СОДЕРЖАНИЕ

Введение 3
1 Описание задачи на исследование и ее проектирование 4
1.1 Формулировка и описание и задачи 4
1.2 Разработка требования к программному продукту 4
1.3 Проектирование программного продукта 7
2. Реализация программного продукта 10
2.1 Обоснование выбора инструментов для разработки 10
2.2 Технология связи приложения с базой данных 12
2.3 Анализ характерных частей кода 15
3 Тестирование и развёртывание приложения 23
3.1 Разработка плана тестирования 23
3.2 Тестирование приложение 24
3.3 Развертывание приложения 25
Заключение 26
Список использованных источников 27
ПРИЛОЖЕНИЕ А – UML диаграмма 28
ПРИЛОЖЕНИЕ В – Схема базы данных 29

ВВЕДЕНИЕ

В наше время, когда информационные технологии сопровождают человека на каждом шагу, сложно представить сферу деятельности, в которой не применена какая-либо автоматизация процессов. В данной курсовой работе речь пойдет об автоматизации процессов службы доставки лекарств. Важность такой службы очевидна. Лекарства необходимы в основном пожилым людям, которым, порой, сложно просто взять и пойти в аптеку, чтобы купить медикаменты, прописанные врачом. Однако, практически у каждого человека есть телефон для связи или компьютер с электронной почтой, с помощью которых, любой желающий может заказать доставку лекарственных средств на дом.

Целью курсовой работы является создание работоспособной системы для служб доставки лекарств, в которой будет просто вести учет заказов, следить за своевременной доставкой и наличием лекарств на складе. У программы должен быть интуитивно понятный интерфейс. Вся информация будет храниться в базе данных.

Основная задача состоит в том, чтобы создать программу, в которой будут храниться данные о покупателях, заказы покупателей на доставку, храниться информация о лекарствах и их доступном количестве.

Программа будет состоять из формы входа, основного окна для пользователя и администратора, а также вспомогательных окон, в которых, будут добавляться новые покупатели, добавляться/удаляться лекарства, доставка которых осуществляется на текущий момент и т.д.

1 ОПИСАНИЕ ЗАДАЧИ НА ИССЛЕДОВАНИЕ И ЕЕ ПРОЕКТИРОВАНИЕ

1.1Формулировка и описание и задачи

Сама экспресс-доставка появилась очень давно, конечно, она сильно отличается от современной доставки. Однако его суть была сохранена, включая доставку товара получателю в определенные сроки, сохранив при этом его первоначальный внешний вид. В современном мире весь процесс кажется гораздо более сложным, начиная с получения запроса на доставку и заканчивая состоянием "завершенной" доставки.

Учитывая конкретные обстоятельства деятельности организации, трудно вести учет без внедрения программного обеспечения. В бумажном виде такой учет займет много времени, чтобы заполнить, отследить и сохранить всю необходимую информацию. Невозможно или трудно выбрать из тех программ, которые уже имеются на рынке, которые будут отвечать всем требованиям и тонкостям.

Решение этой проблемы заключается в том, чтобы написать приложение для ваших собственных нужд и требований. Использование базы данных для хранения информации о деятельности вашей компании облегчит работу по сохранению накопленных данных путем создания резервных копий базы данных. Если компания переезжает в новый офис, то дешевле транспортировать компьютеры с базами данных и приложениями, нежели транспортировать картотечные шкафы и сейфы с бумажными носителями. Кроме того, анализ, обработка и использование такой базы данных менее ресурсоемки.

Исходя из этого, нужно написать приложение, которое позволит решить все бумажные вопросы, а также ускорить обработку данных и уменьшить количество ошибок. Программа должна представлять собой приложение, которое в зависимости от роли отображает пользователю свои интерактивные возможности.

1.2 Разработка требования к программному продукту

Доступ к программе должен осуществляться после ввода логина и пароля, которые будут выдаваться пользователям после запуска программы. На главной форме программы должно быть основное меню, которое

позволит получить доступ к разрешенным для текущего пользователя разделам программы.

Главная форма должна иметь возможность зарегистрироваться или войти в свой аккаунт.

Форма пользователя должна содержать в себе возможность оформить новый заказ с выбором лекарств, их количества и временем доставки. Также нужно предусмотреть возможность пользователю просматривать текущие еще не доставленные заказы и уже доставленные.

Для работы администратора, на его форме должны быть расположены элементы взаимодействия со всеми таблицами. Нужно предусмотреть добавление/удаление элементов из таких таблиц как тип лекарств, лекарства, склад, пользователи, заказы.

Форма администратора также должна содержать информацию о лекарствах, у которых истекает срок годности.

Программа не должна хранить в своих модулях пользовательскую информацию. Вся информация введенная пользователем должна записываться в базу данных. Система управления базой данных должна быть MySQL.

Запуск программы сопровождается предложением ввести логин и пароль, или зарегистрироваться. Если пользователь выбирает регистрацию – он переходит на окно регистрации, где вводит свои данные, а именно фио, домашний адрес, номер телефона, после чего нажимает кнопку регистрация. Если регистрация произошла успешно, пользователь получает уведомление об этом и возвращается на страницу входа, если же нет – то получает соответствующее уведомление

После ввода данных необходимо нажать кнопку «Войти», данное событие сформирует запрос к базе данных для проверки введенных логина и пароля. Логин пользователя хранится в базе в открытом виде и сравнивается посимвольно.

Дальнейшая работа программы зависит от прав пользователя, под которым она была запущена. Если пользователь вводит неверные данные, программа предупреждает его об этом и просит повторно их ввести. После успешного ввода данных авторизации происходит проверка полномочий пользователя. Есть 2 вида пользователей: admin и user.

Если входит пользователь, открывается окно взаимодействия с пользователем, если администратор – окно администратора соответственно.

Форма работы с пользователем предоставляет ему на выбор меню с вариантами действий: «новый заказ», «текущие заказы.

При нажатии кнопки «новый заказ» будет осуществлен переход на форму оформление заказа. Она в себе содержит выбор из списка лекарства, количество этого лекарства, а также дату заказа.

На форме администратора можно перейти ко следующим формам работы с данными: типы лекарств, лекарства, лекарства на складе, пользователи, заказы, а также форма с информацией об лекарствах с истекающим сроком годности

Таким образом, мы имеем следующую диаграмму взаимодействия.

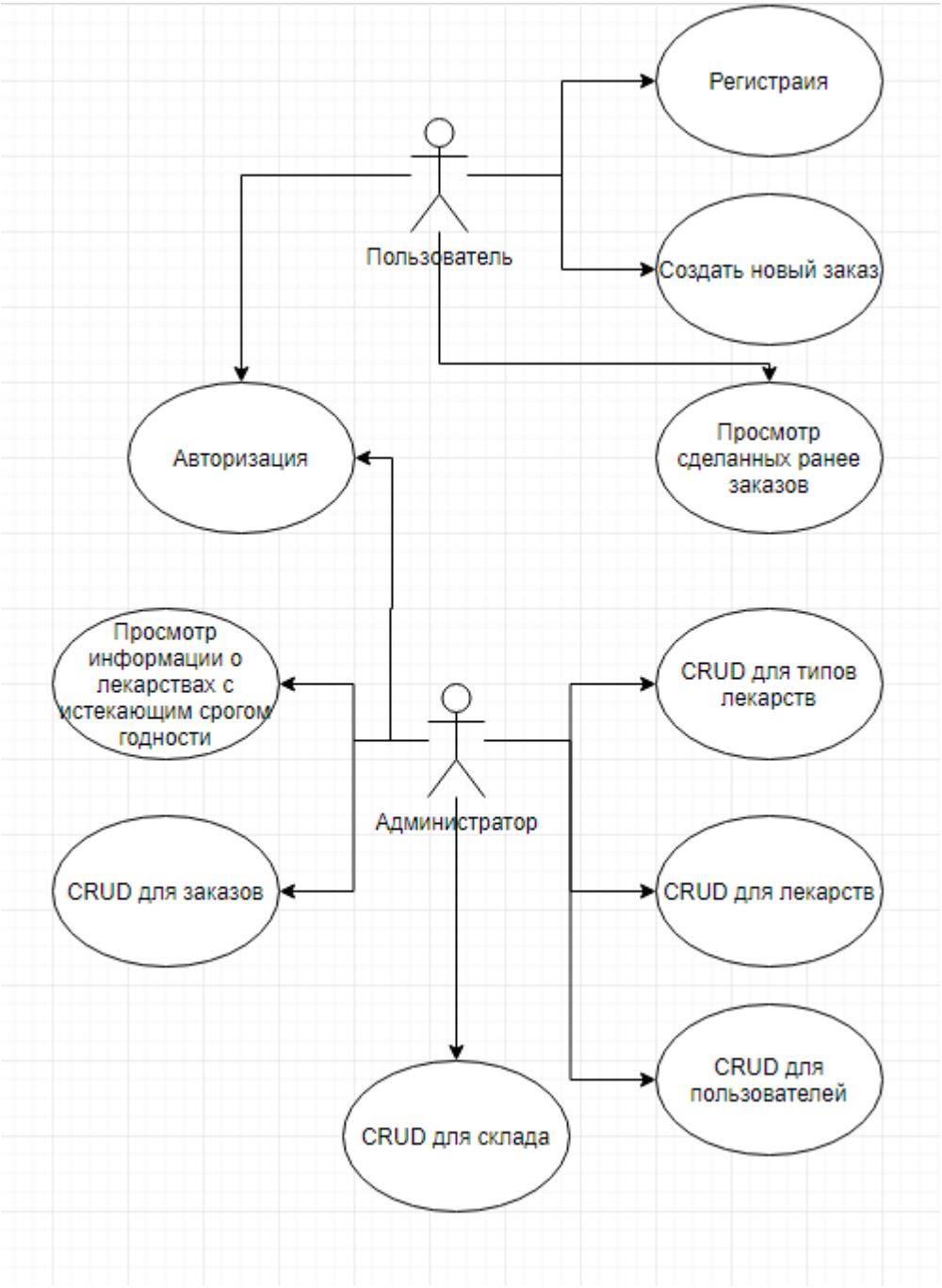


Рисунок 1.1 – UML диаграмма

1.3 Проектирование программного продукта

Для взаимодействия внутри приложения с базой данных, нам необходимо использовать классы-сущности для каждой из таблиц. Поскольку такие классы всецело повторяют типы полей из соответствующих таблиц, можем описать таблицу базы данных и сущность класса одного типа в одной таблице.

Так, у нас будут использоваться следующие таблицы и классы сущности:

Таблица 1 – MedicType (тип лекарства)

Название поля	**Тип поля**
type	string

Таблица, представляющая собой список типов для всех лекарст

Таблица 2 – Medic (лекарство)

Название поля	**Тип поля**
Id	string
Name	String
type	String
sizeType	string

Таблица, представляющая собой все лекарства, последний параметр – это объем одного лекарства, к примеру упаковка 100шт

Таблица 3 – Role (роли)

Название поля	**Тип поля**
role	string

Таблица ролей для пользователя, чтобы открывать форму в зависимости от роли.

Таблица 4 – Users (пользователь)

Название поля	**Тип поля**
Id	long
Fio	String
Adres	String
Phone	String
role	string

Таблица, представляющая собой список пользователей с их параметрами. В качестве типа данных для телефона используется строка, поскольку некоторые организации используют в поле номера телефона такие знаки как скобки, знака плюса.

Таблица 5 – Warehouse (склад)

Название поля	**Тип поля**
id	Long
Medic	Int
Count	Int
dateGetting	Date
dateDead	Date
priceGetting	Float
priceSelling	Float

Таблица, представляющая собой склад лекарств. Она дает данные о лекарстве на складе, его количестве, даты привоза на склад и даты истечения срока годности. Также дает информацию о цене закупке и цене продажи данного лекарства.

2. РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА

2.1 Обоснование выбора инструментов для разработки

Java-это объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Новый язык программирования, первоначально известный как Oak(Джеймс Гослинг), был разработан для бытовой электроники, но позже был переименован в Java для написания апплетов,приложений и серверного программного обеспечения

Отличительной особенностью Java по сравнению с другими универсальными языками программирования является то, что она обеспечивает высокую производительность программирования, а не эффективность работы приложений или использования памяти.

Java использует почти те же соглашения для объявления переменных, передачи параметров, операторов и управления потоком выполнения кода. Java добавляет Все хорошие возможности C++.

Технология языка Java сочетает в себе три ключевых элемента

Java предоставляет свои собственные апплеты для широкого использования-небольшие,надежные, динамичные, независимые от платформы, активные веб-приложения, встроенные в веб-страницы. Java-апплеты можно легко настроить и распространить среди потребителей, как и любой HTML-документ

Java высвобождает возможности объектно-ориентированной разработки приложений, сочетая простой и привычный синтаксис с мощной и простой в использовании средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты

Java предоставляет программистам богатый набор классов объектов для четкого абстрагирования многих системных функций для Windows, сетей и ввода-вывода.Ключевой особенностью этих классов является то, что они предоставляют независимые от платформы абстракции для различных системных интерфейсов

Самое большое преимущество Java заключается в том, что вы можете использовать его для создания приложений, которые могут работать на разных платформах. К интернету подключены различные типы компьютеров-Pentium PC,Macintosh,Sun workstation и др. Даже в компьютерах на базе процессоров Intel существует несколько платформ, таких как Microsoft

Windows версии 3.1, Windows95, Windows NT, OS/2, Solaris, различные операционные системы UNIX с графическими оболочками XWindows. В то же время, создавая веб-сервер в интернете, мы хотим, чтобы им пользовалось как можно больше людей. В этом случае помогут Java-приложения, предназначенные для работы на разных платформах и не зависящие от конкретного типа процессора и операционной системы.

Программы, написанные на языке программирования Java, можно разделить на две категории в зависимости от их использования.

К первой группе относятся Java-приложения, предназначенные для автономной работы под управлением специальных машин интерпретации Java. Реализация этой машины создана для всех основных машин. Вторая группа-это так называемые апплеты. Небольшая программа-это Java-приложение, которое интерпретируется виртуальной машиной Java, встроенной почти во все современные браузеры.

Приложения, относящиеся к первой группе, являются обычными автономными программами. Поскольку они не содержат машинного кода и работают под управлением специальных интерпретаторов, их производительность значительно ниже, чем у скомпилированных обычных программ, например, на языке программирования C++. Java-программы, не требующие повторной передачи, могут работать на любой платформе, что само по себе очень важно при разработке Интернета.

Поскольку AWT имеет недостатки, которые мы обсуждали ранее, Sun последовала за Awt, чтобы разработать набор графических компонентов, называемых Swing. Стоит отметить, что компоненты Swing полностью написаны на Java. 2D используется для рендеринга,и это время приносит несколько преимуществ. В результате набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности.

Элементы пользовательского интерфейса, такие как меню, кнопки и т. д., отображаются в пустом окне. Специфичные для платформы оконные системы должны только отображать окно и рисовать в нем графику.

Поэтому элементы графического интерфейса пользователя, созданные с помощью swing, выглядят и ведут себя точно так же, но не зависят от платформы, на которой запущена программа. Swing теперь является частью библиотеки Java base class (JFC).

Более тогоSun Microsystemsразработала независимый от платформы стиль по названием Metal, который сообщество программирующихна Javaпрозвало стилемJava.

Преимущества библиотеки Swing:

− содержит более богатый и удобный набор элементов пользовательского интерфейса

− намного меньше зависит от той платформы, на которой должна выполняться программа. Следовательно, меньше подвержена ошибкам, характерным для конкретной платформы

− обеспечивает одинаковое восприятие конечными пользователями приложений с ГПИ на разных платформах

− встроенный редактор форм почти во всех средах разработки

− на базе свинга есть много расширений типа SwingX

− поддержка различных стилей (Look and feel)

Недостатки:

− окно с множеством компонентов начинает подтормаживать;

− работа с менеджерами компоновки может стать настоящим кошмаром в сложных интерфейсах

Так как библиотека swing имеет больше возможностей для создания ГПИ, в дальнейшем мы будем использовать именно ее

2.2 Технология связи приложения с базой данных

JDBC (Java Database Connectivity) - это интерфейс, а не протокол, основанный на спецификации SAG CLI (SQL Access Group Call Level Interface).

Сам JDBC не работает и использует базовые абстракции и методы ODBC. Хотя стандарт API JDBC обеспечивает не только возможность работы через ODBC, но и возможность использования прямых ссылок на базы данных с использованием схем двойного или тройного связывания (см. Рис. 1), он сравнивается с широко используемым JDBC-ODBC-Bridge., (см. Рис. 2.2)

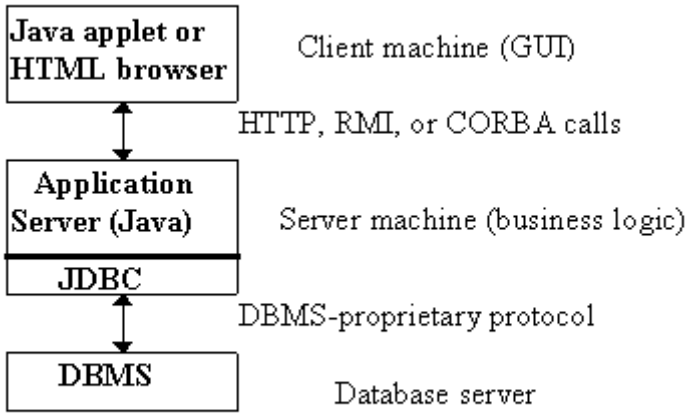


Рисунок 2.1 - Непосредственный доступ к базе данных по 3-х-звенной схеме.

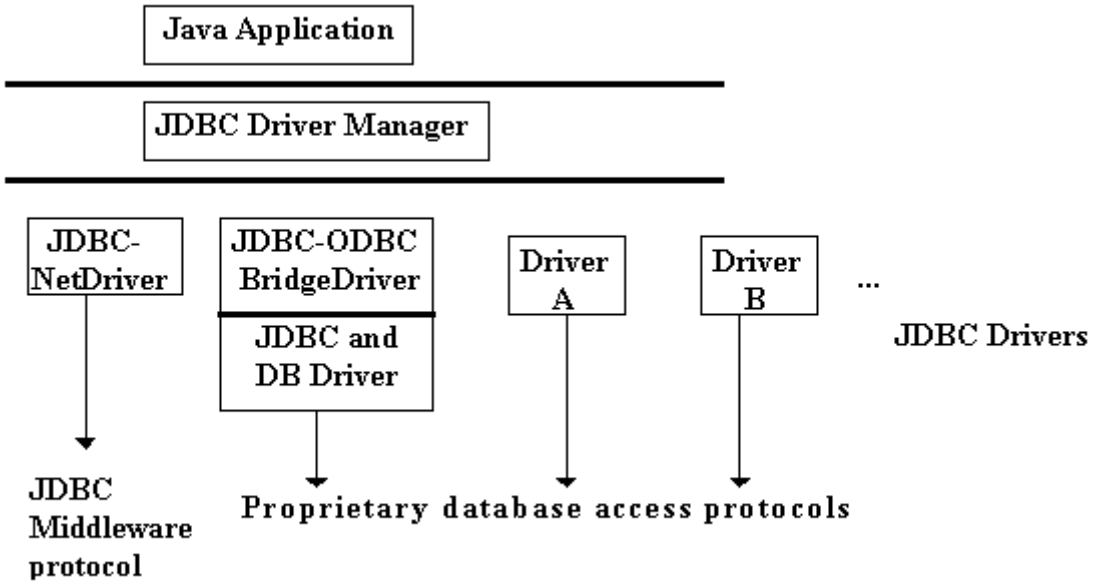


Рисунок 2.2 - Схема взаимодействия интерфейсов.

Даже беглого взгляда на Рис. 2 вполне достаточно, чтобы понять - общая схема взаимодействия интерфейсов в Java удивительным образом напоминает столь всем знакомую схему ODBC с ее гениальным изобретением драйвер-менеджера к различным СУБД и единого универсального пользовательского интерфейса. JDBC Driver Manager - это основной ствол JDBC-архитектуры. Его первичные функции очень просты - соединить Java-программу и соответствующий JDBC драйвер и затем выйти из игры. Естественно, что ODBC был взят в качестве основы JDBC из-за его популярности среди независимых поставщиков программного обеспечения и

пользователей. Но тогда возникает законный вопрос - а зачем вообще нужен JDBC и не легче ли было организовать интерфейсный доступ к ODBC-драйверам непосредственно из Java? Ответом на этот вопрос может быть только однозначное нет. Путь через JDBC-ODBC-Bridge, как ни странно, может оказаться гораздо короче.

ODBC нельзя использовать непосредственно из Java, поскольку он основан на интерфейсе C. Вызов из кода Java C нарушает всю концепцию Java,нарушает уязвимости безопасности и затрудняет перенос программ.

Перенос ODBC C API на Java API не рекомендуется. Например, Java не имеет указателей, и ODBC использует их.

ODBC слишком сложен для понимания. Он смешивает простые и сложные вещи и иногда применяет сложные параметры к самым простым запросам.

Java API необходим для реализации абсолютно чистого Java-решения. При использовании ODBC необходимо установить драйвер ODBC и диспетчер ODBC на каждом клиентском компьютере. В то же время драйверы JDBC полностью написаны на Java и могут быть легко перенесены на любую платформу с сетевого компьютера на мэйнфрейм.

JDBC API - это естественный Java-интерфейс с базовой абстракцией SQL и принимает дух и базовую абстракцию концепции ODBC, в конце концов, он реализован как реальный Java-интерфейс, совместимый с остальной частью Java-системы.

В отличие от интерфейса ODBC, JDBC гораздо проще организовать. Основная его часть-это драйвер, предоставляемый JavaSoft для доступа к источникам данных из JDBC. Этот драйвер является самым высоким драйвером в иерархии классов JDBC и называется DriverManager. Согласно установленным правилам интернета, база данных и средства ее обслуживания идентифицируются с помощью URL-адресов.

jdbc

где под понимается имя конкретного драйвера, или некоего механизма установления соединения с базой данных, например, ODBC. В случае применения ODBC, в URL-строку подставляется именно эта аббревиатура, а в качестве используется обычный DSN (Data Source Name), т.е. имя ODBC-источника из ODBC.INI файла. Например:

jdbc:odbc:dBase

В некоторых случаях вместо ODBC может быть использовано имя прямого сетевого сервиса к базе данных, например:

jdbc:dcenaming:accounts-payable,

или

jdbc:dbnet://ultra1:1789/state

В последнем случае часть URL //ultra1:1789/state представляет собой и описывает имя хоста, порт и соответствующий идентификатор для доступа к соответствующей базе данных.

Однако, как уже говорилось выше, чаще всего, все-таки используется механизм ODBC благодаря его универсальности и доступности. Программа взаимодействия между драйвером JDBC и ODBC разработана фирмой JavaSoft в сотрудничестве с InterSolv и называется JDBC-ODBC-Bridge. Она реализована в виде JdbcOdbc.class (для платформы Windows JdbcOdbc.dll) и входит в поставку JDK1.1. Помимо JdbcOdbc-библиотек должны существовать специальные драйвера (библиотеки), которые реализуют непосредственный доступ к базам данных через стандартный интерфейс ODBC. Как правило эти библиотеки описываются в файле ODBC.INI. На внутреннем уровне JDBC-ODBC-Bridge отображает медоды Java в вызовы ODBC и тем самым позволяет использовать любые существующие драйверы ODBC, которых к настоящему времени накоплено в изобилии.

2.3 Анализ характерных частей кода

Все классы приложения делятся на части, в зависимости от функционала. Так, у нас есть несколько групп классов, которые работают подобным друг-другу образом. Группы:

− Контроллеры

− ДАО

− Сущности

− Формы отображения

− Модели отображения в таблице

Формы отвечают за отображение данных пользователю. К примеру, форма MedicTypeFrame отображает пользователю таблицу типов товаров с кнопками взаимодействия.

Для каждой кнопки формы предусмотрены свои методы

Листинг 1 – Код кнопок формы типов лекарств

```
//отображение данные в полях при нажатии на строку таблицы
private void actionTableMouseClicked() {
```

```
            if (table.getRowCount() > 0)
                try {
                    int row = table.getSelectedRow();
                    textField.setText(table.getModel().getValueAt(row,
0).toString());
                } catch (Exception e) {
                }
        }
        //кнопка удаления
        private void actionDeleteButton() {
            if (table.getRowCount() > 0 && table.getSelectedRowCount() > 0) {
                int row = table.getSelectedRow();
                String type = table.getModel().getValueAt(row, 0).toString();

        ApplicationController.MedicTypeController.actionDeleteButton(type);
            }
        }
        //кнопка обновления
        private void actionUpdateButton() {
            int column = 0;
            int row = table.getSelectedRow();
            ApplicationController.MedicTypeController.actionUpdateButton(text
Field.getText(),table.getModel().getValueAt(row, 0).toString());
            refreshView();
        }
        //кнопка создания
        private void actionCreateButton() {
            ApplicationController.MedicTypeController.actionCreateButton(textF
ield.getText());
        }
        //кнопка поиска
        private void actionSearchButton() {
            ApplicationController.MedicTypeController.actionSearchButton(textF
ield_4.getText().trim(),table);
        }
```

Как видно из кода, для связи формы и функционала используются контроллеры, методы которого вызываются из формы

Листинг 2 – Контроллер типов лекарств

```
// конструктор
public MedicTypeController(JFrame frame) {
	super(frame);
	try {
		DAO = new MedicTypeDAO();
	} catch (Exception e) {e.printStackTrace();}
}
// поиск по полям
public void actionSearchButton(String name, JTable table) {
	try {
		List<MedicType> list = null;

		if (name != null && name.trim().length() > 0)
			list = DAO.search(name);
		else
			list = DAO.readAll();

		MedicTypeTableModal model = new
MedicTypeTableModal(list);
		table.setModel(model);
	} catch (Exception e) {
		JOptionPane.showMessageDialog(getFrame(), "Ошибка: " + e,
"Ошибка", JOptionPane.ERROR_MESSAGE);
	}
}
// добавление
public void actionCreateButton(String type) {
	if (type.length() > 0 ) {
		MedicType mt = new MedicType(type);
		try {
			DAO.create(mt);
			((MedicTypeFrame) getFrame()).refreshView();
			JOptionPane.showMessageDialog(getFrame(),
"Успешно добавлено", "Успех", JOptionPane.INFORMATION_MESSAGE);
		} catch (Exception e) {
```

```
                    JOptionPane.showMessageDialog(getFrame(), "Ошибка:
" + e, "Ошибка", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
        // обновление
        public void actionUpdateButton(String typeNew, String typeOld) {
            if (typeNew.length()>0 ) {
                try {
                    MedicType mt = new MedicType(typeNew);
                    DAO.update(typeNew,typeOld);
                    ((MedicTypeFrame) getFrame()).refreshView();
                    JOptionPane.showMessageDialog(getFrame(),
"Успешно обновлено", "Успех", JOptionPane.INFORMATION_MESSAGE);
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(getFrame(), "Ошибка:
" + e, "Ошибка", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
        // удаление
        public void actionDeleteButton(String type) {

                try {
                    DAO.Delete(type);
                    ((MedicTypeFrame) getFrame()).refreshView();
                    JOptionPane.showMessageDialog(getFrame(),
"Успешно удалено", "Успех", JOptionPane.INFORMATION_MESSAGE);
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(getFrame(), "Ошибка.
Возможно элемент используется в другой таблице" , "Ошибка",
JOptionPane.ERROR_MESSAGE);
                    System.out.println(e);
                }

        }
```

Для работы с данными используются классы DAO, которые напрямую работают с бд. В них передаются сущности, с которыми идет работа

Листинг 3 – DAO типа лекарств

```
// конструктор, подключающийся к бд
public MedicTypeDAO() throws Exception {
	Properties props = new Properties();
	props.load(new FileInputStream("db.properties"));

	String user = props.getProperty("user");
	String password = props.getProperty("password");
	String dburl = props.getProperty("dburl");
	myConn = DriverManager.getConnection(dburl, user, password);
	System.out.println("DB medicType connection success");
}
// чтение всех записей
public List<MedicType> readAll() throws Exception {
	List<MedicType> list = new ArrayList<MedicType>();
	Statement myStmt = null;
	ResultSet myRs = null;
	try {
		myStmt = myConn.createStatement();
		myRs = myStmt.executeQuery("SELECT * FROM type_medic");
		while (myRs.next()) {
			MedicType tempEntity = convertRowToEntity(myRs);
			list.add(tempEntity);
		}
		return list;
	} finally {
		close(myStmt, myRs);
	}
}
// поиск элементов
public List<MedicType> search(String name) throws Exception {
	List<MedicType> list = new ArrayList<MedicType>();
	PreparedStatement myStmt = null;
	ResultSet myRs = null;
```

```
            try {
                name = "%" + name + "%";
                myStmt = myConn.prepareStatement("SELECT * FROM
type_medic WHERE type LIKE ?");
                myStmt.setString(1, name);
                myRs = myStmt.executeQuery();
                while (myRs.next()) {
                    MedicType tempEntity = convertRowToEntity(myRs);
                    list.add(tempEntity);
                }
                return list;
            } finally {
                close(myStmt, myRs);
            }
        }
        // создание записи
        public void create(MedicType entity) throws Exception {
            PreparedStatement myStmt = null;
            try {
                myStmt = myConn
                        .prepareStatement("insert into type_medic" + "
(type)" + " values (?)");
                myStmt.setString(1, entity.getType());
                myStmt.executeUpdate();
            } finally {
                close(myStmt);
            }
        }
        // чтение определенной записи
        public List<MedicType> read(Long id) throws Exception {
            List<MedicType> list = new ArrayList<MedicType>();
            PreparedStatement myStmt = null;
            ResultSet myRs = null;
            try {
                myStmt = myConn
                        .prepareStatement("SELECT * FROM type_medic
WHERE id=?");
```

```
                myStmt.setLong(1, id);
                myRs=myStmt.executeQuery();
                while (myRs.next()) {
                    MedicType tempEntity = convertRowToEntity(myRs);
                    list.add(tempEntity);
                }
                return list;
            } finally {
                close(myStmt, myRs);
            }
        }
        // обновление записи
        public void update(String typeNew, String typeOld) throws Exception {
            PreparedStatement myStmt = null;
            try {
                myStmt = myConn.prepareStatement("UPDATE type_medic
SET type=? WHERE type=?");
                myStmt.setString(1, typeNew);
                myStmt.setString(2, typeOld);
                myStmt.executeUpdate();
            } finally {
                close(myStmt);
            }
        }
        // удаление записи
        public void Delete(String type) throws Exception {
            PreparedStatement myStmt = null;
            try {
                myStmt = myConn.prepareStatement("DELETE FROM
type_medic WHERE type=?");
                myStmt.setString(1, type);
                myStmt.executeUpdate();
            } finally {
                close(myStmt);
            }
        }
        // преобразование сущности в строку таблицы для отображения
```

```
    private MedicType convertRowToEntity(ResultSet myRs) throws SQLException {
        String type = myRs.getString("type");
        MedicType temp = new MedicType(type);
        return temp;
    }
    // закрытие всех соединений
    private static void close(Connection myConn, Statement myStmt, ResultSet myRs) throws SQLException {
        if (myRs != null) {
            myRs.close();
        }

        if (myStmt != null) {
            myStmt.close();
        }

        if (myConn != null) {
            myConn.close();
        }
    }
    // закрытие соединений
    private void close(Statement myStmt, ResultSet myRs) throws SQLException {
        close(null, myStmt, myRs);
    }
    // закрытие соединений
    private void close(Statement myStmt) throws SQLException {
        close(null, myStmt, null);
```

Модели таблиц используются только для правильного отображение столбцов таблицы.

3 ТЕСТИРОВАНИЕ И РАЗВЁРТЫВАНИЕ ПРИЛОЖЕНИЯ

3.1 Разработка плана тестирования

Контроль качества программы-это деятельность, которая позволяет определить, соответствует ли программный продукт ожидаемым результатам (т. е. заданным требованиям) при различных условиях, входных данных и средах. Качество программного продукта определяется определенными качественными характеристиками (качественная характеристика-это совокупность атрибутов программного средства, описывающая и оценивающая его качество). К качественным характеристикам первого уровня относятся:

Функция-это свойство набора программных средств, определяемое наличием и специфической функциональностью набора функций, удовлетворяющих заданному или неявному требованию. Функциональные требования определяют функциональность программного обеспечения и решаемые им задачи.

Надежность-это совокупность характеристик, которая представляет собой способность программных средств поддерживать заданный уровень пригодности в течение заданного интервала времени при заданном условии.

Доступность - это набор свойств программного средства, характеризующий усилия, необходимые для его тестирования в заданной или неявной пользовательской области. В ходе тестирования доступности было установлено, что разработанный интерфейс соответствует требованиям доступности.

Эффективность-это совокупность свойств программного средства, характеризующая те аспекты его применимости, уровень которых связан с характеристиками и временем использования ресурсов, что необходимо при заданных условиях эксплуатации.

Качество ремонтопригодности-это совокупность свойств программного средства, характеризующих работу, необходимую для его модификации.

Переносимость-это совокупность свойств программного средства, характеризующая его адаптивность к переходу из одной операционной среды в другую.

В зависимости от направления теста проверьте конкретную функциональность приложения. Как правило, процесс тестирования

записывается в виде плана тестирования и тестового случая (test case). План тестирования описывает стратегию тестирования,методы и инструменты тестирования, процесс тестирования и другие функции. TestCase описывает последовательную пошаговую операцию,используемую для проверки функциональности программных средств. Это минимальные базовые операции выверки для каждой функции или элемента приложения.

3.2 Тестирование приложение

Программа тестирования включает в себя проверку его работы в различных условиях для определения правильности его работы.

Разработка любой программы предполагает наличие ошибок в исходном тексте и борьбу с ними. Все почти бесчисленные возможные ошибки обычно делятся на несколько групп:

Синтаксическая ошибка;

Семантическая (логическая) ошибка.

Синтаксические ошибки-это самые простые ошибки, которые легко исправить на этапе компиляции. Их причина заключается в том, что в сервисе, значит, один из операторов набрал неверный ввод.

Семантические (логические) ошибки-самые сложные и неуловимые. Они ведут себя так, что программа действует не так, как ожидалось.

Последствия семантических ошибок могут быть самыми разными: неправильное содержимое окна, невыполненное или неправильное выполнение пользовательских команд, неправильное содержимое таблицы и так далее. Обработка семантических ошибок требует почти всего времени отладки.

В таблице 6 приведены результаты тестирования программного обеспечения. Используйте метод анализа граничных условий для тестирования.

Таблица 6– Test Case

Идентификатор	Описание Test Case	Ожидаемый результат	Полученный Результат
1	1.Открыть программу 2. Войти как админ	Переход на страницу админа	Открывается форма админа
1	1.Открыть программу 2. Войти как пользователь	Переход на страницу пользователя	Открывается форма пользователя

2	1.Открыть программу 2.Ввести несуществующие данные	Невозможность входа	Появление сообщения об ошибке, невозможно куда-то войти
3	1.Открыть программу 2. Войти как админ. 3.Перейти на страницу типа лекарства 4. Добавить тип	В таблицу добавился новый тип. В таблице	В таблице отображен новый тип.
5	1.Открыть программу 2. Перейти к форме регистрации 3.Зарегистрироваться	Добавление нового пользователя	Пользователю показалось уведомление об успешной регистрации, сейчас можно войти в систему

3.3 Развертывание приложения

Для запуска приложения у клиента должно быть установлено следующее программное обеспечение:

− Виртуальная машина Java и Jdk 1.8;

− Среда разработки Eclipse;

− MySSL Server;

После того как пользователь установил данное программное обеспечение следует импортировать создать базу данных, использовав скрипт «SQL» и открыть проект в IDE.

Для запуска приложения можно использовать комбинацию клавиш Alt+shift+x или круглую иконку с зеленым фоном и белым знаком воспроизведения.

Для экспортирования проекта как приложение, нужны выбрать проект из списка проектов, нажать по нему правой кнопкой мыши. Из перечисленных вариантов выбрать Export-java-Runnable JAR file. Далее выбрать место для хранения и нажать кнопку Export. После чего запускать приложение можно без открытия IDE.

ЗАКЛЮЧЕНИЕ

Результатом курсового проекта стала разработка программного обеспечения моделирование службы доставки лекарств.

При выполнении работы, были детально изучена используемая функциональность и возможности объектно-ориентированного языка Java.

Кроме того, работа способствовала приобредению навыков работы с Java Spring, JDBC, а также основные навыки рационального использования современного программного обеспечения персональных компьютеров для решения различных задач.

В результате курсового проекта все цели и задачи, поставленные для его разработки на начальном этапе, были полностью реализованы.

Программа соответствует исходному техническому заданию. В теории, ее можно использовать для большинства работ, связанных с медициной, в частности в организациях, занимающихся доставкой лекарств.

По итогам всех испытаний можно смело сделать вывод, что программный продукт соответствует всем заданным критериям оценки, а также его высокой надежности и простоте использования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Клюев, А.С. Наладка средств автоматизации и автоматических систем регулирования: Справочное пособие / А.С. Клюев, А.Т. Лебедев, С.А. Клюев. - М.: Альянс, 2009. - 368 с

2. MVC [Электронный ресурс]. – Режим доступа: https://javarush.ru/groups/posts/2536-chastjh-7-znakomstvo-s-patternom-mvc-model-view-controller. – Дата доступа: 10.10.2020.

3. Гупта, Арун Java EE 7. Основы / Арун Гупта. - М. : Вильямс, 2014. - 336 с

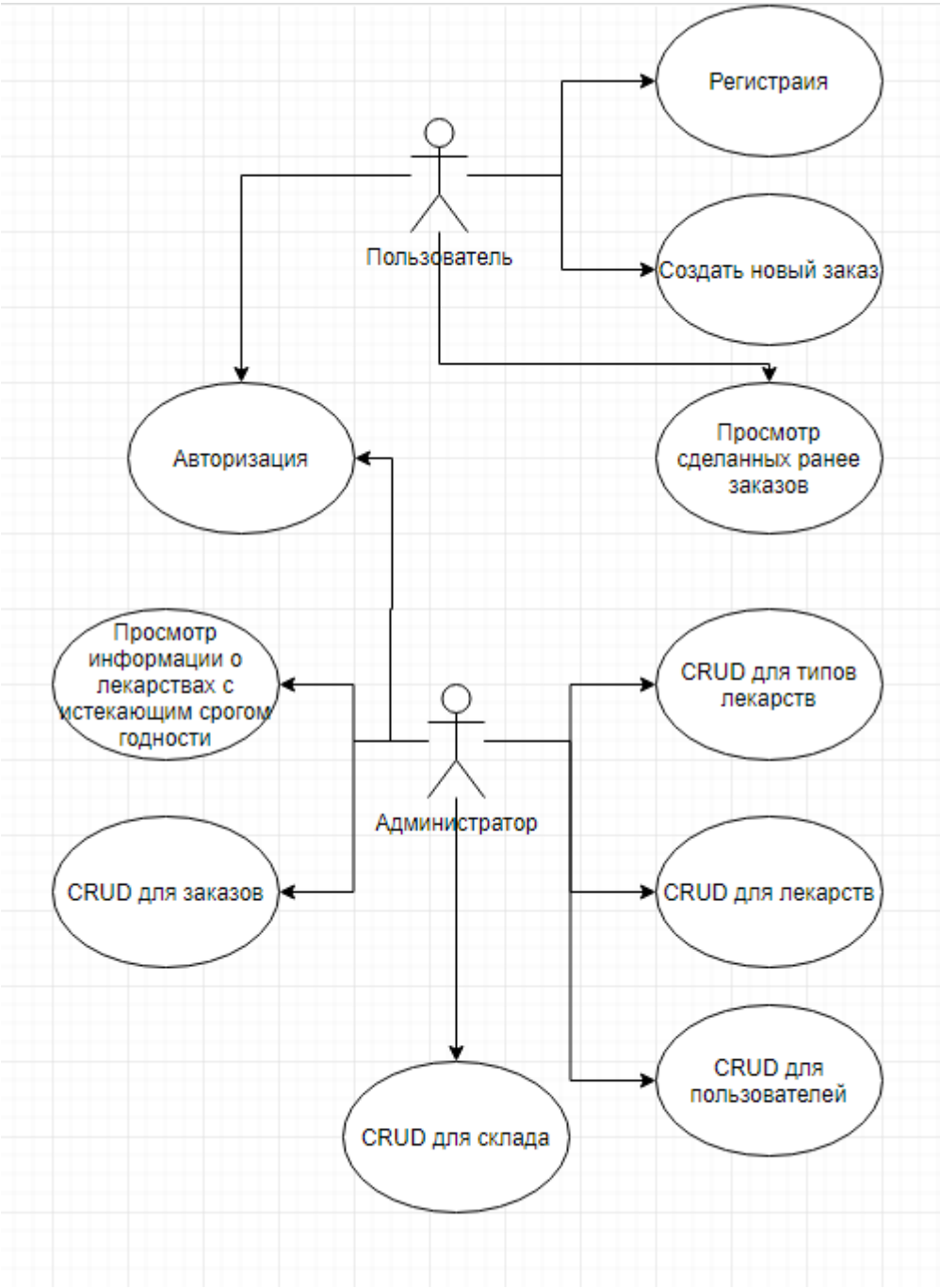
4. Java и базы данных JDBC [Электронный ресурс]. – Режим доступа: https://metanit.com/java/database/1.1.php. – Дата доступа: 10.11.2020.

5. Java Swing [Электронный ресурс]. – Режим доступа: http://java-online.ru/libs-swing.xhtml. – Дата доступа: 10.11.2020.

6. MySql [Электронный ресурс]. – Режим доступа: https://www.methodlab.ru/technology/mysql.shtml. – Дата доступа: 10.10.2020.

7. Куликов, С. С. / Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — Минск: Четыре четверти, 2017. — 312 с

ПРИЛОЖЕНИЕ А – UML ДИАГРАММА



ПРИЛОЖЕНИЕ В – СХЕМА БАЗЫ ДАННЫХ