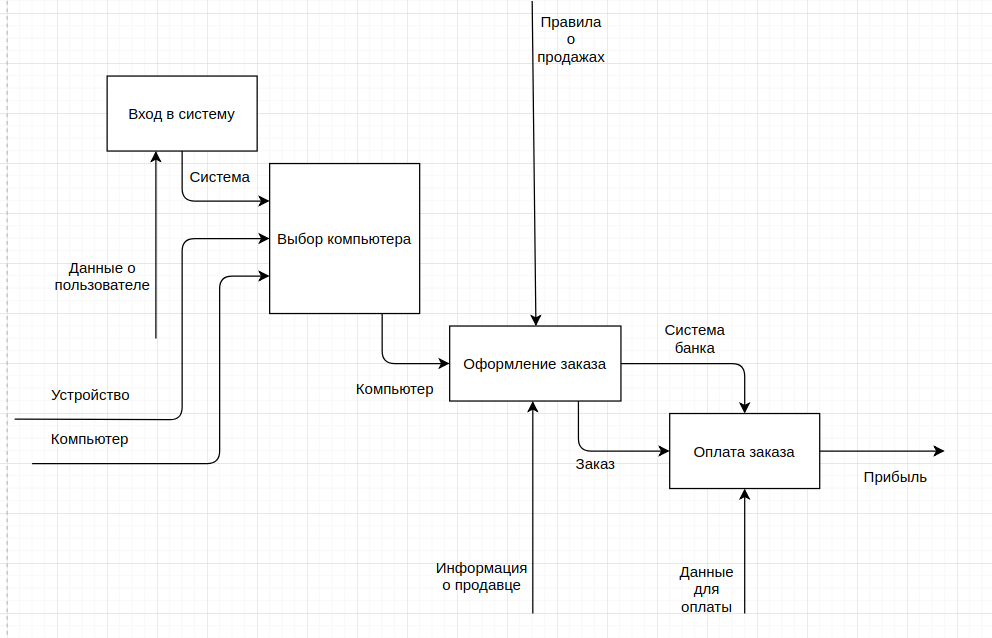
**3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0**

Главный процесс системы, представляет из себя продажу компьютеров. Клиент выбирает понравившийся ему компьютер и совершает сделку. Тем самым, зарегистрировав в системе информацию о себе и о товаре. Данный процесс представлен на следующей контекстной диаграмме первого уровня.



Как видно этот глобальный процесс можно разбить на подпроцессы.

Данные подпроцессы представлены контекстной диаграммой верхнего уровня.



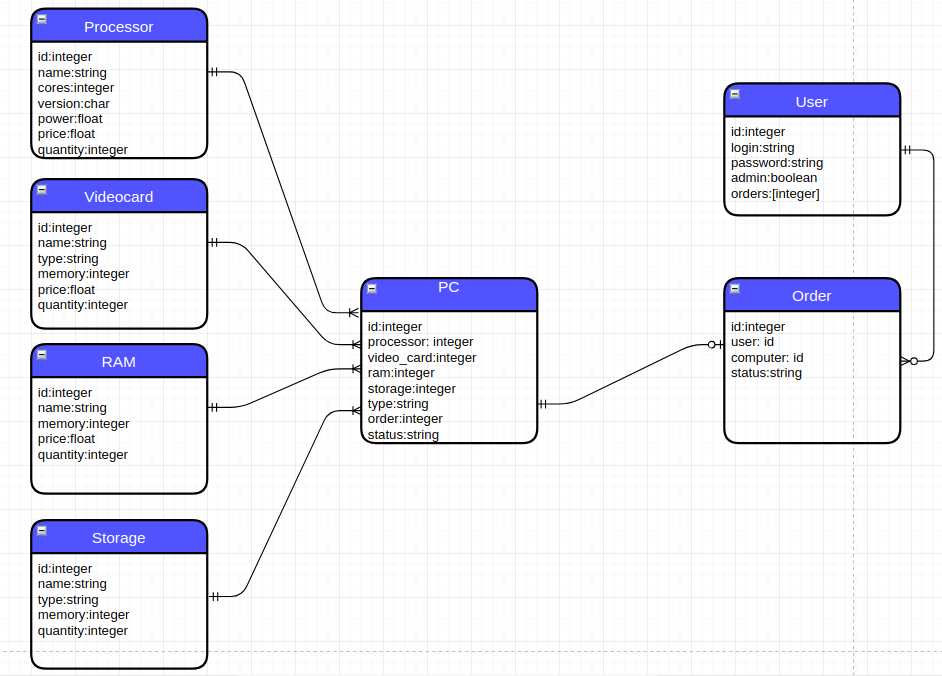
Как видно на данной диаграмме, глобальный процесс разбит на:

1. Вход в систему продавца
2. Продавец вносит данные компьютера, который покупают
3. Оформляет заказ
4. И клиент оплачивает заказ

В входные и выходные данные также можно увидеть на диаграмме.

**4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ**

Необходимо создать информационную модель, чтобы в базе данных SQL максимально подробно отобразить суть. Представлением информационной модели является набор таблиц в базе данных, связи между таблицами и атрибуты.

****

Компьютер как основной объект будет состоять из процессора, видеокарты, оперативной памяти, хранилища память(HDD или SSD). Для того чтобы пользователь сам мог собрать компьютер.

Компьютер связан с заказом 1:1 при этом компьютер может существовать без заказа, пока его не захотел купить клиент.

Заказ имеет связь с пользователем, а именно продавцом, что позволит быстро составлять статистику о продавце и оценивать результат его работы.

На складе может быть большое количество одинаковых устройств например видеокарт Nvidia 740m поэтому для устройств были введены такие значения как количество(quantity).

**5. МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ КРАТКОЕ ОПИСАНИЕ**

**5.1 Диаграмма вариантов использования**

Диаграмма вариантов использования в UML — диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

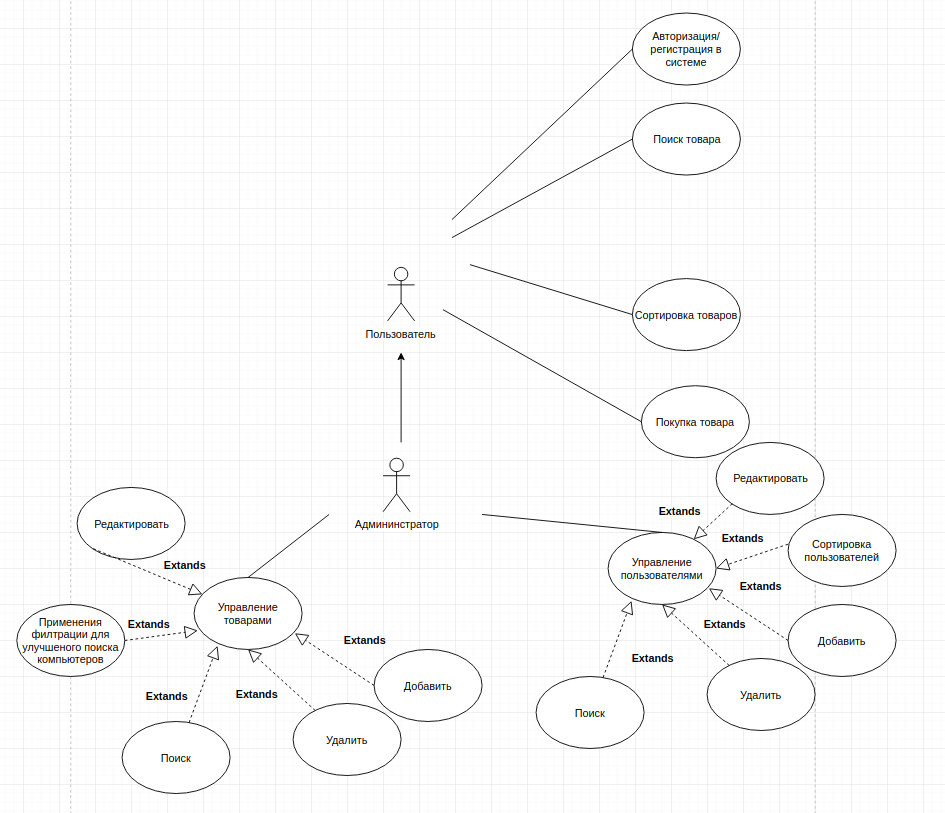
В языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:

• Отношение ассоциации

• Отношение расширения

• Отношение обобщения

• Отношение включения



Выше представлена диаграмма вариантов использования. Пользователь имеет два способа для авторизации:

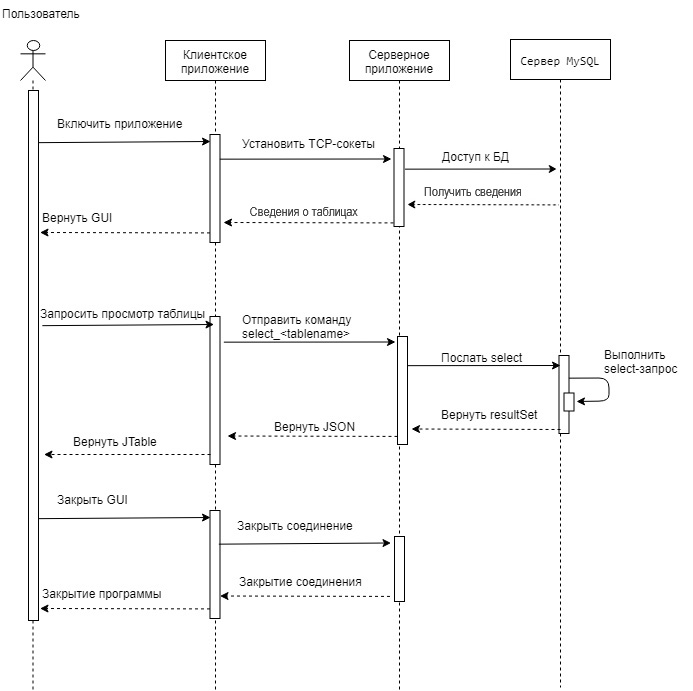
• Администратор

• Сотрудник магазина

**5.2 Диаграмма последовательности**

Диаграмма последовательности — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определенного объекта и взаимодействие актёров ИС в рамках какого-либо определённого прецедента. Всего существует 5 видов стрелок: синхронное сообщение, ответное сообщение, асинхронное сообщение.

Ниже представлена диаграмма описывающая работу программы.



**5.3 Диаграмма классов.**

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов, методов, интерфейсов и взаимосвязей между ними.

Графически класс изображается в виде прямоугольника, разделенного на 3 блока горизонтальными линиями:

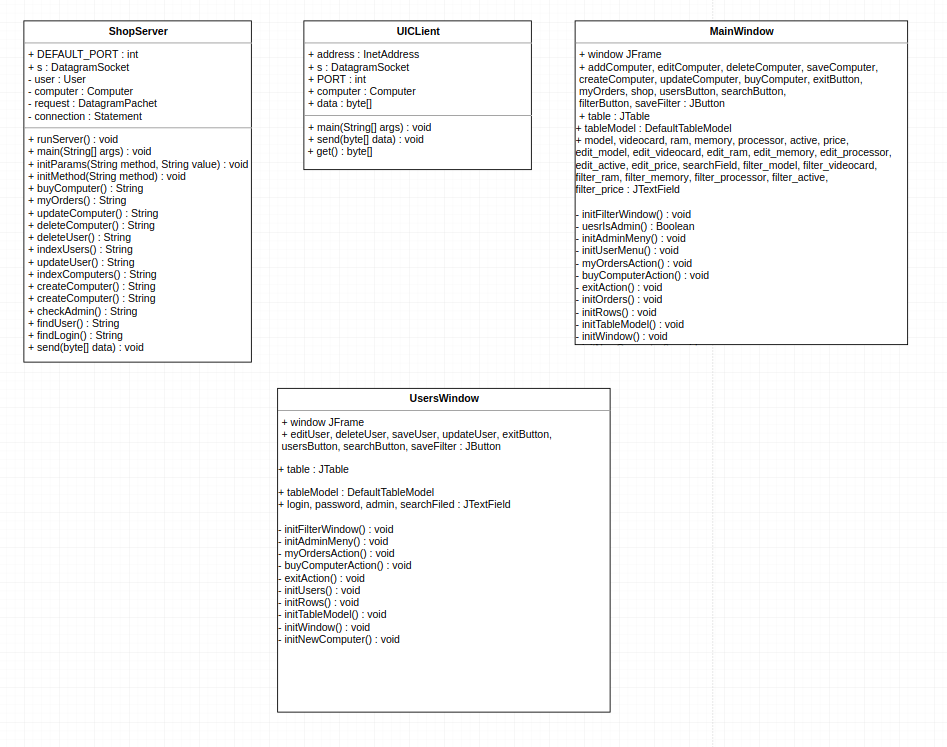
• имя класса

• атрибуты (свойства) класса

• операции (методы) класса.

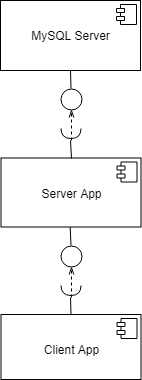
Возможные уровни видимости атрибутов:открытый, защищенный, закрытый.

Типа связей в UML: зависимость, ассоциация, обобщение, реализация.



**5.4 Диаграмма компонентов.**

Диаграмма компонентов — элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.



**5.5 Диаграмма развертывания.**

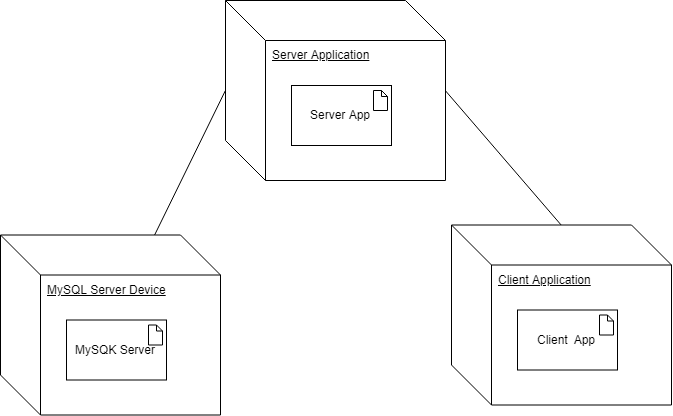
Диагра́мма развёртывания в UML моделирует физическое развертывание артефактов на узлах

Цели, преследуемые при разработке диаграммы развертывания:

• • определить распределение компонентов системы по ее физическим узлам.

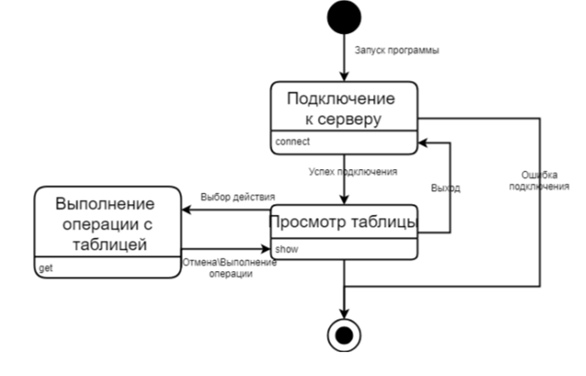
• • показать физические связи между всеми узлами реализации системы на этапе ее исполнения.

• • выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.



**5.6 Диаграмма состояний**

Диаграмма состояний — ориентированный граф для конечного автомата, в котором вершины обозначают состояния дуги показывают переходы между двумя состояниями На практике вершины обычно изображаются в виде окружностей и, если нужно, двойных окружностей



**6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ**

**2.1 Сервер**

Класс ShopServer включает в себя все необходимые методы и инструменты для обработки информации из базы данных.

Я строил весь оборот данных на строках. Т.к. было удобно переводить их в байты.

После запуска сервер принимает запрос в виде строки состоящей из параметра и название метода. В начале идут обязательные параметры - данные о пользователе. Потом параметры для метода и потом сам метод.

Пример:

login:user1,password:3,method:adminCheck

Сервер парсит это строку на массив и присваивает значения нужным полям. Если приходит неизвестный параметр, ответ клиенту будет равен 0, что означает необработанный запрос.

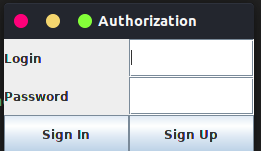
После обработки параметров, запускается необходимый метод, метод в свою очередь делает запрос в БД, обрабатывает его и результат отправляется клиенту в виде результата какого того значения или же “1” как успешно обработанный запрос или “0” как не обработанный.

Класс Computer включает в себя поля для хранения данных о компьютере.

Класс User включает в себя поля для хранения данных о пользователе.

**2.1 Клиент**

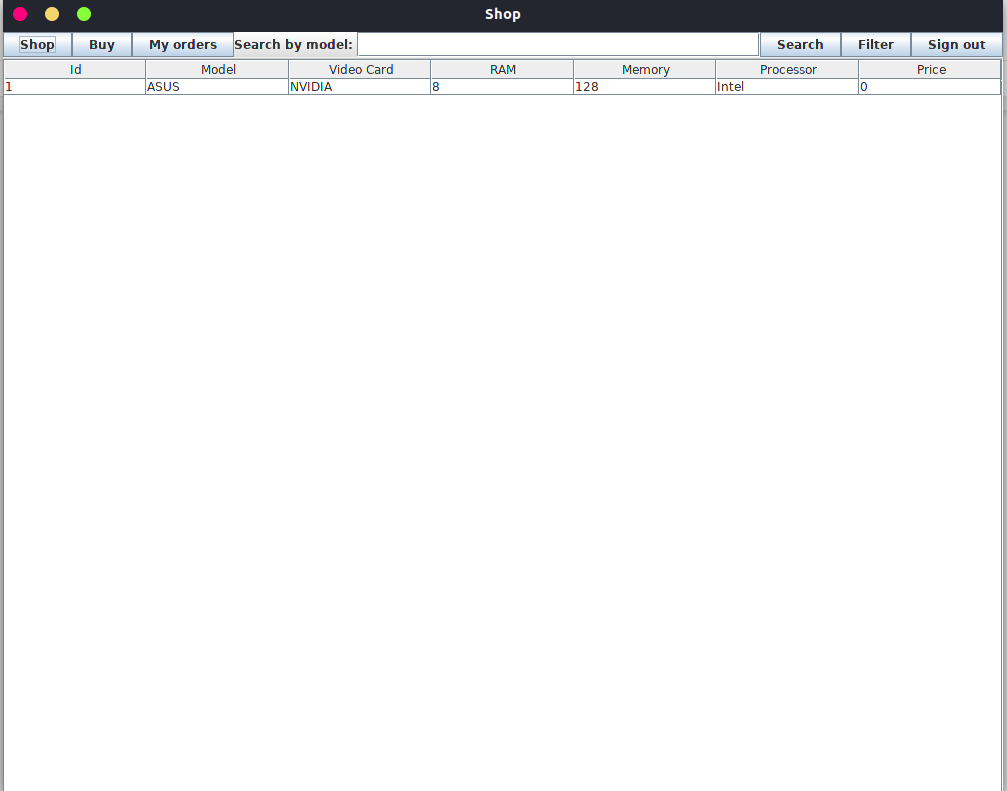
Класс Authorization включает в себя окно для ввода логина и пароля, кнопка для регистрации и входа в систему.



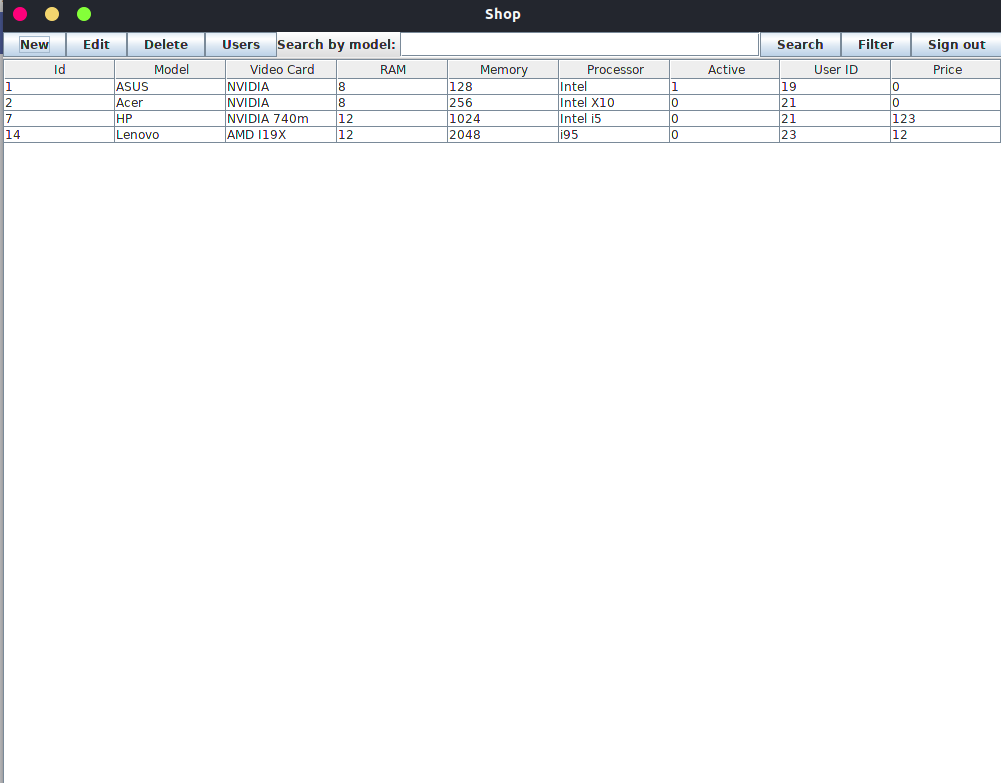
После ввода данных информация отправляется к серверу и получает ответ на регистрацию или вход.

Класс MainWindow главное окно, которое включается в себя меню пользователя или админа, таблицу компьютеров с описанием и ценами.

Меню пользователя: покупка компьютера, выбор таблицы компьютеров магазина или купленных. Поиск.

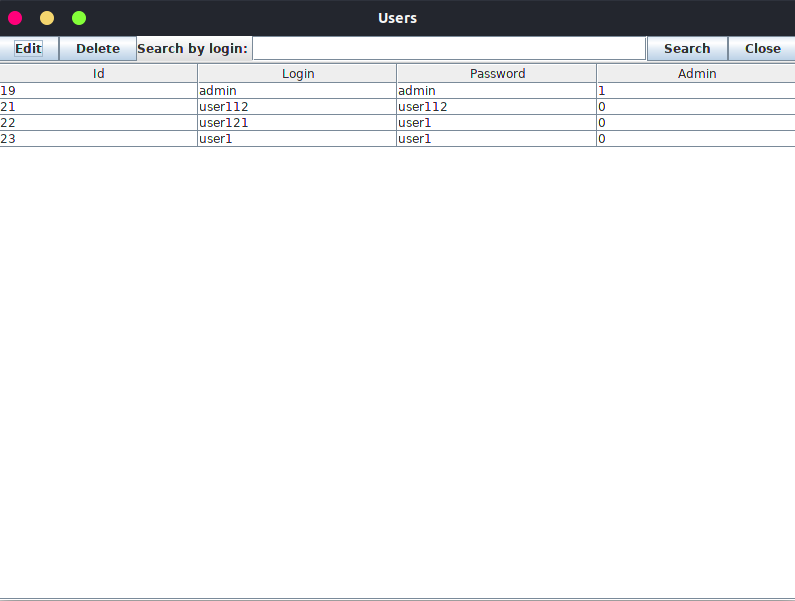


Меню админа включает в себя кнопки для управления компьютерами. И возможность открыть таблицу пользователей.



Клиент на генерирует запрос из строки вытягивая необходимую информацию из полей для ввода.

Класс UsersWindow включает в себя таблицу пользователей, которую также можно сортировать, найти необходимого пользователя и обновить/удалить его.



**ЗАКЛЮЧЕНИЕ**

Была подготовлена первая версия проекта, она весьма голая, но очень пытался попасть в сроки. Существует явные недоработки в защите. И в дизайне. Но проекту явно есть куда расти.

Моя главная задача была написать систему учета продаж, в моем понимании написать таблицу Excel в методами… Поэтому я смотря шире реализовал мини магазин. Который в будущем сможет включить в себя клиента который тоже будет торговать. И система будет напоминать торговую площадку.

Как по мне, использовать UDP в наше время жутко неудобно и не логично. Ведь наступил век интернета, так что на замену лучше подойдет WEB. Ведь интернет-магазин звучит уже лучше, чем магазин-приложение, клиент которого нужно устанавливать и т.д.

Используемый язык также древность. Есть куда более современные языки, приятные по синтаксису. Например: Ruby, Python. Сравнивая код, его можно было упростить заменив 18 строк одной. А порой и кучи методов одним.

Но в общем я считаю задача была выполнена.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

[1] Программирование сетевых приложений[ЭОРД]

[2] Википедия [Электронный ресурс]. – Режим доступа https://ru.wikipedia.org

**ПРИЛОЖЕНИЕ**

**ShopServer.java**

import java.net.\*;

import java.sql.\*;

import java.util.Properties;

import java.util.\*;

import javax.swing.\*;

import java.io.\*;

public class ShopServer {

public final static int DEFAULT\_PORT = 8001;

public static DatagramSocket s;

private static User user = new User();

private static Computer computer = new Computer();

private static DatagramPacket request;

private static Statement connection;

public static void runServer() throws IOException {

try {

boolean stopFlag = false;

byte[] data;

System.out.println("UDPServer: Started on " + s.getLocalAddress() + ":" + s.getLocalPort());

while(!stopFlag) {

data = new byte[10240];

request = new DatagramPacket(data, data.length);

s.receive(request);

try {

System.out.println(new String(request.getData()).trim());

for( String param : new String(request.getData()).trim().split(",")) {

String[] kv = param.split(":");

System.out.println(kv[0]+ " " + kv[1]);

initParams(kv[0], kv[1]);

}

} catch(Exception e) {

System.out.println("==== SERVER RUN SERVER EXCEPTION ====");

e.printStackTrace();

send("0".getBytes());

}

}

System.out.println("UDPServer: Stopped");

} finally {

if (s != null) {

s.close();

}

}

}

public static void main(String[] args) {

try {

s = new DatagramSocket(DEFAULT\_PORT);

Class.forName("com.mysql.jdbc.Driver");

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/ComputerShop?autoReconnect=true&useSSL=true", "root", "password").createStatement();

runServer();

} catch(Exception e) {

System.out.println("==== SERVER CONNECT EXCEPTION ====");

e.printStackTrace();

}

}

public static void initParams(String method, String value) {

try {

switch(method.trim()) {

case "login":

user.login = value;

break;

case "password":

user.password = value;

break;

case "id":

computer.id = value;

break;

case "model":

computer.model = value;

break;

case "videocard":

computer.videocard = value;

break;

case "ram":

computer.ram = Integer.parseInt(value);

break;

case "memory":

computer.memory = Integer.parseInt(value);

break;

case "processor":

computer.processor = value;

break;

case "active":

computer.active = String.valueOf(Boolean.valueOf(value));

break;

case "price":

computer.price = Integer.parseInt(value);

break;

case "user\_id":

user.user\_id = value;

break;

case "user\_login":

user.user\_login = value;

break;

case "user\_password":

user.user\_password = value;

break;

case "user\_admin":

user.user\_admin = value;

break;

case "method":

initMethod(value);

break;

}

} catch(Exception e) {

System.out.println("==== SERVER INIT PARAMS EXCEPTION ====");

send("0".getBytes());

}

}

public static void initMethod(String method) {

System.out.println("|" + method + "|");

switch(method.trim()) {

case "findUser":

System.out.println("==== SERVER START FIND USER ====");

send(findUser().getBytes());

break;

case "createUser":

System.out.println("==== SERVER START CREATE USER ====");

send(createUser().getBytes());

break;

case "checkAdmin":

System.out.println("==== SERVER START CHECK ADMIN ====");

send(checkAdmin().getBytes());

break;

case "createComputer":

System.out.println("==== SERVER START CREATE COMPUTER ====");

send(createComputer().getBytes());

break;

case "indexComputers":

System.out.println("==== SERVER START INDEX COMPUTERS ====");

send(indexComputers().getBytes());

break;

case "deleteComputer":

System.out.println("==== SERVER START DELETE COMPUTER ====");

send(deleteComputer().getBytes());

break;

case "updateComputer":

System.out.println("==== SERVER START UPDATE COMPUTER ====");

send(updateComputer().getBytes());

break;

case "buyComputer":

System.out.println("==== SERVER START BUY COMPUTER ====");

send(buyComputer().getBytes());

break;

case "myOrders":

System.out.println("==== SERVER START MY ORDERS ====");

send(myOrders().getBytes());

break;

case "indexUsers":

System.out.println("==== SERVER START INDEX USERS ====");

send(indexUsers().getBytes());

break;

case "updateUser":

System.out.println("==== SERVER START UPDATE USER ====");

send(updateUser().getBytes());

break;

case "deleteUser":

System.out.println("==== SERVER START DELETE USER ====");

send(deleteUser().getBytes());

break;

default:

System.out.println("==== SERVER INIT METHOD EXCEPTION ====");

send("0".getBytes());

break;

}

}

public static String buyComputer() {

try {

connection.execute("SET FOREIGN\_KEY\_CHECKS=1;");

connection.executeUpdate("UPDATE computers SET user\_id=" + findUser() + ", active=false WHERE id=" + computer.id + ";");

System.out.println("==== TRUE BUY COMPUTER ====");

return "1";

} catch(Exception e) {

System.out.println("==== BUY COMPUTER EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String myOrders() {

try {

ResultSet myResultSet = connection.executeQuery(

"SELECT COUNT(\*) AS count\_objects FROM computers WHERE user\_id=" + findUser() + ";"

);

myResultSet.next();

String[] data = new String[myResultSet.getInt("count\_objects")];

myResultSet = connection.executeQuery("SELECT \* FROM computers WHERE user\_id=" + findUser() + ";");

int i = 0;

while(myResultSet.next()) {

String[] row = { myResultSet.getString("id"), myResultSet.getString("model"),

myResultSet.getString("videocard"), myResultSet.getString("ram"),

myResultSet.getString("memory"), myResultSet.getString("processor"), myResultSet.getString("price") };

data[i] = String.join("&", row);

System.out.println(data[i]);

i++;

}

System.out.println("==== TRUE MY ORDERS ====");

return String.join("#", data);

} catch(Exception e) {

System.out.println("==== MY ORDERS EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String updateComputer(){

try {

connection.executeUpdate("UPDATE computers SET model='" + computer.model + "', videocard='" +

computer.videocard + "', ram=" + String.valueOf(computer.ram) + ", memory=" +

String.valueOf(computer.memory) + ", processor='" + computer.processor + "', active=" + computer.active +

", price=" + String.valueOf(computer.price) + " where id=" + String.valueOf(computer.id) + ";");

System.out.println("==== UPDATE COMPUTER TRUE ====");

return "1";

} catch(Exception e) {

System.out.println("==== SERVER UPDATE COMPUTER EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String deleteComputer() {

try {

connection.executeUpdate("DELETE FROM computers WHERE id=" + computer.id + ";");

System.out.println("==== TRUE DELETE COMPUTER ====");

return "1";

} catch(Exception e) {

System.out.println("==== DELETE COMPUTER EXCEPTION ====");

return "0";

}

}

public static String deleteUser() {

try {

connection.executeUpdate("DELETE FROM users WHERE id=" + user.user\_id + ";");

System.out.println("==== TRUE DELETE USER ====");

return "1";

} catch(Exception e) {

System.out.println("==== DELETE USER EXCEPTION ====");

return "0";

}

}

public static String indexUsers() {

try {

String[] data;

ResultSet myResultSet = connection.executeQuery(

"SELECT COUNT(\*) AS count\_objects FROM users;"

);

myResultSet.next();

data = new String[myResultSet.getInt("count\_objects")];

myResultSet = connection.executeQuery("SELECT \* FROM users;");

int i = 0;

while(myResultSet.next()) {

String[] row = { myResultSet.getString("id"), myResultSet.getString("login"),

myResultSet.getString("password"), myResultSet.getString("admin")};

data[i] = String.join("&", row);

System.out.println(data[i]);

i++;

}

return String.join("#", data);

} catch(Exception e) {

System.out.println("==== INDEX USERS EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String updateUser(){

try {

connection.executeUpdate("UPDATE users SET login='" + user.user\_login + "', password='" + user.user\_password + "', admin='" + user.user\_admin + "' WHERE id=" + user.user\_id + ";");

System.out.println("==== UPDATE USER TRUE ====");

return "1";

} catch(Exception e) {

System.out.println("==== SERVER UPDATE USER EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String indexComputers() {

try {

String[] data;

if(checkAdmin().equals("0")) {

ResultSet myResultSet = connection.executeQuery(

"SELECT COUNT(\*) AS count\_objects FROM computers WHERE active=true;"

);

myResultSet.next();

data = new String[myResultSet.getInt("count\_objects")];

myResultSet = connection.executeQuery("SELECT \* FROM computers WHERE active=true;");

int i = 0;

while(myResultSet.next()) {

String[] row = { myResultSet.getString("id"), myResultSet.getString("model"),

myResultSet.getString("videocard"), myResultSet.getString("ram"),

myResultSet.getString("memory"), myResultSet.getString("processor"), myResultSet.getString("price") };

data[i] = String.join("&", row);

System.out.println(data[i]);

i++;

}

} else {

ResultSet myResultSet = connection.executeQuery(

"SELECT COUNT(\*) AS count\_objects FROM computers;"

);

myResultSet.next();

data = new String[myResultSet.getInt("count\_objects")];

myResultSet = connection.executeQuery("SELECT \* FROM computers;");

int i = 0;

while(myResultSet.next()) {

String[] row = { myResultSet.getString("id"), myResultSet.getString("model"),

myResultSet.getString("videocard"), myResultSet.getString("ram"),

myResultSet.getString("memory"), myResultSet.getString("processor"), myResultSet.getString("active"),

myResultSet.getString("user\_id"), myResultSet.getString("price") };

data[i] = String.join("&", row);

System.out.println(data[i]);

i++;

}

}

return String.join("#", data);

} catch(Exception e) {

System.out.println("==== INDEX COMPUTERS EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String createComputer() {

try {

connection.executeUpdate("INSERT INTO computers(model, videocard, ram, memory, processor, price, user\_id) "

+ "VALUES ('" + String.join("', '", computer.returnParams()) + "'," + findUser() + ");");

System.out.println("==== CREATE COMPUTER TRUE ====");

return "1";

} catch(Exception e) {

System.out.println("==== SERVER CREATE COMPUTER EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String checkAdmin() {

try {

ResultSet myResultSet = connection.executeQuery(

"SELECT u.admin FROM users u WHERE u.login='" + user.login + "' and u.password='" + user.password + "';"

);

myResultSet.next();

System.out.print(myResultSet.getString("admin"));

System.out.println("==== CHECK ADMIN TRUE ====");

return myResultSet.getString("admin");

} catch(Exception e) {

System.out.println("==== SERVER CHECK ADMIN EXCEPTION ====");

return "0";

}

}

public static String createUser() {

if (Integer.valueOf(findLogin()) > 0) {

System.out.println("==== SERVER CREATE USER EXCEPTION ====");

return "0";

} else {

try {

connection.executeUpdate("INSERT INTO users(login, password) "

+ "VALUES ('" + user.login + "', '" + user.password + "');");

System.out.println("==== CREATE USER TRUE ====");

return String.valueOf(findUser());

} catch(Exception e) {

System.out.println("==== SERVER CREATE USER EXCEPTION ====");

return "0";

}

}

}

public static String findUser() {

try {

ResultSet myResultSet = connection.executeQuery(

"SELECT u.id FROM users u WHERE u.login='" + user.login + "' and u.password='" + user.password + "';"

);

System.out.println("SELECT u.id FROM users u WHERE u.login='" + user.login + "' and u.password='" + user.password + "';");

myResultSet.next();

System.out.println("==== FIND USER TRUE ====");

return myResultSet.getString("id");

} catch(Exception e) {

System.out.println("==== SERVER FIND USER EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static String findLogin() {

try {

ResultSet myResultSet = connection.executeQuery(

"SELECT u.id FROM users u WHERE u.login='" + user.login + "';"

);

myResultSet.next();

System.out.println("==== SERVER FIND LOGIN TRUE ====");

return myResultSet.getString("id");

} catch(Exception e) {

System.out.println("==== SERVER FIND LOGIN EXCEPTION ====");

e.printStackTrace();

return "0";

}

}

public static void send(byte[] data) {

try {

s.send(new DatagramPacket(data, data.length, request.getAddress(), request.getPort()));

System.out.println("SEND MESSAGE");

} catch(Exception e) {

System.out.println("==== SERVER SEND EXCEPTION ====");

}

}

}

**UIClient.java**

import java.net.\*;

import java.io.\*;

public class UIClient {

public static InetAddress address;

public final static int PORT = 8001;

public static DatagramSocket s;

public static byte[] data = new byte[10240];

public static void send(byte[] data) {

try {

s.send(new DatagramPacket(data, data.length, address, PORT));

} catch(Exception e) {

System.out.println("==== 1 EXCEPTION ====");

}

}

public static byte[] get() {

try {

data = new byte[512];

DatagramPacket result = new DatagramPacket(data, data.length);

s.receive(result);

return result.getData();

} catch(Exception e) {

return "0".getBytes();

}

}

public static void main(String[] args) {

try {

address = InetAddress.getByName("127.0.0.1");

s = new DatagramSocket();

new Authorization();

} catch(IOException ex) {

ex.printStackTrace();

System.out.println("==== 2 EXCEPTION ====");

}

}

}