Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных технологий

Кафедра проектирования информационных компьютерных систем

Дисциплина "Современные технологии проектирования информационных систем"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
06.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка информационной системы для предметной области "Виртуальная аптека"

БГУИР КР 1-40 05 01-10 № 180 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
06.07.2025
_____
(подпись студента)

2025

Реферат

БГУИР КР 1-40 05 01-10 № 180 ПЗ, гр. 784371

, Разработка информационной системы для предметной области "Виртуальная аптека", Минск: БГУИР - 2025.

Пояснительная записка 131681 с., 19 рис., 2 табл.

Ключевые слова:

Предмет Современные технологии проектирования информационных систем, А.В.Михалькевич

Содержание

Введение

В условиях рынка все большее число компаний осознают преимущества использования информационных систем (ИС). В некоторых случаях ИС - это не только набор услуг, но и важнейший компонент бизнеса, как, например, система резервирования билетов или средства предоставления финансовой информации. Число интернет пользователей резко увеличивается с каждым годом. По данным на 2016 год мировое число пользователей интернета составляет 3,5 миллиарда человек. Web – место, где проводит время почти каждый второй человек нашей планеты, стало золотой жилой для организации своего бизнеса. Люди стали больше доверять удаленным транзакциям и привыкать совершать платежи через сеть. Так, в 2016 году объем российского рынка электронной торговли составлял 645 млрд. рублей, в 2017 – 1040 млрд., а по прогнозам, в конце 2018 года вырастет еще в 1,5 раза и достигнет отметки в 1570 млрд. рублей. Такая динамика роста говорит о многочисленных плюсах электронных сделок как для потребителей, так и для предлагающих товары и услуги. Порог входа в интернет-бизнес достаточно низкий. Чтобы начать бизнес в интернете, не нужно брать кредит для приобретения помещения и проведения ремонта в нем. Достаточно лишь обратиться в одну из многочисленных web-студий, где можно заказать сайт или взять его в аренду. Чтобы получить выгоду от использования информационной системы, ее следует создавать в короткие сроки и с уменьшенными затратами. Информационная система должна быть легко сопровождаемой и

управляемой. Создание информационной системы аптеки по учету поставок медпрепаратов - достаточно сложный и многоступенчатый процесс, который, весьма часто, содержит фазу информационного моделирования. Информационная модель - это спецификация структуры данных и бизнес правил (правил предметной области). В настоящее время конкурентоспособность аптечного предприятия и повышение рентабельности зависит не только от его месторасположения, но и обеспечивается: внедрением стандартов мерчендайзинга, ориентированных на максимизацию прибыли. разработкой типовых стандартов эффективного взаимодействия с посетителями. дифференциацией ассортиментной политики, ориентацией на специфичных для данной аптеки клиентов. совершенствованием ценовой политики: дифференциация в различных товарных и ценовых сегментах. Актуальность разработки настоящей информационной системы для предметной области «Виртуальная аптека» заключается в необходимости сокращения времени обработки информации и скорости обработки данных. Целью данной работы является проектирование информационной системы для аптеки. В процессе выполнения работы необходимо провести анализ предметной области, продумать назначение информационной системы, приобрести практические навыки по проектированию структуры, разработки и реализации информационной системы.

1 Описание предметной области

Кратко рассмотрим предметную область, задачи которой подлежат автоматизации в ходе выполнения курсового проекта.

Рассматриваемая предметная область – учет информации о поставках медпрепаратов. Основная задача предметной области – данные по поставкам. Подлежащая автоматизации задача – систематизация учета поступлений медпрепаратов в аптеку.

Проведем краткий анализ предметной области, задачи которой подлежат автоматизации.

На сегодняшний день применение баз данных приобрело важное значение для многих организаций, которые для упрощения своей работы применяют компьютерные технологии.

В данном курсовом проекте разработана база данных для информационной поддержки деятельности аптеки с целью автоматизированного ведения данных о лекарствах.

Аптека работает с населением (клиентами). Аптека оказывает услуги: реализация населению лекарственных препаратов. Компания производит однотипные работы, в частности производит продажу лекарственных средств.

Процесс внесения в систему записи о поставке осуществляется следующим образом:

создание записи о поставщике, если такой нет;
создание записи о медпрепарате, если такой нет;
создание записи о поставке с указанием поставщика, медпрепарата и сопутствующих аттрибутов.

Сущность «поставщик» содержит данные о внесенных в систему записях о поставщиках с описанием. Характеризуется следующими атрибутами:

наименование,
адрес,
телефон,
контакты.

В данной сущности хранятся данные о медпрепаратах, продаваемых в аптеке. Характеризуется следующими атрибутами:

наименование,
описание.

В данной сущности хранятся данные о поставках медпрепаратов. Характеризуется следующими атрибутами:

наименование,
дата поставки,
объем поставки,
стоимость партии,
поставщик,
медпрепарат.

Осознание потребности в проекте – поводом для осознания потребности чаще всего необходимость упорядочивания всех записей по организации данных. Постановка целей и задач – определение причин затруднений и ошибок, возникающих при получении данных о поставках. Например, количество людей, которым необходима данная информация может расти, а справочные могут не справиться с наплывом запросов. Необходима интерактивная система.

Выбор поставщика/системы – когда цели и задачи определены, встает вопрос о выборе поставщика услуг автоматизации и ПО. У многих крупных и средних компаний есть давние партнеры (поставщики), которые становятся генеральными подрядчиками и самостоятельно решают, какие третьи компании привлечь для реализации проекта. В данной ситуации оптимальным будет выбор небольшой компании, занятой в сфере разработки ПО – это обусловлено сравнительно малыми масштабами задачи.

Инициирование проекта – бюджет, сроки, структура работ по проекту на этом этапе либо еще не известны, либо сильно размыты. Часто, особенно в крупных организациях процесс подписания договора и предварительной оплаты может длиться не один месяц и чтобы уложиться в отведенные сроки, исполнитель начинает работы по проекту. На этом этапе Заказчик лишний раз может убедиться о надежности исполнителя.

Обследование – подразумевает сбор данных и полный анализ бизнес-процессов, связанных с учетом правонарушений. Этап может быть проведен специалистами в любой из аптек. Реализация проекта – кодирование и сборка подсистемы учета поставок медпрепаратов.

Тестирование и наладка – заключается в поэтапном тестировании разработанной подсистемы и проверке корректности ПО (на предмет соответствия функциональным требованиям).

Развертывание – включает в себя процесс создания инфраструктуры для работы ПО (установка сервера баз данных, сервера приложений, настройка клиентского ПО) и непосредственную установку самого ПО. Проводится тестирование всей системы в целом в реальных условиях.

Сопровождение – включает в себя процесс расширения функциональных возможностей подсистемы и исправления найденных ошибок реализации.

Что касается существующего программного обеспечения для учета правонарушений, то на данный момент такие системы существуют.

Основой предметной области является закупка медпрепаратов. Данная тема лишь отражает коммерческую сторону работы аптеки как и любой торговой точки. Поэтому для моделирования было решено выбрать процесс организации работы новой аптеки (рисунок 1.1-1.4).

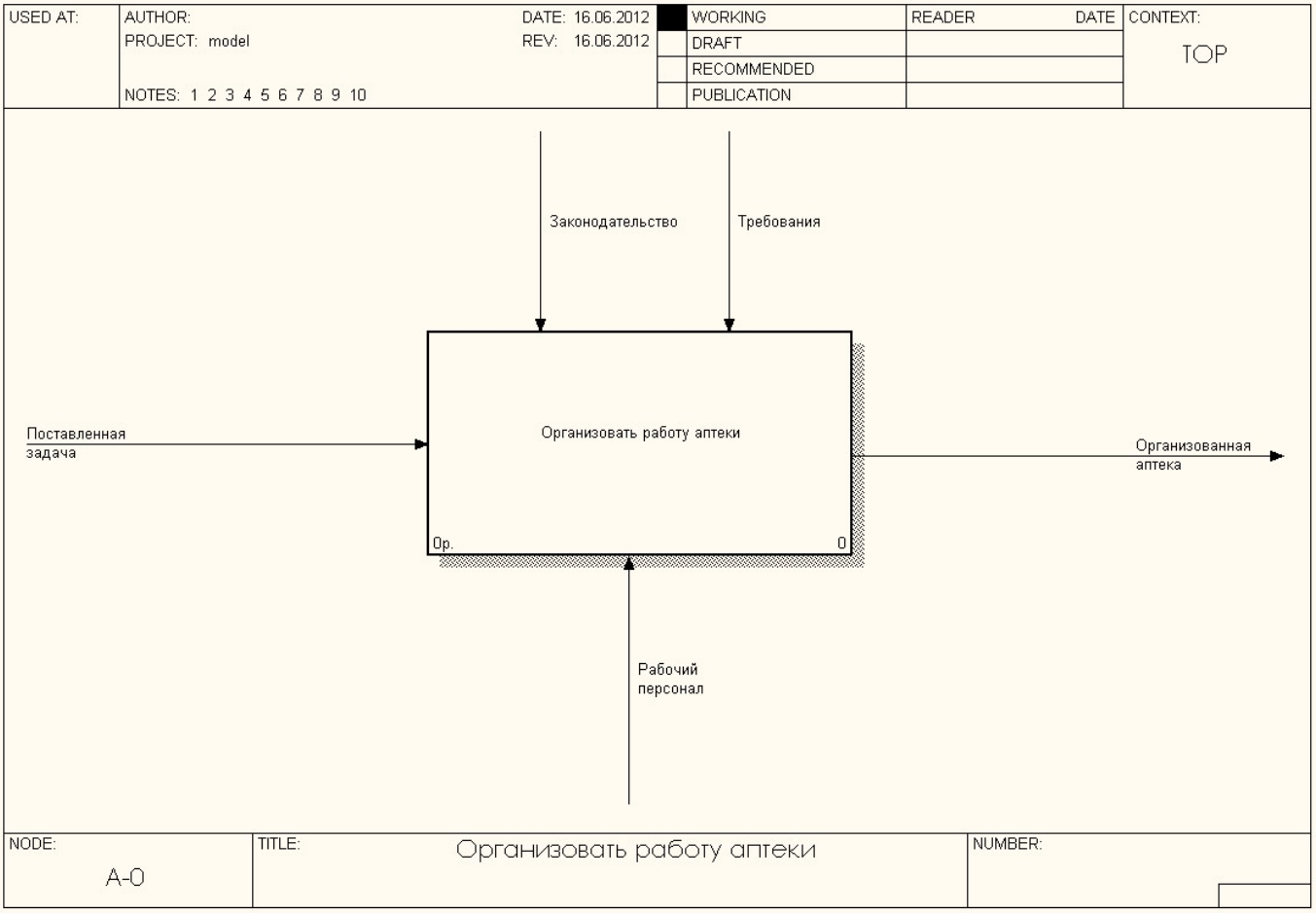

Рисунок 1.1. Уровень A-0

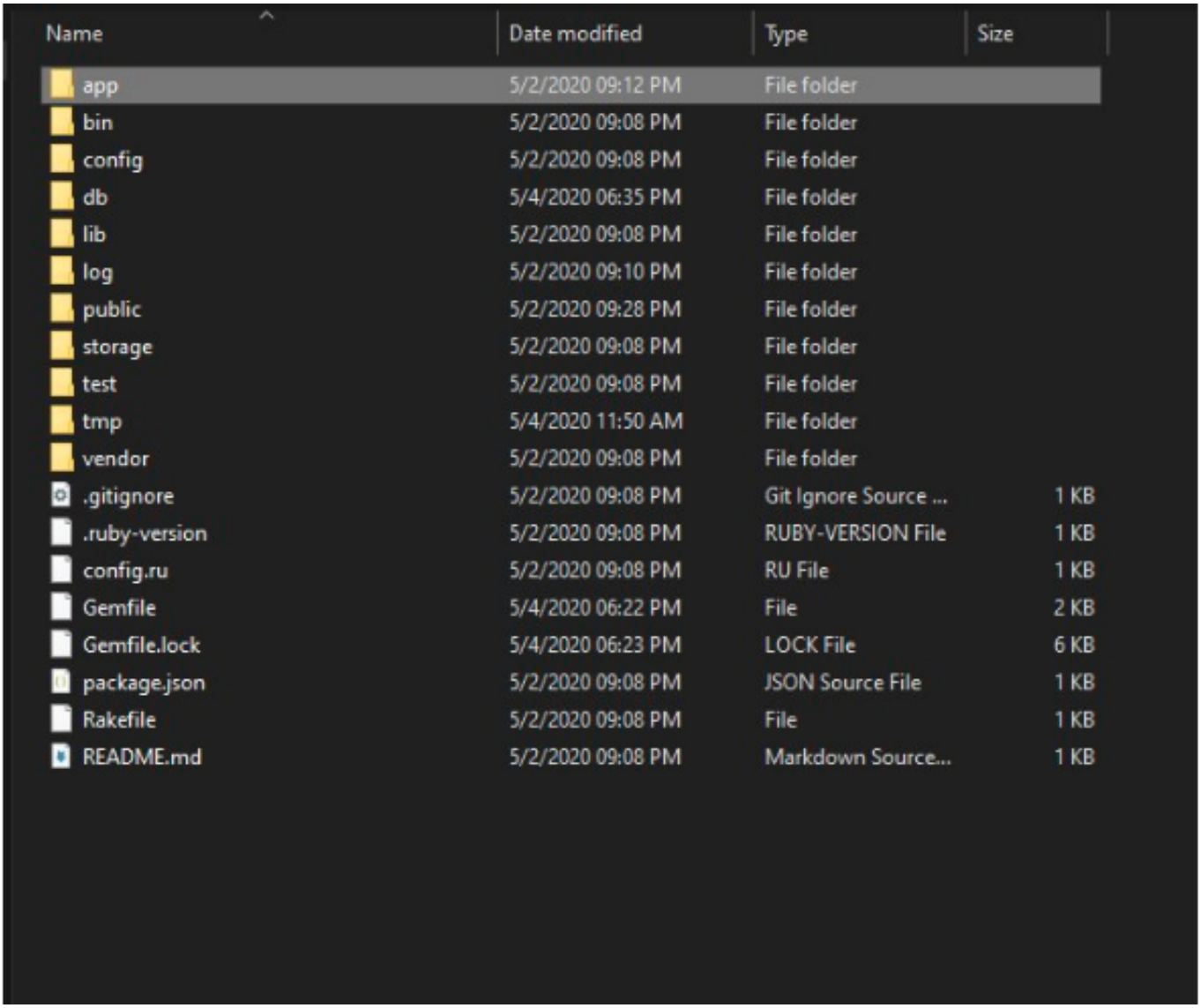

Рисунок 1.2. Уровень A0

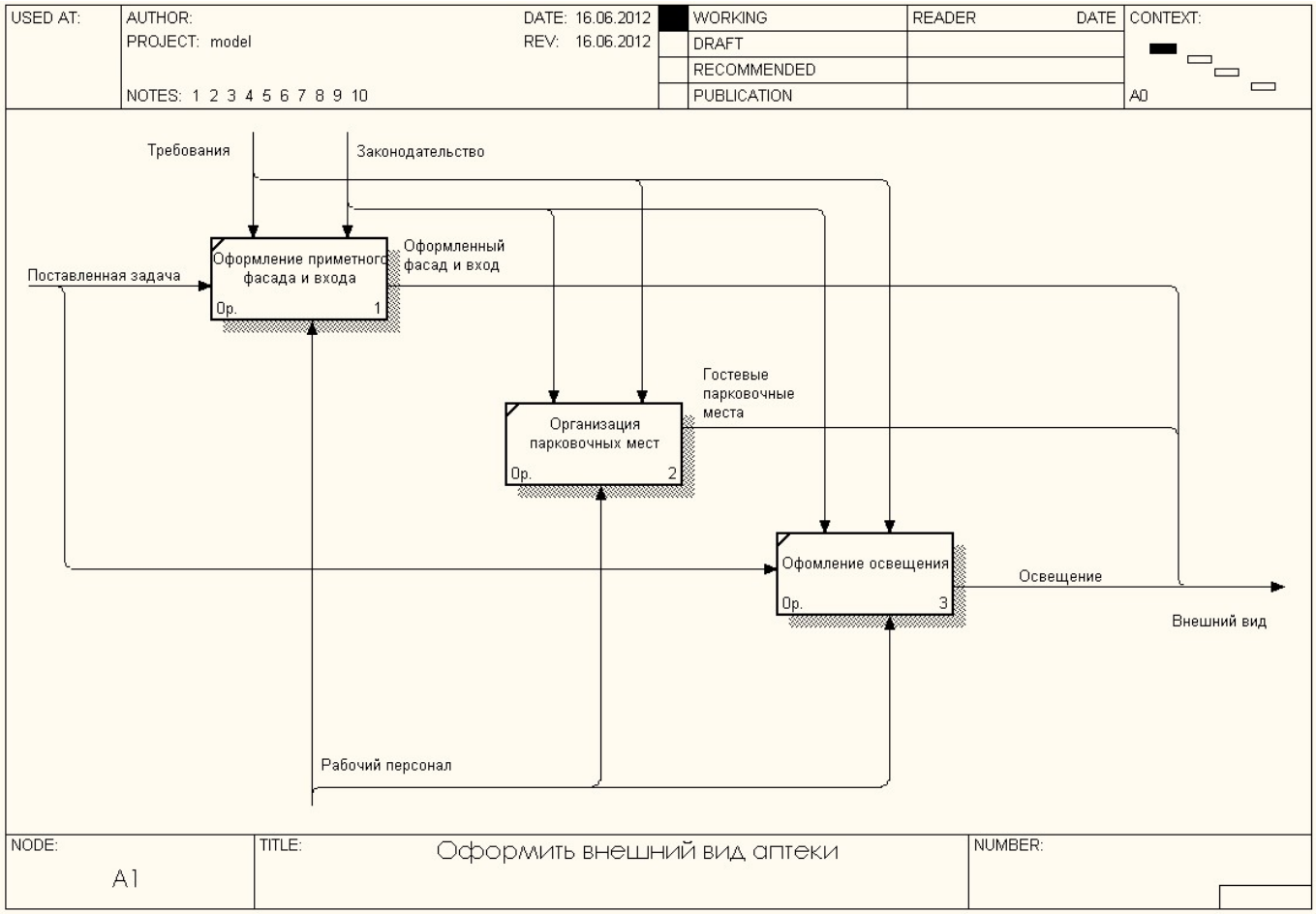

Рисунок 1.3. Уровень A1

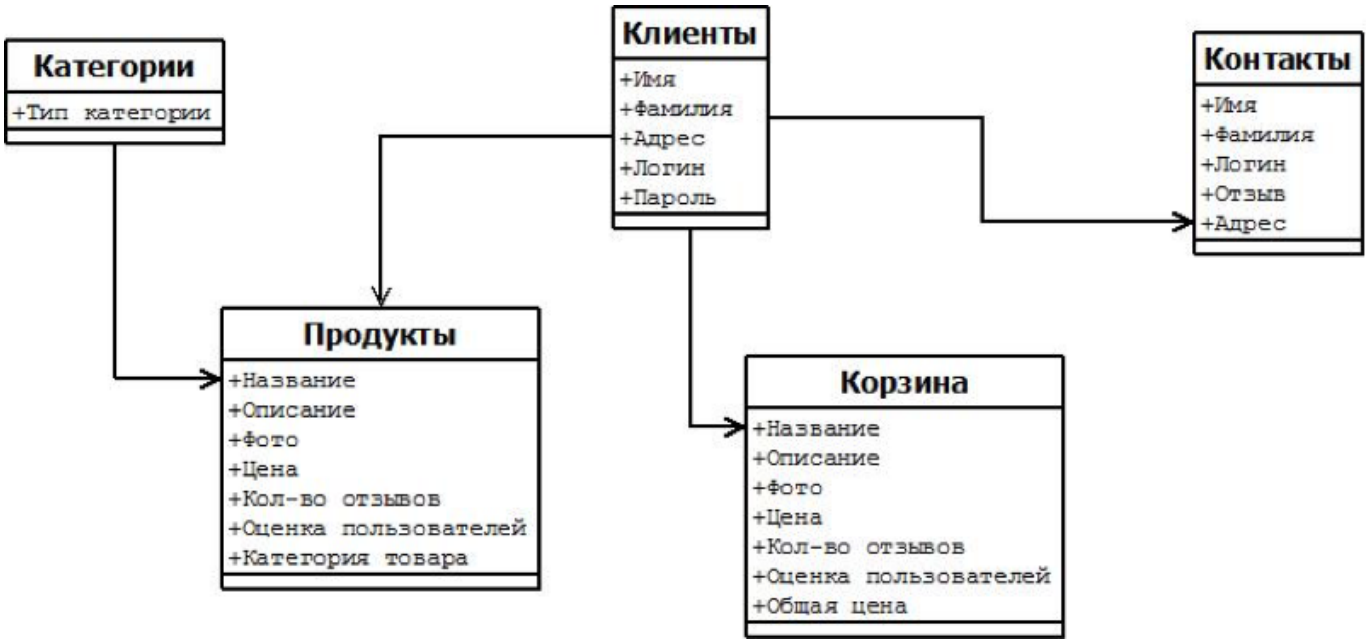

Рисунок 1.4. Уровень A3

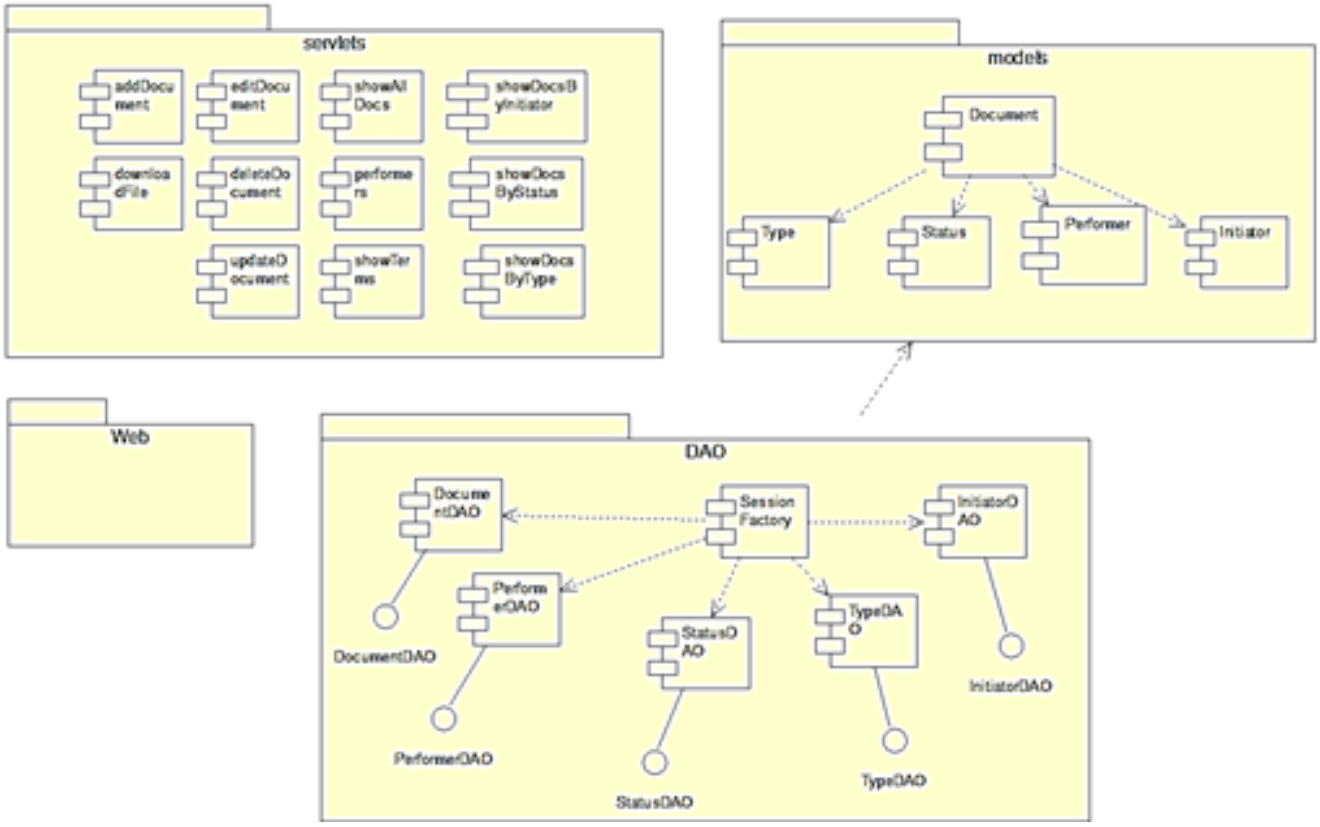

Рисунок 1.5. Уровень A4

2 Постановка задачи и обзор методов ее решения

Постановку задачи определим следующим образом:

выбрать и провести краткий аналитический обзор литературных источников, затрагивающих требуемые для реализации подсистемы учетов поставок медпрепаратов, технологии (JSF, EJB, Sun AppServer, UML и т.п.);
провести анализ предметной области задачи;
разработать методы и модели представления системы учета поставок медпрепаратов в аптеке;
должна быть реализована возможность добавления, удаления и редактирования записей о препаратах;
должна быть реализована возможность добавления, удаления и редактирования записей о поставках;
должна быть предусмотрена возможность сортировки записей о поставщиках;
разработать информационную модель системы (структуру базы данных) и создать базу данных для MySQL;
наполнить разработанную БД соответствующей информацией;
детализировать разработанные ранее модели ПО;
разработать подсистему виртуальной библиотеки;
провести сборку и установку подсистемы учета поставок медпрепаратов, проверить корректность сборки и развертывания;
протестировать программу на предмет соответствия функциональным требованиям с использованием разработанной БД;
описать алгоритмы программных модулей;
описать тестовый пример;
разработать руководство пользователя;

описать полученные результаты.

База данных представляет собой структурированную совокупность данных. Эти данные могут быть любыми - от простого списка предстоящих покупок до перечня экспонатов картинной галереи или огромного количества информации в корпоративной сети. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, необходима система управления базой данных, каковой и является ПО MySQL. Поскольку компьютеры справляются с обработкой больших объемов данных, управление базами данных играет центральную роль в вычислениях. Реализовано такое управление может быть по-разному - как в виде отдельных утилит, так и в виде кода, входящего в состав других приложений.

В реляционной базе данных данные хранятся не все скопом, а в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

ПО с открытым кодом означает, что применять и модифицировать его может любой желающий. Такое ПО можно получать по Internet и использовать бесплатно. При этом каждый пользователь может изучить исходный код и изменить его в соответствии со своими потребностями.

Использование программного обеспечения MySQL регламентируется лицензией GPL (GNU General Public License), http://www.gnu.org/licenses/, в которой указано, что можно и чего нельзя делать с этим программным обеспечением в различных ситуациях. Если работа в рамках GPL вас не устраивает или планируется встраивание MySQL-кода в коммерческое приложение, есть возможность купить коммерческую лицензированную версию у компании MySQL AB.

MySQL является очень быстрым, надежным и легким в использовании. Если вам требуются именно эти качества, попробуйте поработать с данным сервером. MySQL обладает также рядом удобных возможностей, разработанных в тесном контакте с пользователями.

Первоначально сервер MySQL разрабатывался для управления большими базами данных с целью обеспечить более высокую скорость работы по сравнению с существующими на тот момент аналогами. И вот уже в течение нескольких лет данный сервер успешно используется в условиях промышленной эксплуатации с высокими требованиями.

Несмотря на то, что MySQL постоянно совершенствуется, он уже сегодня обеспечивает широкий спектр полезных функций. Благодаря своей доступности, скорости и безопасности MySQL очень хорошо подходит для доступа к базам данных по Internet.

Java - объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения.

Отличительной особенностью Java в сравнении с другими языками программирования

общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.

Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.

Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows версии 3.1, Windows 95, Windows NT, OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой XWindows.

3 Спецификация вариантов использования системы

Пользователю системы предоставляются определенные функциональные возможности, которые указываются в виде диаграммы вариантов использования, реализованной на основе синтаксиса языка UML. Построение программного обеспечения после предварительного моделирования аспектов его работы с помощью графических языков моделирования гораздо проще, чем создание приложения на основе исключительно текстовой документации.

Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
сформулировать общие требования к функциональному поведению проектируемой системы;
разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей
подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

На рисунке 3.1 можно видеть диаграмму вариантов использования предлагаемой к рассмотрению модели.

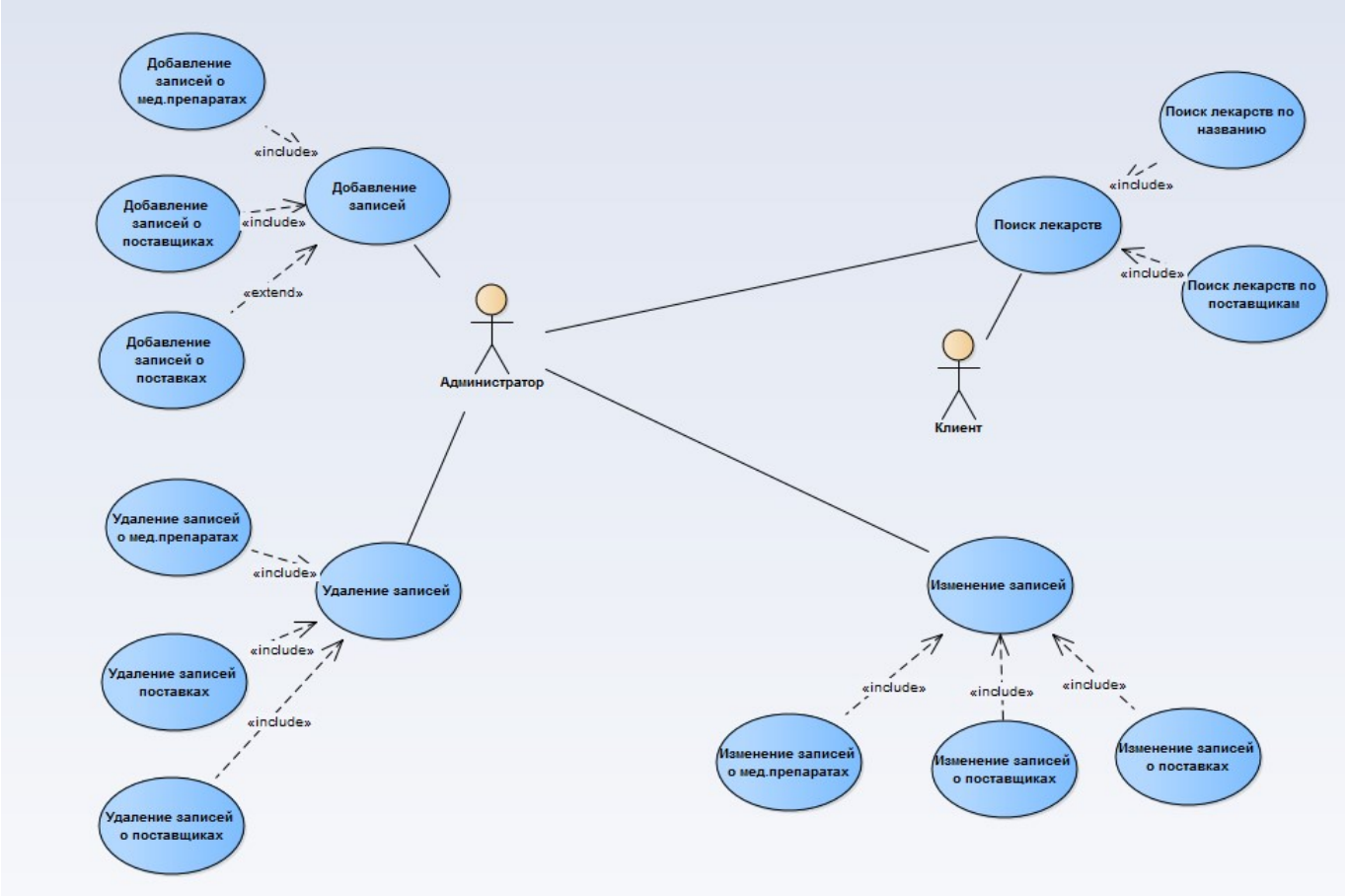

Рисунок 3.1 Диаграмма вариантов использования

Как видно из диаграммы, в системе определен один актера. Работа приложения

заключается в работе с бизнес объектами, представленными типами правонарушений, правонарушениями и штрафами.

4 Модели представления системы

В UML диаграмма классов является типом диаграммы статической структуры. Она описывает структуру системы, показывая её классы, их атрибуты и операторы, а также взаимосвязи этих классов.

Взаимосвязи

Взаимосвязь — это особый тип логических отношений между сущностями, показанных на диаграммах классов и объектов. В UML’е представлены следующие виды отношений:

Ассоциации

Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности.

Существует пять различных типов ассоциации. Наиболее распространёнными являются двунаправленная и однонаправленная. Например, классы «рейс» и «самолёт» связаны двунаправленной ассоциацией, а классы «человек» и «кофейный автомат» связаны однонаправленной.

Двойные ассоциации (с двумя концами) представляются линией, соединяющей два классовых блока. Ассоциации более высокой степени имеют более двух концов и представляются линиями, один конец которых идет к классовому блоку, а другой к общему ромбику. В представлении однонаправленной ассоциации добавляется стрелка, указывающая на направление ассоциации.

Ассоциация может быть именованной, и тогда на концах представляющей её линии будут подписаны роли, принадлежности, индикаторы, мультипликаторы, видимости или другие свойства.

Агрегация

Диаграмма классов, показывающая Агрегацию между двумя классами

Агрегация — это разновидность ассоциации при отношении между целым и его частями. Как тип ассоциации агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).

Агрегация встречается, когда один класс является коллекцией или контейнером других. Причём по умолчанию, агрегацией называют агрегацию по ссылке, то есть когда время существования содержащихся классов не зависит от времени существования содержащего их класса. Если контейнер будет уничтожен, то его содержимое — нет.

Графически агрегация представляется пустым ромбиком на блоке класса и линией, идущей от этого ромбика к содержащемуся классу.

Композиция

Композиция — более строгий вариант агрегации. Известна также как агрегация по значению.

Композиция имеет жёсткую зависимость времени существования экземпляров класса

контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

Графически представляется как и агрегация, но с закрашенным ромбиком.

На рисунке 4.1 представлена диаграмма классов рассматриваемой модели.

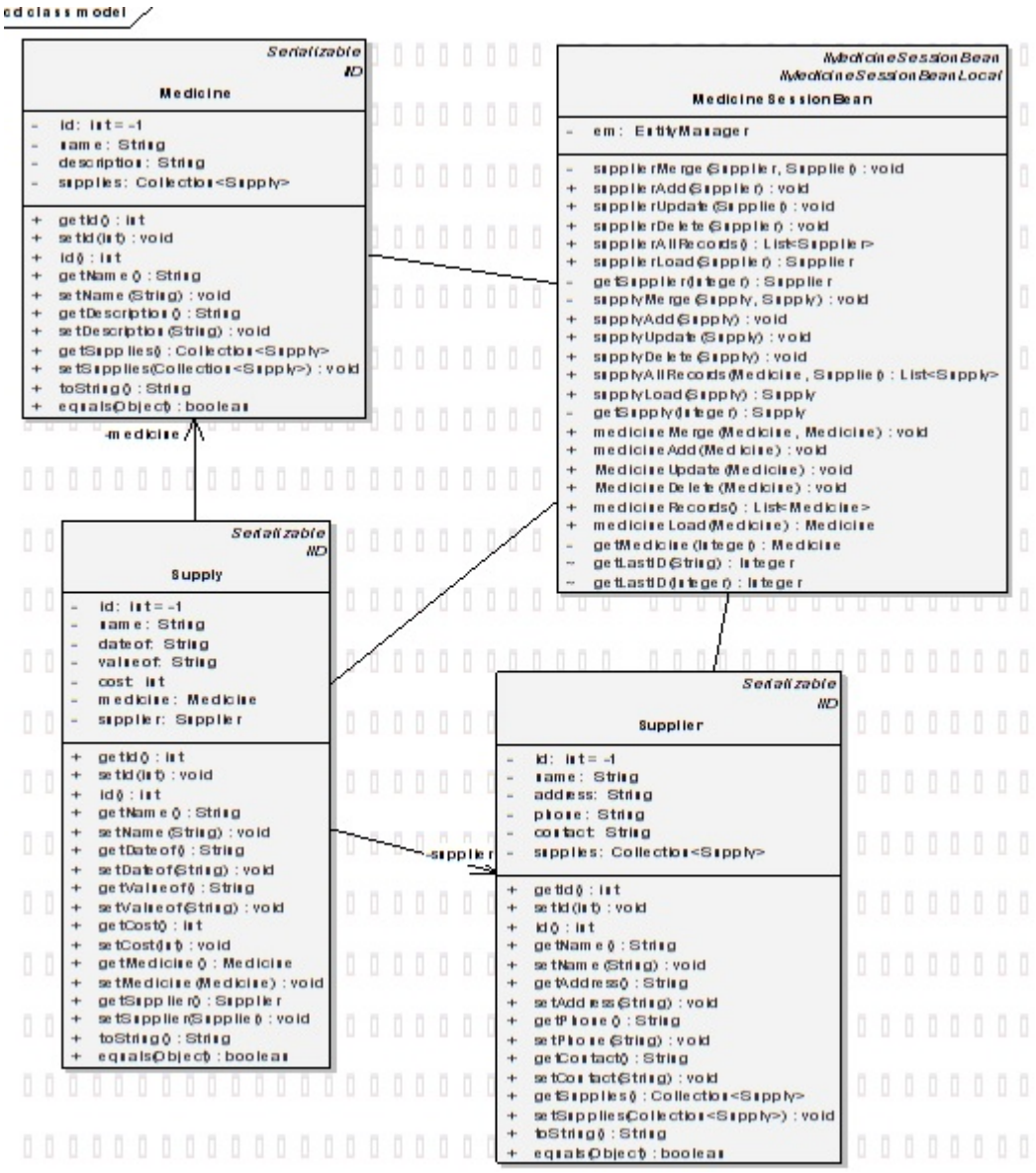

Рисунок 4.1 - Диаграмма классов

На данной диаграмме показаны классы ejb-бинов. Обратимся теперь к диаграмме компонентов, представленной на рисунке 4.2. Диаграмма компонентов, в отличие от других диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

визуализации общей структуры исходного кода программной системы;
спецификации исполнимого варианта программной системы;
обеспечения многократного использования отдельных фрагментов программного кода;
представления концептуальной и физической схем баз данных.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие - на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

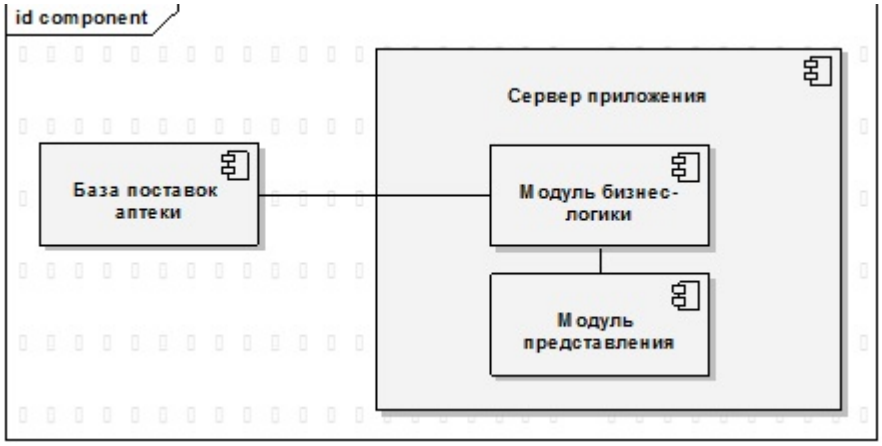

Рисунок 4.2 - Диаграмма компонентов

Взаимодействие этих компонентов иллюстрирует диаграмма последовательностей, изображенная на рисунке 4.3. Диаграмма последовательности (Sequence Diagram)

Удобное средство для обозначения очередности следования друг за другом различных стимулов (сообщений), с помощью которых объекты взаимодействуют между собой. Например, когда нужно проработать буквально по шагам какой-то очень важный участок выполнения программы.

Главный акцент - порядок и динамика поведения, т.е. как и в каком порядке происходят события.

Отличие от диаграммы классов заключается в том, что диаграмма классов дает статическую картинку, т.е. описание которое не меняется во время выполнения программы. Отличие от диаграммы коммуникаций (или как она раньше называлась colaboration):

Диаграмма последовательности фокусирует наше внимание на очередности выполнения по времени, а диаграмма коммуникаций - на составляющих элементах.

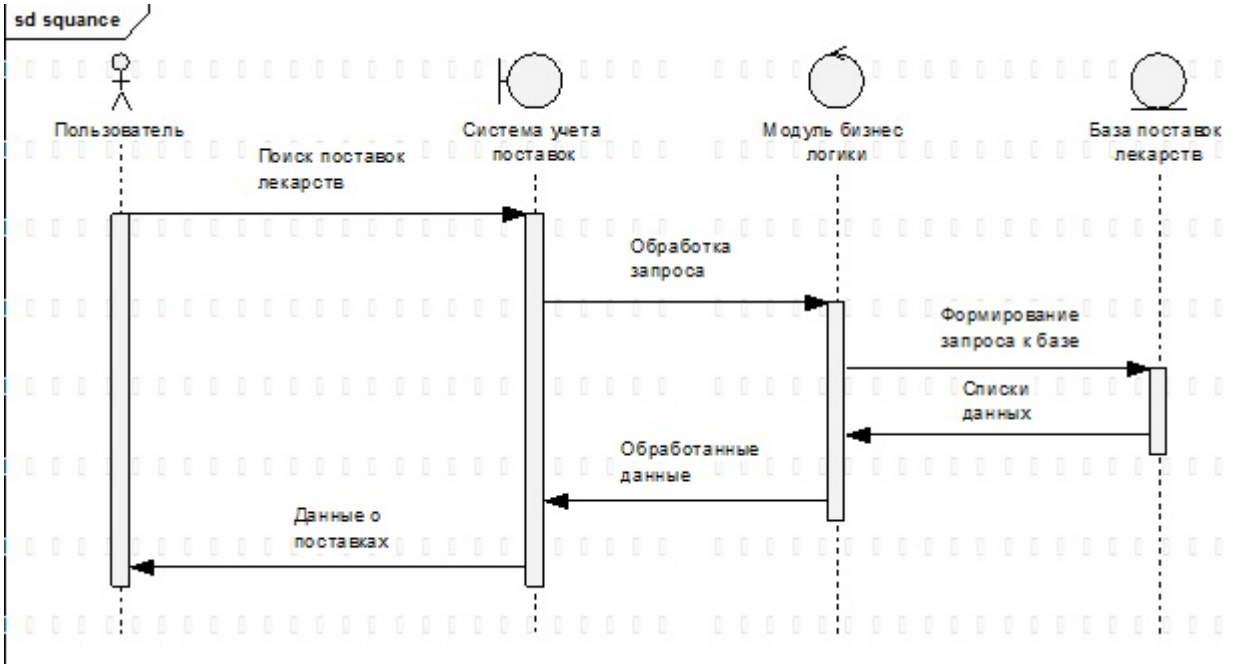

Рисунок 4.3-Диаграмма последовательностей

Диаграмма была построена на основании следующих соображений:

пользователь при помощи интерфейса (экранная форма, стереотип boundary) просит загрузить список перевозок;
jsf-компоненты обращается к объявленным методам session-бина;
session-бин обращается к ejb для выборки данных;
после получения результатов запроса из БД сервер возвращает результат выполнения метода клиента;
экранная форма выполняет действия по обновлению в соответствии с полученным результатом.

Каждая диаграмма состояний в UML описывает все возможные состояния одного экземпляра определенного класса и возможные последовательности его переходов из одного состояния в другое, то есть моделирует все изменения состояний объекта как его реакцию на внешние воздействия.

Диаграммы состояний чаще всего используются для описания поведения отдельных объектов, но также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний является графом специального вида, который представляет некоторый автомат.

Вершинами графа являются возможные состояния автомата, изображаемые соответствующими графическими символами, а дуги обозначают его переходы из состояния в состояние.

Диаграммы состояний могут быть вложены друг в друга для более детального представления отдельных элементов модели.

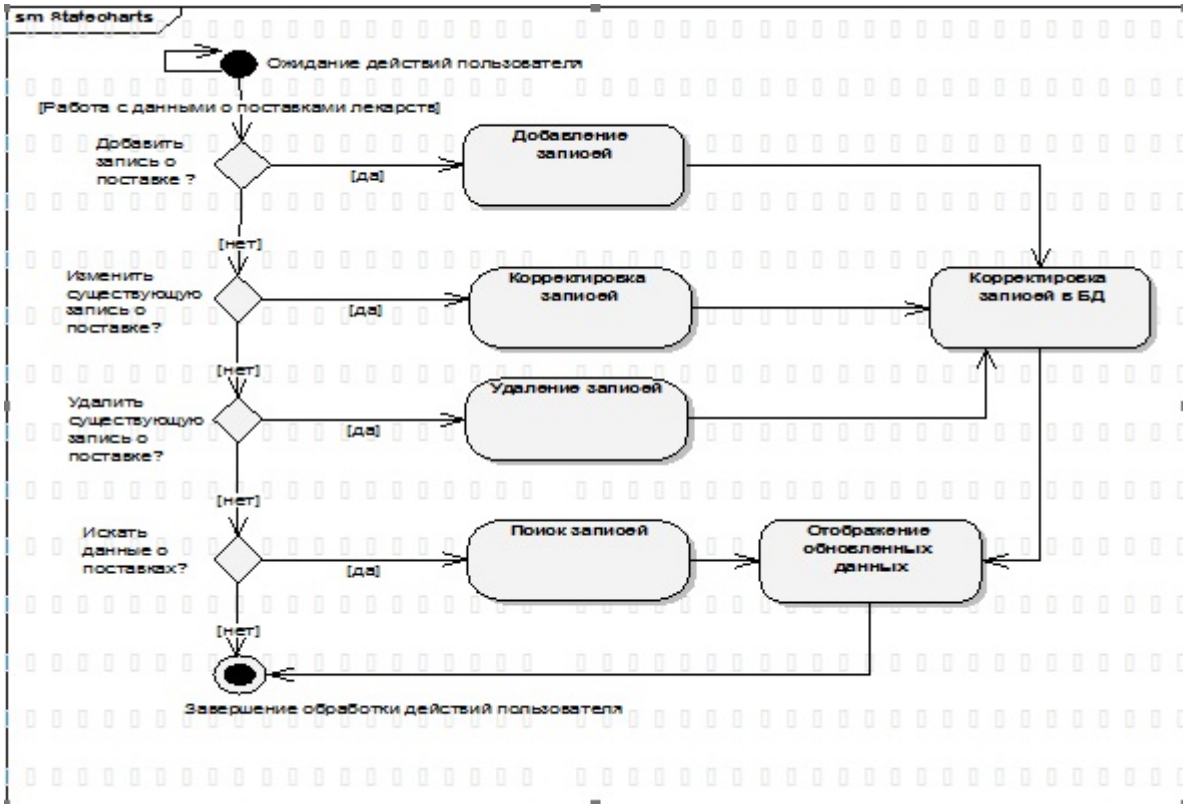

Рисунок 4.4 - Диаграмма состояний

Последняя представляющая интерес диаграмма – диаграмма развертывания, изображенная на рисунке 4.5.

Диаграмма развёртывания, Deployment diagram в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

Узлы представляются как прямоугольные параллелепипеды с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь подузлы, которые представляются как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

Существует два типа узлов:

узел устройства;
узел среды выполнения.

Узлы устройств — это физические вычислительные ресурсы со своей памятью и сервисами для выполнения программного обеспечения, такие как обычные ПК, мобильные телефоны. Узел среды выполнения — это программный вычислительный ресурс, который работает внутри внешнего узла и который предоставляет собой сервис, выполняющий другие исполняемые программные элементы.

Система, в самом общем случае, распределена по 2 узлам – это сервер базы данных MySQL,

сервер JBoss, на котором установлено приложение.

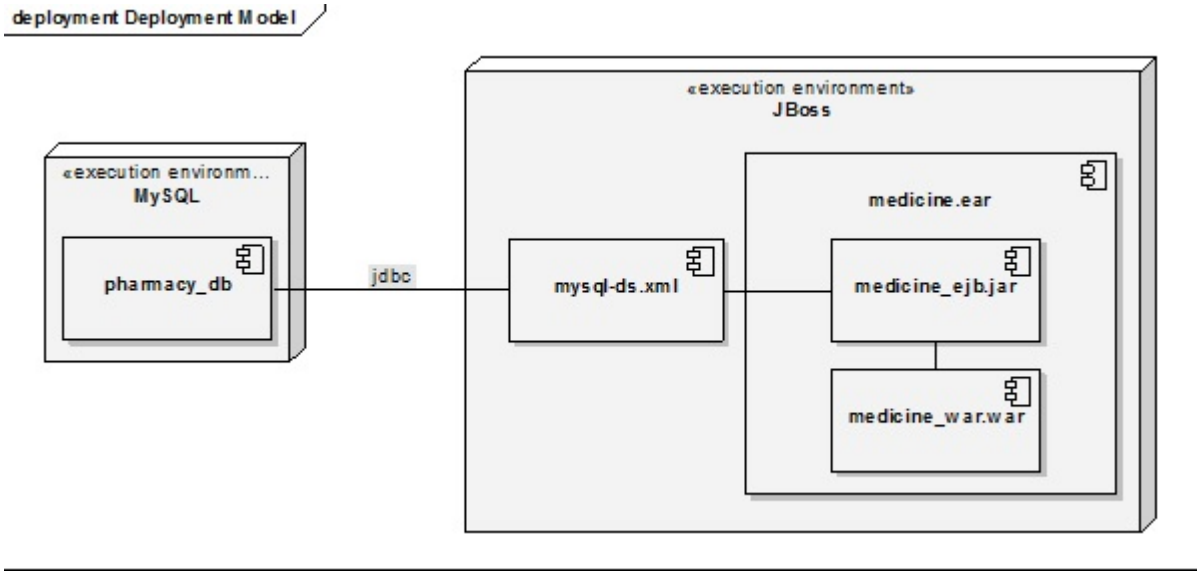

Рисунок 4.5 - Диаграмма развертывания

5 Информационная модель системы

При разработке информационной модели использовалось CASE-средство Erwin. ERwin предназначен в основном для разработчиков, проектировщиков БД, системных аналитиков. Функциональность ERwin делает его также незаменимым инструментом для администраторов БД и руководителей проектов. Руководители проектов могут с помощью ERwin тщательно задокументировать структуру БД, получить отчеты презентационного качества и обеспечить эффективное управление проектом, используя интеграцию Erwin со специализированным средством организации коллективной работы – CA ModelMart. Поскольку ERwin поддерживает работу с БД на физическом уровне, учитывая особенности каждой конкретной СУБД, администраторы БД могут с его помощью максимально повысить производительность информационной системы. Разработчики с помощью ERwin могут сначала, используя визуальные средства, описать схему БД, а затем автоматически сгенерировать файлы данных для выбранной реляционной СУБД (прямое проектирование). Автоматически генерируются также триггеры, обеспечивающие ссылочную целостность БД. Поддерживаются хранимые процедуры. ERwin поддерживает нотации IDEF1X, IE и DIMENSIONAL. Пользователь описывает структуру данных визуально. Он задает служащие прообразами реляционных таблиц сущности с их атрибутами и при помощи мыши "натягивает" между ними связи, которые являются прототипами реляционных отношений.

Возможна также обратная разработка. ERwin позволяет по уже существующим файлам БД восстанавливать логическую структуру данных. Это называется обратным проектированием (reverse engineering). Оно позволяет, во-первых, переносить структуру БД из одной СУБД в другую и, во-вторых, исследовать старые проекты. Этот процесс наиболее распространен в процессе перехода с одной технологии на другую (с файл-сервер на клиент-сервер), а также при смене сервера БД. На основе модели данных предоставляется возможность создавать отчеты, которые позволяют существенно упростить процесс документирования технического

проекта.

ERwin поддерживает прямое и обратное проектирование более 20 типов баз данных различных производителей, от настольных БД до реляционных СУБД и специализированных СУБД, предназначенных для создания информационных хранилищ.

На рисунках 5.1 и 5.2 можно видеть физическое и логическое представление разработанной модели данных. Рассмотрены сущности предметной области. Рассмотрим каждую по отдельности.

Поставщик (Supplier) характеризуется следующими атрибутами:

sid – уникальный ключ типа поставщика
name – наименование
address – адрес
contact – контактные данные
phone – телефон

Мед. препарат (Medicine) характеризуется следующими атрибутами:

mid – уникальный ключ мед. препарата
name – наименование
description – примечание

Поставка (Supply) характеризуется следующими атрибутами:

spid – уникальный ключ поставки
name – наименование
dateof – дата поставки
valueof – закупленный объем
cost – стоимость партии
sid - родительский ключ поставщика
mid – ключ родительской записи мед. препарата.

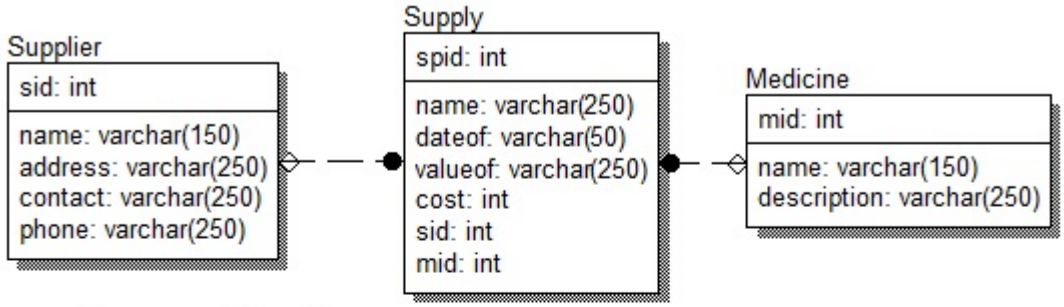

Рисунок 5.1- Физическое представление модели данных

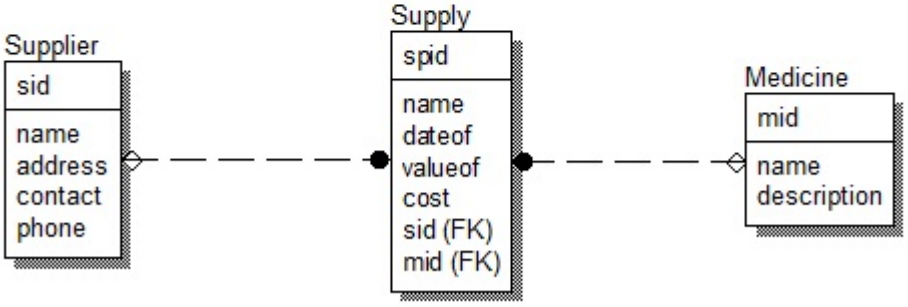

Рисунок 5.2 - Логическое представление модели данных

В модели присутствуют 2 идентифицирующих связи «один-ко-многим». Данные связи были выбраны по следующим соображениям:

- в системе регистрируются правонарушения различных типов
- на каждое правонарушение регистрируется назначенный штраф.

6 Обоснование решений по использованию программных средств, не включенных в требования

В данном разделе рассматриваются использованные в курсовой работе программные и технические средства с точки зрения эффективности и удобства их использования.

Обычно разработка модели базы данных состоит из двух этапов: составление логической модели и создание на ее основе физической модели. ERwin полностью поддерживает такой процесс, он имеет два представления модели: логическое (logical) и физическое (physical). Таким образом, разработчик может строить логическую модель базы данных, не задумываясь над деталями физической реализации, т.е. уделяя основное внимание требованиям к информации и бизнес-процессам, которые будет поддерживать будущая база данных. ERwin имеет очень удобный пользовательский интерфейс, позволяющий представить базу данных в самых различных аспектах.

Например, ERwin имеет такие средства визуализации как "хранимое представление" (stored display) и "предметная область" (subject area). Хранимые представления позволяют иметь несколько вариантов представления модели, в каждом из которых могут быть подчеркнуты определенные детали, которые вызвали бы перенасыщение модели, если бы они были помещены на одном представлении. Предметные области помогают вычленить из сложной и трудной для восприятия модели отдельные фрагменты, которые относятся лишь к определенной области, из числа тех, что охватывает информационная модель. Интерфейс среды разработки ERwin представлен на рисунке.

Возможности редактирования и визуализации в среде ERwin весьма широки, так, например, создание отношений возможно при помощи перетаскивания атрибута из одной сущности в другую. Такое редактирование модели позволяет вносить изменения и проводить нормализацию быстрее и эффективнее, чем с использованием других инструментов. Для того, чтобы добавить новый элемент на диаграмму, его просто нужно выбрать на панели инструментов (Toolbox) и перенести в нужное место диаграммы. Добавив новую сущность на диаграмму, в нее можно добавить атрибуты, не открывая никаких редакторов, а просто ввести их названия прямо на диаграмме.

Таким образом, ERwin позволяет значительно снизить время на создание самой диаграммы и сконцентрироваться на самих задачах, стоящих перед разработчиком.

ERwin имеет мощные средства визуализации модели, такие, как использование различных шрифтов, цветов и отображение модели на различных уровнях, например, на уровне описания сущности, на уровне первичных ключей сущности и т.д. Эти средства ERwin значительно помогают при презентации модели в кругу разработчиков системы или сторонним лицам.

Возможность использования модели ERwin одновременно для логического и физического представления данных позволяет по окончании работы получить полностью документированную модель.ERwin, как и инструмент моделирования бизнес-процессов BPwin, интегрирован с генератором отчетов фирмы Logic Works - RPTwin. Это средство позволяет получать подробные отчеты по модели, освещая самые различные ракурсы и аспекты. Инструмент RPTwin поставляется вместе с ERwin и имеет богатый набор встроенных отчетов, позволяющих получать многогранную информацию по модели. Документирование структуры данных является очень важной частью моделирования, т.к. это позволяет другим разработчикам или лицам, которые будут сопровождать систему, быстрее начать ориентироваться во внутренней структуре и понимать назначение компонентов.

Пакет BPwin мощный инструмент моделирования, который используется для анализа, документирования и реорганизации сложных бизнес-процессов. Модель, созданная средствами BPwin, позволяет четко документировать различные аспекты деятельности - действия, которые необходимо предпринять, способы их осуществления, требующиеся для этого ресурсы и др. Таким образом, формируется целостная картина деятельности предприятия - от моделей организации работы в маленьких отделах до сложных иерархических структур. При разработке или закупке программного обеспечения модели бизнес-процессов служат прекрасным средством документирования потребностей, помогая обеспечить высокую эффективность инвестиций в сферу IT. В руках же системных аналитиков и разработчиков BPwin - еще и мощное средство моделирования процессов при создании корпоративных информационных систем (КИС).

Модели BPwin дают основу для осмысления бизнес-процессов и оценки влияния тех или иных событий, а также описывают взаимодействие процессов и потоков информации в организации. Неэффективная, высокозатратная или избыточная деятельность может быть легко выявлена и, следовательно, усовершенствована, изменена или устранена в соответствии с общими целями организации.

Внешние обстоятельства зачастую вынуждают вносить изменения в деятельность организации. Последствия этих изменений должны быть тщательно изучены и осмыслены перед тем, как система будет переделана с их учетом. BPwin может помочь пользователю на протяжении всего цикла, предоставив возможность оптимизировать бизнес-процесс, которого коснутся эти изменения.

С помощью BPwin пользователь может сделать свою работу более продуктивной. Действия и другие объекты создаются буквально несколькими щелчками мыши, а затем легко отбуксированы в нужное место. Интерфейс BPwin, выполненный в стиле "проводника" облегчает навигацию и редактирование сложных процессов с иерархической структурой. Развитые возможности изменения масштаба представления позволяют быстро найти и сосредоточиться на необходимой для работы части модели процесса.

BPwin позволяет:

Обеспечить эффективность операций, рассматривая текущие бизнес-операции через мощные инструменты моделирования.
Совершенствовать бизнес-процессы, формулируя и определяя альтернативные реакции

на воздействия рынка.
Быстро исключать непродуктивные операции, легко и интуитивно сопоставляя операционные изменения. Неэффективные, неэкономичные или избыточные операции могут быть легко выявлены и, следовательно, улучшены, изменены или вовсе исключены - в соответствии с целями компании.

В качестве целевой СУБД в данном курсовом проекте был взят MySQL. MySQL. MySQL является собственностью компании Sun Microsystems, осуществляющей разработку и поддержку приложения. Распространяется под GNU General Public License и под собственной коммерческой лицензией, на выбор. Помимо этого разработчики создают функциональность по заказу лицензионных пользователей, именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в LAMP. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

IntelliJ IDEA – это ведущая и, по мнению многих отраслевых экспертов и Java-разработчиков, просто лучшая на сегодняшний день среда быстрой разработки на языке Java.

IntelliJ IDEA представляет собой высокотехнологичный комплекс тесно интегрированных инструментов программирования, включающий интеллектуальный редактор исходных текстов с развитыми средствами автоматизации, мощные инструменты рефакторинга кода, встроенную поддержку технологий J2EE, механизмы интеграции со средой тестирования Ant/JUnit и системами управления версиями, уникальный инструмент оптимизации и проверки кода Code Inspection, а также инновационный визуальный конструктор графических интерфейсов.

Уникальные возможности IDEA избавляют программиста от груза рутинной работы, помогают своевременно устранить ошибки и повысить качество кода, поднимая продуктивность разработчика на новую высоту.

JavaServer Faces (JSF) — это фреймворк для веб-приложений, написанный на Java. Он служит для того, чтобы облегчать разработку пользовательских интерфейсов для Java EE приложений. В отличие от прочих MVC фреймворков, которые управляются запросами, подход JSF основывается на использовании компонентов. Состояние компонентов пользовательского интерфейса сохраняется, когда пользователь запрашивает новую страницу и затем восстанавливается, если запрос повторяется. Для отображения данных обычно используется JSP, Facelets, но JSF можно приспособить и под другие технологии, например XUL.

Технология JavaServer Faces включает:

Набор API для представления компонент пользовательского интерфейса (UI) и управления их состоянием, обработкой событий и валидацией вводимой информации, определения

навигации, а также поддержку интернационализации (i18n) и доступности (accessibility).

Специальная библиотека JSP тегов для выражения интерфейса JSF на JSP странице.

Призванная быть гибкой, технология JavaServer Faces усиливает существующие, стандартные концепции пользовательского интерфейса (UI) и концепции Web-уровня без привязки разработчика к конкретному языку разметки, протоколу или клиентскому устройству. Классы компонентов пользовательского интерфейса, поставляемые вместе с технологией JavaServer Faces, содержат функциональность компонент, а не специфичное для клиента отображение, открывая тем самым возможность рендеринга JSF-компонент на различных клиентских устройствах.

Совмещая функциональность компонент интерфейса пользователя со специальными рендерерами, разработчики могут конструировать специальные теги для заданного клиентского устройства.

В качестве удобства технология JSF предоставляет специфичный рендерер и специальную библиотеку JSP-тегов для рендеринга на HTML-клиенте, позволяя разработчикам приложений на J2EE платформе использовать технологию JSF в своих приложениях.

7 Описание алгоритмов

Алгоритм — набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное время. В старой трактовке вместо слова «порядок» использовалось слово «последовательность», но по мере развития параллельности в работе компьютеров слово «последовательность» стали заменять более общим словом «порядок». Это связано с тем, что работа каких-то инструкций алгоритма может быть зависима от других инструкций или результатов их работы. Таким образом, некоторые инструкции должны выполняться строго после завершения работы инструкций, от которых они зависят. Независимые инструкции или инструкции, ставшие независимыми из-за завершения работы инструкций, от которых они зависят, могут выполняться в произвольном порядке, параллельно или одновременно, если это позволяют используемые процессор и операционная система.

Ранее часто писали «алгорифм», сейчас такое написание используется редко, но, тем не менее, имеет место (например, Нормальный алгорифм Маркова).

Часто в качестве исполнителя выступает некоторый механизм (компьютер, токарный станок, швейная машина), но понятие алгоритма необязательно относится к компьютерным программам, так, например, чётко описанный рецепт приготовления блюда также является алгоритмом, в таком случае исполнителем является человек.

Понятие алгоритма относится к первоначальным, основным, базисным понятиям математики. Вычислительные процессы алгоритмического характера (арифметические действия над целыми числами, нахождение наибольшего общего делителя двух чисел и т. д.) известны человечеству с глубокой древности. Однако, в явном виде понятие алгоритма сформировалось лишь в начале XX века.

Частичная формализация понятия алгоритма началась с попыток решения проблемы разрешения (нем. Entscheidungsproblem), которую сформулировал Давид Гильберт в 1928 году.

Следующие этапы формализации были необходимы для определения эффективных вычислений или «эффективного метода»; среди таких формализаций — рекурсивные функции Геделя — Эрбрана — Клини 1930, 1934 и 1935 гг., λ-исчисление Алонзо Чёрча 1936 г., «Формулировка 1» Эмиля Поста 1936 года и машина Тьюринга. В методологии алгоритм является базисным понятием и получает качественно новое понятие как оптимальности по мере приближения к прогнозируемому абсолюту. В современном мире алгоритм в формализованном выражении составляет основу образования на примерах, по подобию.

На основе сходства алгоритмов различных сфер деятельности была сформирована концепция (теория) экспертных систем.

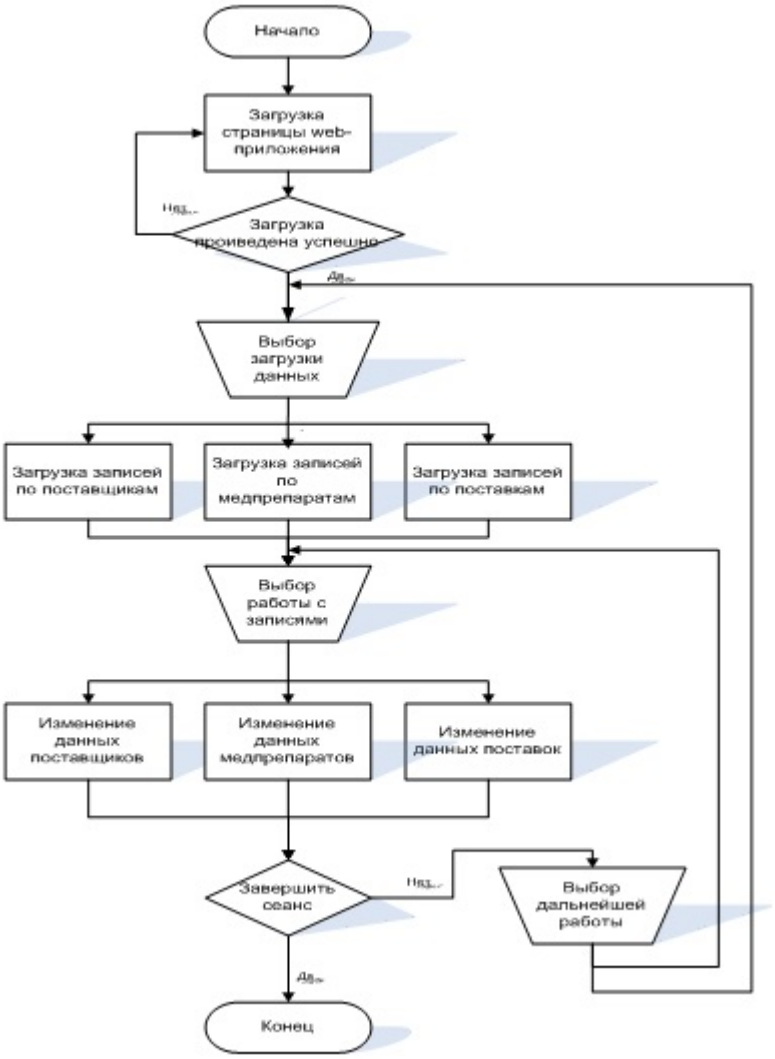

Рисунок 7.1- Обобщенный алгоритм работы

Алгоритм обработки действий пользователя:

пользователь вводит в браузере адрес страницы или нажимает на гиперссылку;
браузер посылает запрос на сервер;
в случае доступности сервера вызывается соответствующая JSF-страница;
страница обращается к серверным компонентам бизнес-уровня, отвечающим за работу с данными;
страница отображает результаты обработки данных (поиск, результаты удаления и т.п.);
клиент просматривает результаты действия в окне браузера.

8 Руководство пользователя

Для развертывания системы необходим следующий Environment:

Jdk 1.6;
MySQL 5;
Jboss.

Развертывание состоит из 3 этапов:

запуск базы данных
настройка сервера для подключения к базе данных
разворачивание приложения.

База разворачивается при помощи скрипта, поставляемого с приложением.

JDBC resorce для работы JBoss-a создается при помощи mysql-ds.xml, который следует положить в JBOSS_HOME \server\default\deploy\

После того, как настроено подключение к базе данных, необходимо развернуть приложение. Для этого нужно положить ear-файл в JBOSS_HOME \server\default\deploy\.

Приложение готово к работе, стартовая страница будет доступна по адресу http://<host>:<port>/medicine/index.jsf.

На главной странице отображена информация, занесенная в систему о поставщиках, мед. препаратах и их поставках.

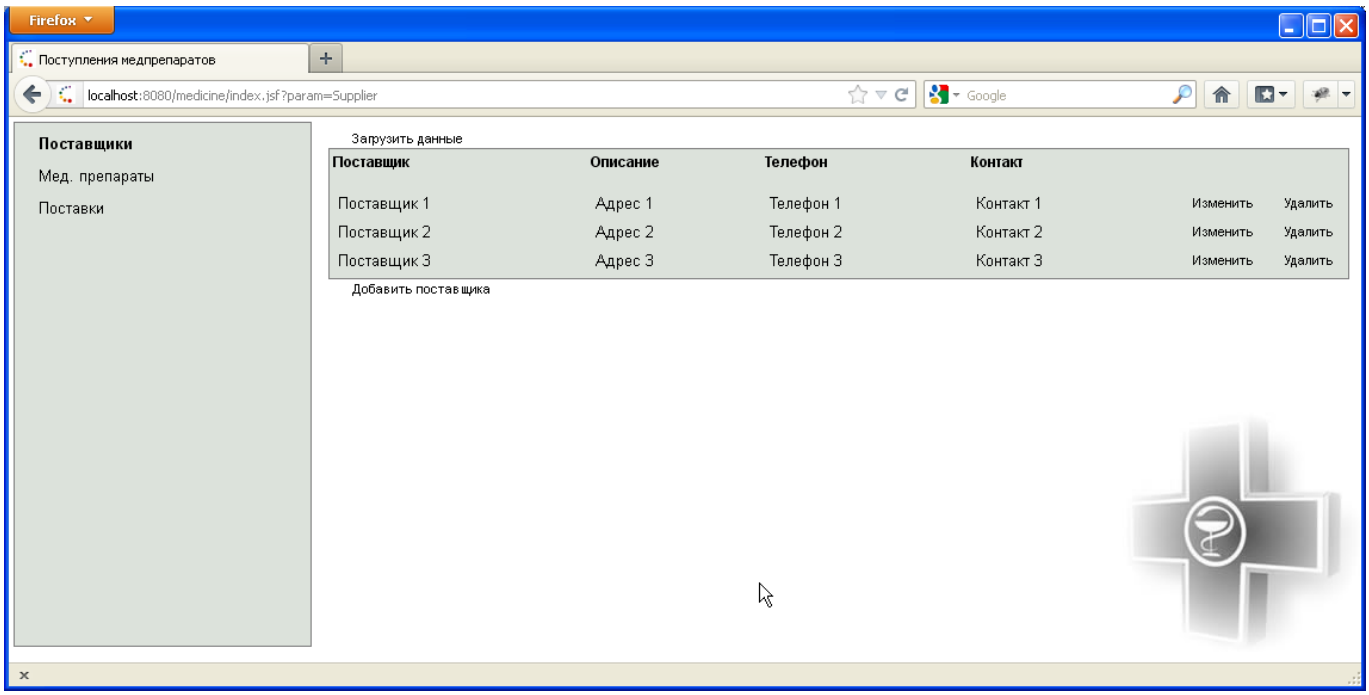

Рисунок 8.1- Страница работы с записями о поставщиках

Пользователю доступна стартовая страница, на которой слева отображаются ссылки для работы с основными сущностями бизнес-логики. Возле записей предоставлены ссылки управления данными.

Пользователь может редактировать данные о поставщиках, мед. препаратах, поставках. Чтобы добавить новую запись о поставщике нужно нажать на ссылку внизу таблицы со списком данных «Добавить тип поставщика». Если нужно отредактировать данные, то напротив каждой

записи есть соответствующая ссылка.

При этом показывается форма редактирования, приведенная на рисунке 8.2.

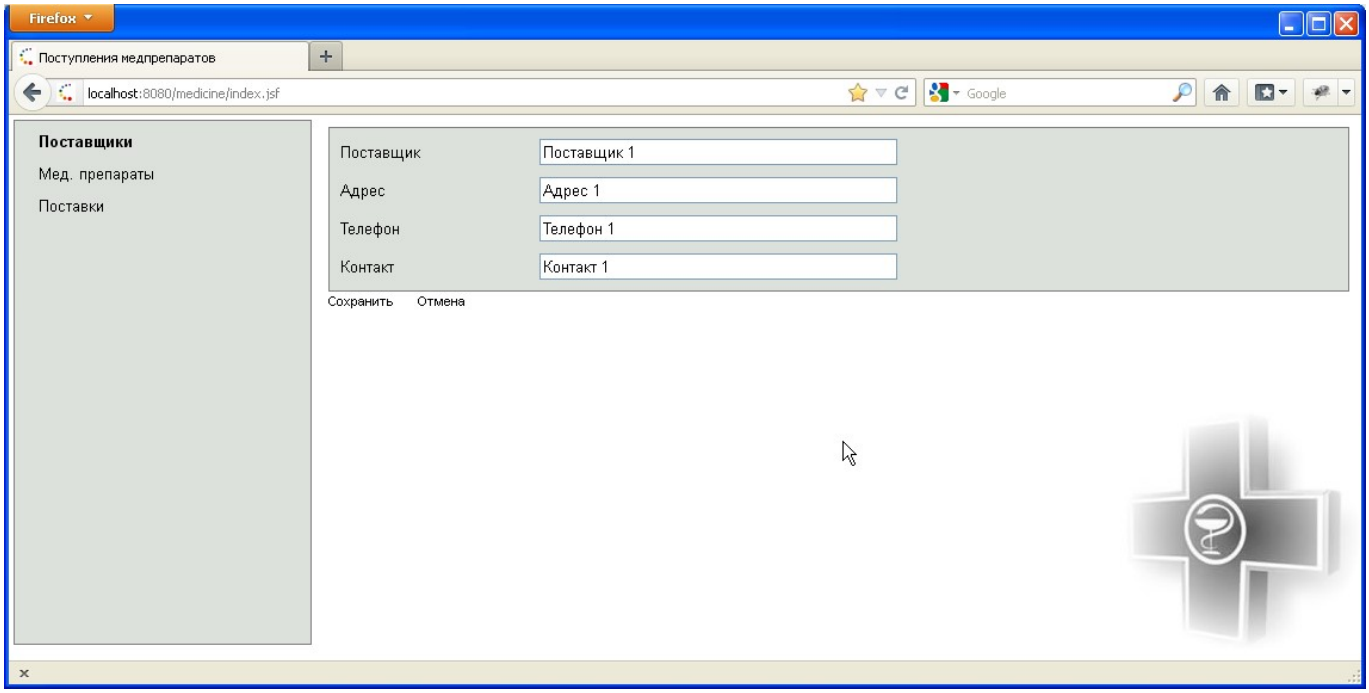

Рисунок 8.2 - Форма редактирования данных о поставщике

На закладке с правонарушениями доступен список всех медицинских препаратах, занесенных в систему (рис. 8.3).

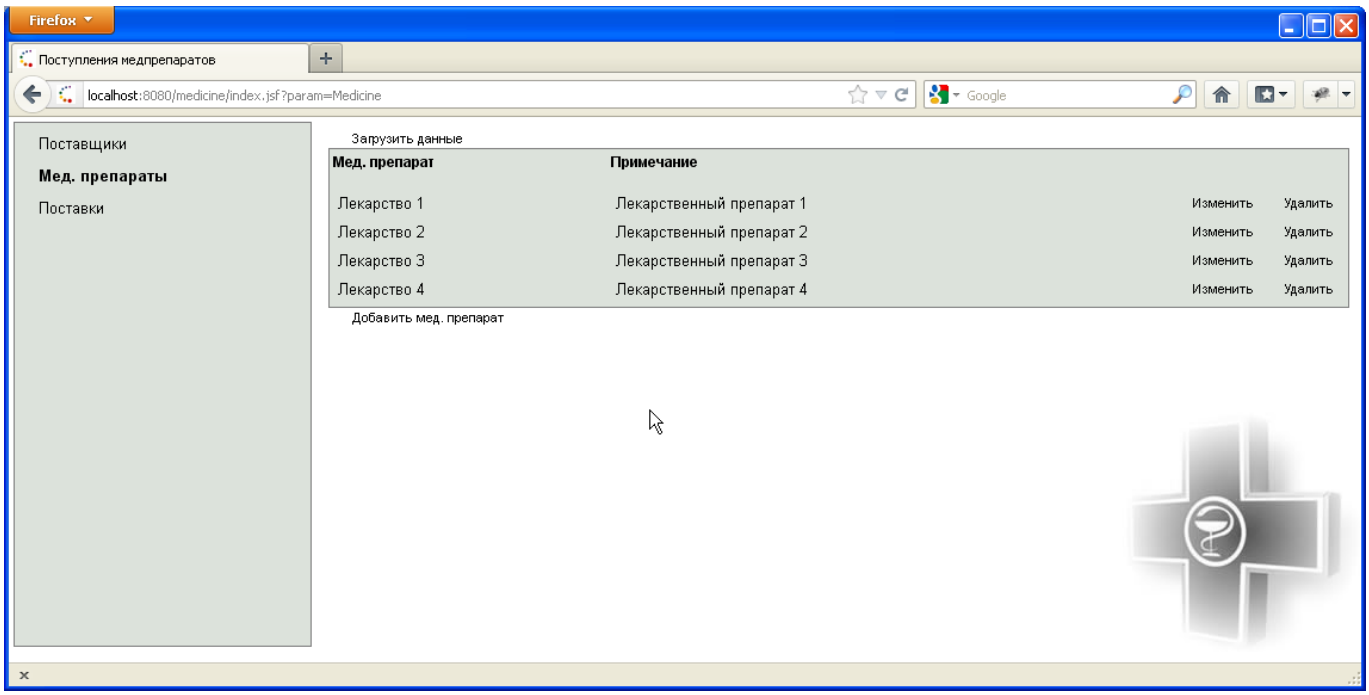

Рисунок 8.3- Страница работы с записями о медицинских препаратах

Форма редактирования данных о поставках показана на рисунке 8.4.

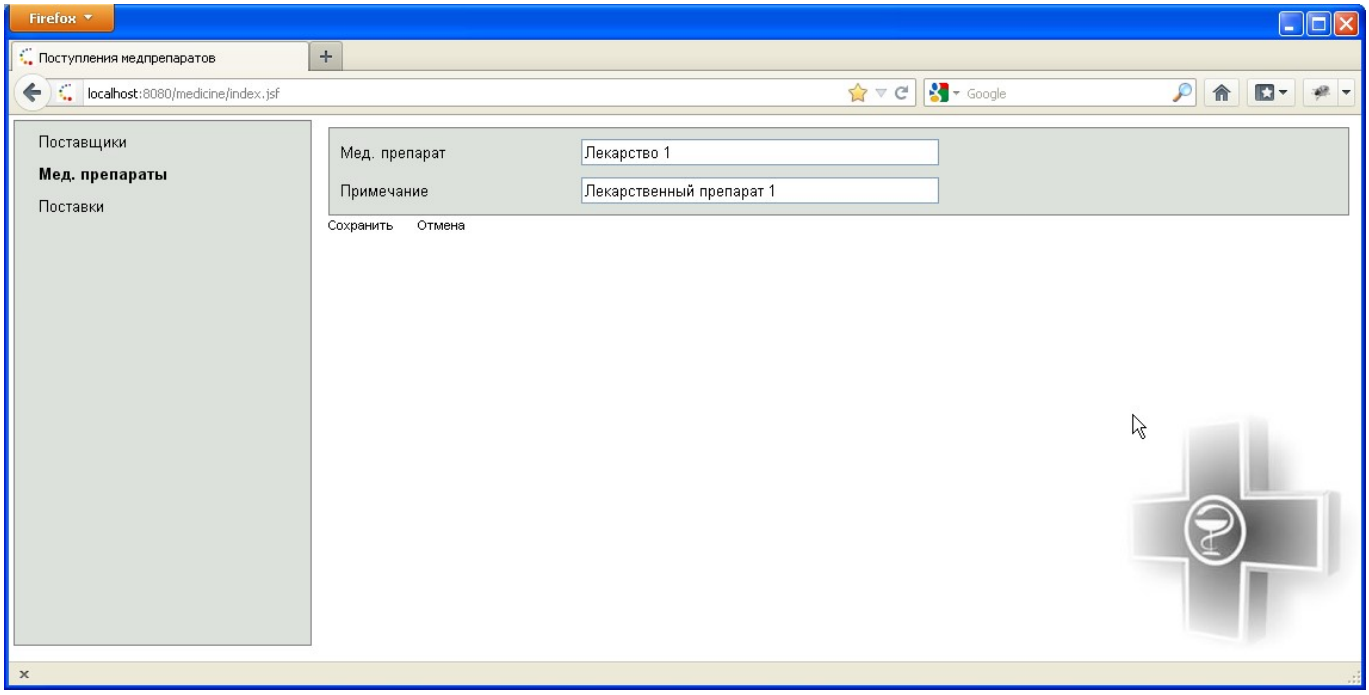

Рисунок 8.4 - Форма редактирования данных о медицинских препаратах

Работа со поставками показана на рисунке 8.5 и 8.6.

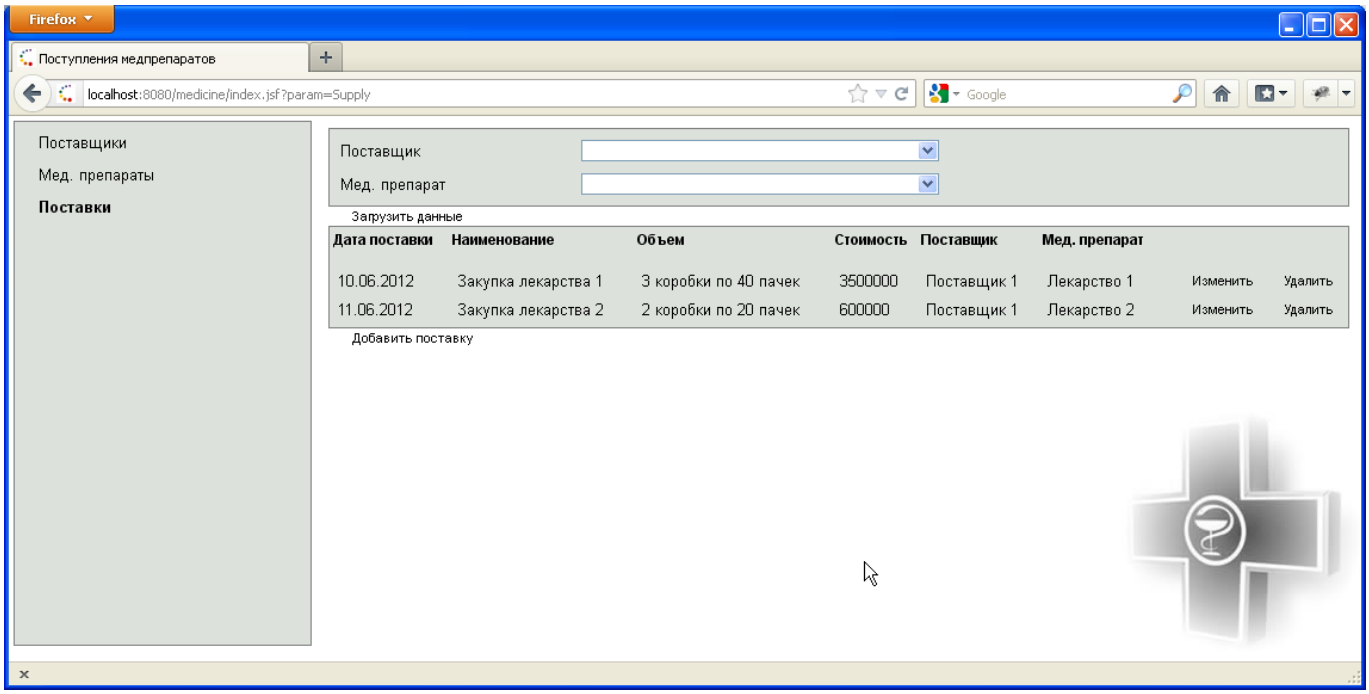

Рисунок 8.5- Страница работы с записями о поставках

Форма редактирования данных о поставке показана на рисунке 8.6.

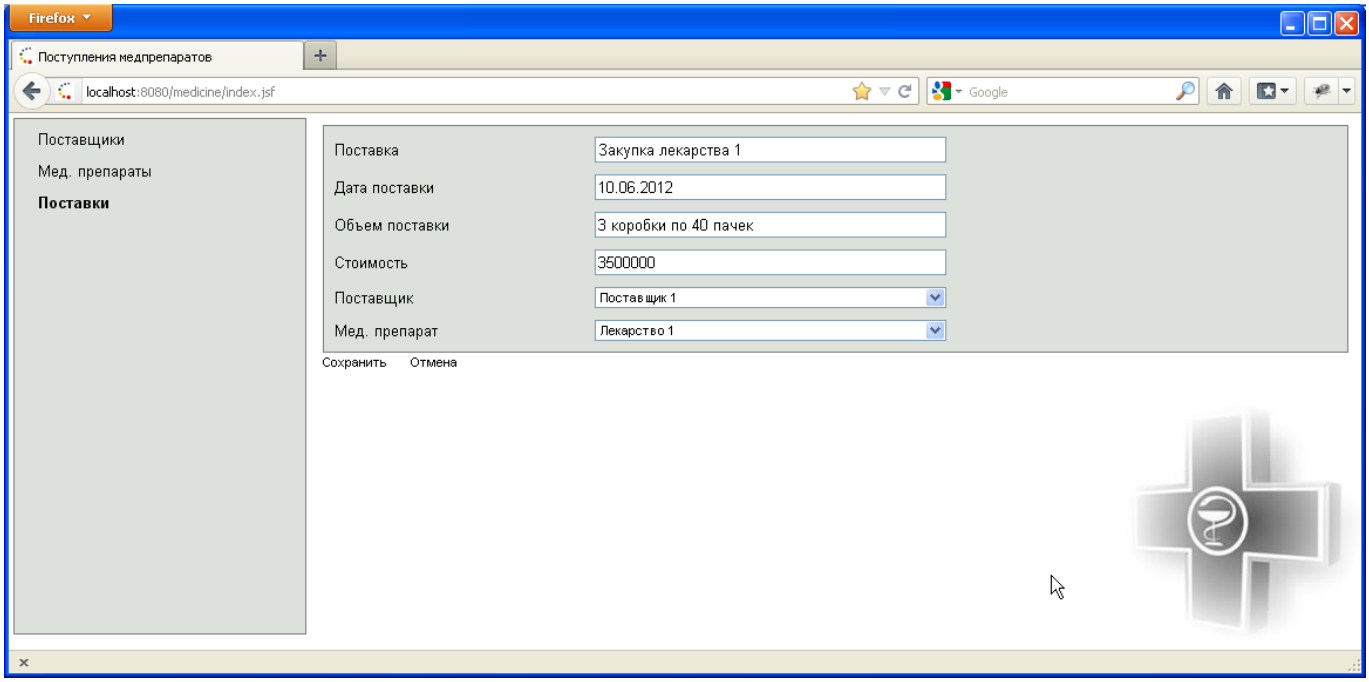

Рисунок 8.6 - Форма редактирования данных поставки

9 Листинг кода

```
DROP DATABASE IF EXISTS `pharmacy_db`;

CREATE DATABASE `pharmacy_db`
    CHARACTER SET 'cp1251'
    COLLATE 'cp1251_general_ci';

USE `pharmacy_db`;

#
# Structure for the `medicine` table :
#

DROP TABLE IF EXISTS `medicine`;

CREATE TABLE `medicine` (
  `mid` int(11) NOT NULL,
  `name` varchar(150) DEFAULT NULL,
  `description` varchar(500) DEFAULT NULL,
  PRIMARY KEY (`mid`),
  UNIQUE KEY `mid` (`mid`)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

```
#
# Data for the `medicine` table  (LIMIT 0,500)
#

INSERT INTO `medicine` (`mid`, `name`, `description`) VALUES
  (1,'Лекарство 1','Лекарственный препарат 1'),
  (2,'Лекарство 2','Лекарственный препарат 2'),
  (3,'Лекарство 3','Лекарственный препарат 3'),
  (4,'Лекарство 4','Лекарственный препарат 4');

COMMIT;

#
# Structure for the `supplier` table :
#

DROP TABLE IF EXISTS `supplier`;

CREATE TABLE `supplier` (
  `sid` int(11) NOT NULL,
  `name` varchar(250) DEFAULT NULL,
  `address` varchar(250) DEFAULT NULL,
  `phone` varchar(250) DEFAULT NULL,
  `contact` varchar(250) DEFAULT NULL,
  PRIMARY KEY (`sid`),
  UNIQUE KEY `sid` (`sid`)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;

#
# Data for the `supplier` table  (LIMIT 0,500)
#

INSERT INTO `supplier` (`sid`, `name`, `address`, `phone`, `contact`) VALUES
  (1,'Поставщик 1','Адрес 1','Телефон 1','Контакт 1'),
  (2,'Поставщик 2','Адрес 2','Телефон 2','Контакт 2'),
  (3,'Поставщик 3','Адрес 3','Телефон 3','Контакт 3');

COMMIT;

#
# Structure for the `supply` table :
```

```
#

DROP TABLE IF EXISTS `supply`;

CREATE TABLE `supply` (
  `spid` int(11) NOT NULL,
  `name` varchar(150) DEFAULT NULL,
  `valueof` varchar(500) DEFAULT NULL,
  `dateof` varchar(500) DEFAULT NULL,
  `cost` int(11) NOT NULL,
  `mid` int(11) NOT NULL,
  `sid` int(11) NOT NULL,
  PRIMARY KEY (`spid`),
  UNIQUE KEY `spid` (`spid`),
  KEY `mid` (`mid`),
  KEY `sid` (`sid`),
   CONSTRAINT `supply_medicine_fk` FOREIGN KEY (`mid`) REFERENCES `medicine` (`mid`) ON DELETE CASCADE ON UPDATE CASCADE,
   CONSTRAINT `supply_supplier_fk` FOREIGN KEY (`sid`) REFERENCES `supplier` (`sid`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;

#
# Data for the `supply` table  (LIMIT 0,500)
#

INSERT INTO `supply` (`spid`, `name`, `valueof`, `dateof`, `cost`, `mid`, `sid`) VALUES
  (1,'Закупка лекарства 1','3 коробки по 40 пачек','10.06.2012',3500000,1,1),
  (2,'Закупка лекарства 2','2 коробки по 20 пачек','11.06.2012',600000,2,1);

COMMIT;



/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Исходные программные коды

```
package medicine.ejb.cmp;

import javax.persistence.*;
import java.io.Serializable;
import java.util.Collection;

@Entity
@Table(catalog = "pharmacy_db", name = "medicine")
@NamedQueries(
  {
    @NamedQuery(name = "Medicine.findAll", query = "SELECT c FROM Medicine c order by
c.name "),
    @NamedQuery(name = "Medicine.getLastID", query = "select max ( c.id ) from Medicine c")
  }
)
public class Medicine implements Serializable, IID {
  private int id = -1;

  @Id
  @Column(name = "mid")
  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public int id() {
    return getId();
  }

  private String name;

  @Basic
  @Column(name = "name")
  public String getName() {
    return name;
  }
```

```
public void setName(String name) {
  this.name = name;
}

private String description;

@Basic
@Column(name = "description")
public String getDescription() {
  return description;
}

public void setDescription(String description) {
  this.description = description;
}

private Collection supplies;

@OneToMany(mappedBy = "medicine")
public Collection getSupplies() {
  return supplies;
}

public void setSupplies(Collection supplies) {
  this.supplies = supplies;
}

@Override
public String toString() {
  return getName();
}

@Override
public boolean equals(Object o) {
  if (this == o) return true;
  if (!(o instanceof Medicine)) return false;

  Medicine medicine = (Medicine) o;

  if (id != medicine.id) return false;
```

```
      return true;
    }
  }

  package medicine.ejb.cmp;

  import javax.persistence.*;
  import java.io.Serializable;
  import java.util.Collection;

  @Entity
  @Table(catalog = "pharmacy_db", name = "supplier")
  @NamedQueries(
    {
      @NamedQuery(name = "Supplier.findAll", query = "SELECT c FROM Supplier c order by
c.name "),
      @NamedQuery(name = "Supplier.getLastID", query = "select max ( c.id ) from Supplier c")
    }
  )
  public class Supplier implements Serializable, IID {
    private int id = -1;

    @Id
    @Column(name = "sid")
    public int getId() {
      return id;
    }

    public void setId(int id) {
      this.id = id;
    }

    public int id() {
      return getId();
    }

    private String name;

    @Basic
    @Column(name = "name")
```

```
public String getName() {
  return name;
}

public void setName(String name) {
  this.name = name;
}

private String address;

@Basic
@Column(name = "address")
public String getAddress() {
  return address;
}

public void setAddress(String address) {
  this.address = address;
}

private String phone;

@Basic
@Column(name = "phone")
public String getPhone() {
  return phone;
}

public void setPhone(String phone) {
  this.phone = phone;
}

private String contact;

@Basic
@Column(name = "contact")
public String getContact() {
  return contact;
}

public void setContact(String contact) {
```

```
    this.contact = contact;
  }

  private Collection supplies;

  @OneToMany(mappedBy = "supplier")
  public Collection getSupplies() {
    return supplies;
  }

  public void setSupplies(Collection supplies) {
    this.supplies = supplies;
  }

  @Override
  public String toString() {
    return getName();
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Supplier)) return false;

    Supplier supplier = (Supplier) o;

    if (id != supplier.id) return false;

    return true;
  }
}

package medicine.ejb.cmp;

import javax.persistence.*;
import java.io.Serializable;

@Entity
@Table(catalog = "pharmacy_db", name = "supply")
@NamedQueries(
  {
```

```
        @NamedQuery(name = "Supply.findAllByMedicine",
          query = "SELECT c FROM Supply c where c.medicine.id = :medicine_id order by c.dateof "
        ),
        @NamedQuery(name = "Supply.findAllBySupplier",
          query = "SELECT c FROM Supply c where c.supplier.id = :supplier_id order by c.dateof "
        ),
        @NamedQuery(name = "Supply.findAllBySupplierAndMedicine",
           query = "SELECT c FROM Supply c where c.supplier.id = :supplier_id and c.medicine.id =
:medicine_id order by c.dateof "
        ),
        @NamedQuery(name = "Supply.findAll", query = "SELECT c FROM Supply c order by c.dateof
"),
        @NamedQuery(name = "Supply.getLastID", query = "select max ( c.id ) from Supply c")
      }
    )
    public class Supply implements Serializable, IID {
      private int id = -1;

      @Id
      @Column(name = "spid")
      public int getId() {
        return id;
      }

      public void setId(int id) {
        this.id = id;
      }

      public int id() {
        return getId();
      }

      private String name;

      @Basic
      @Column(name = "name")
      public String getName() {
        return name;
      }

      public void setName(String name) {
```

```
  this.name = name;
}

private String dateof;

@Basic
@Column(name = "dateof")
public String getDateof() {
  return dateof;
}

public void setDateof(String dateof) {
  this.dateof = dateof;
}

private String valueof;

@Basic
@Column(name = "valueof")
public String getValueof() {
  return valueof;
}

public void setValueof(String valueof) {
  this.valueof = valueof;
}

private int cost;

@Basic
@Column(name = "cost")
public int getCost() {
  return cost;
}

public void setCost(int cost) {
  this.cost = cost;
}

private Medicine medicine;
```

```
  @ManyToOne
  @JoinColumn(name = "mid", referencedColumnName = "mid", nullable = false)
  public Medicine getMedicine() {
    return medicine;
  }

  public void setMedicine(Medicine medicine) {
    this.medicine = medicine;
  }

  private Supplier supplier;

  @ManyToOne
  @JoinColumn(name = "sid", referencedColumnName = "sid", nullable = false)
  public Supplier getSupplier() {
    return supplier;
  }

  public void setSupplier(Supplier supplier) {
    this.supplier = supplier;
  }

  @Override
  public String toString() {
    return getDateof()+". "+getName();
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Supply)) return false;

    Supply supply = (Supply) o;

    if (id != supply.id) return false;

    return true;
  }
}

package medicine.ejb.session;
```

```
import medicine.ejb.cmp.Medicine;
import medicine.ejb.cmp.Supplier;
import medicine.ejb.cmp.Supply;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import java.util.List;

@Stateless(name = "MedicineSessionBeanEJB")
public class MedicineSessionBean implements IMedicineSessionBean,
IMedicineSessionBeanLocal {
  @PersistenceContext
  private EntityManager em;

  private void supplierMerge(Supplier supplier, Supplier ejb) {
    ejb.setAddress( supplier.getAddress() );
    ejb.setContact( supplier.getContact() );
    ejb.setName( supplier.getName() );
    ejb.setPhone( supplier.getPhone() );
  }

  public void supplierAdd(Supplier supplier) {
    try {
      Supplier ejb = new Supplier();
      ejb.setId(getLastID("Supplier.getLastID"));
      supplierMerge(supplier, ejb);
      em.persist(ejb);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void supplierUpdate(Supplier supplier) {
    try {
      Supplier ejb = getSupplier(supplier.getId());
      supplierMerge(supplier, ejb);
      em.persist(ejb);
    } catch (Exception e) {
```

```
    e.printStackTrace();
  }

}

public void supplierDelete(Supplier supplier) {
  try {
    em.remove(getSupplier(supplier.getId()));
  } catch (Exception e) {
    e.printStackTrace();
  }
}

public List supplierAllRecords() {
  try {
    Query query = em.createNamedQuery("Supplier.findAll");
    return query.getResultList();
  } catch (Exception e) {
    e.printStackTrace();
  }
  return null;
}

public Supplier supplierLoad(Supplier supplier) {
  try {
    Supplier ejb = getSupplier(supplier.getId());
    return ejb;
  } catch (Exception e) {
    e.printStackTrace();
  }
  return null;
}

private Supplier getSupplier(Integer recordID) {
  return em.find(Supplier.class, recordID);
}

// ----------------------------------------------------------

private void supplyMerge(Supply supply, Supply ejb) {
  ejb.setCost( supply.getCost() );
```

```
  ejb.setDateof( supply.getDateof() );
  ejb.setName( supply.getName() );
  ejb.setValueof( supply.getValueof() );

  ejb.setMedicine( getMedicine(supply.getMedicine().getId()) );
  ejb.setSupplier( getSupplier(supply.getSupplier().getId()) );
}

public void supplyAdd(Supply supply) {
  try {
    Supply ejb = new Supply();
    ejb.setId(getLastID("Supply.getLastID"));
    supplyMerge(supply, ejb);
    em.persist(ejb);
  } catch (Exception e) {
    e.printStackTrace();
  }
}

public void supplyUpdate(Supply supply) {
  try {
    Supply ejb = getSupply(supply.getId());
    supplyMerge(supply, ejb);
    em.persist(ejb);
  } catch (Exception e) {
    e.printStackTrace();
  }
}

public void supplyDelete(Supply supply) {
  try {
    em.remove(getSupply(supply.getId()));
  } catch (Exception e) {
    e.printStackTrace();
  }
}

public List supplyAllRecords(Medicine medicine, Supplier supplier) {
  try {
    Query query = null;
    if( (medicine!=null && medicine.getId() != -1) && (supplier==null || supplier.getId() == -1))
```

```
{
          query = em.createNamedQuery("Supply.findAllByMedicine");
          query.setParameter("medicine_id", medicine.getId());
        } else
        if( (medicine==null || medicine.getId() == -1) && (supplier!=null && supplier.getId() != -1))
{
          query = em.createNamedQuery("Supply.findAllBySupplier");
          query.setParameter("supplier_id", supplier.getId());
        } else
        if( (medicine!=null && medicine.getId() != -1) && (supplier!=null && supplier.getId() != -1))
{
          query = em.createNamedQuery("Supply.findAllBySupplierAndMedicine");
          query.setParameter("supplier_id", supplier.getId());
          query.setParameter("medicine_id", medicine.getId());
        } else {
          query = em.createNamedQuery("Supply.findAll");
        }
        return query.getResultList();
      } catch (Exception e) {
        e.printStackTrace();
      }
      return null;
    }

    public Supply supplyLoad(Supply supply) {
      try {
        Supply ejb = getSupply(supply.getId());
        return ejb;
      } catch (Exception e) {
        e.printStackTrace();
      }
      return null;
    }

    private Supply getSupply(Integer recordID) {
      return em.find(Supply.class, recordID);
    }

    // -----------------------------------------------------------

    public void medicineMerge(Medicine medicine, Medicine ejb) {
```

```
  ejb.setDescription( medicine.getDescription() );
  ejb.setName( medicine.getName() );
}

public void medicineAdd(Medicine medicine) {
  try {
    Medicine ejb = new Medicine();
    ejb.setId(getLastID("Medicine.getLastID"));
    medicineMerge(medicine, ejb);
    em.persist(ejb);
  } catch (Exception e) {
    e.printStackTrace();
  }
}

public void MedicineUpdate(Medicine medicine) {
  try {
    Medicine ejb = getMedicine(medicine.getId());
    medicineMerge(medicine, ejb);
    em.persist(ejb);
  } catch (Exception e) {
    e.printStackTrace();
  }

}
 public void MedicineDelete(Medicine medicine) {
  try {
    em.remove(getMedicine(medicine.getId()));
  } catch (Exception e) {
    e.printStackTrace();
  }
}

public List medicineRecords() {
  try {
    Query query = em.createNamedQuery("Medicine.findAll");
    return query.getResultList();
  } catch (Exception e) {
    e.printStackTrace();
  }
  return null;
```

```
  }

  public Medicine medicineLoad(Medicine medicine) {
   try {
    Medicine ejb = getMedicine(medicine.getId());
    return ejb;
   } catch (Exception e) {
    e.printStackTrace();
   }
   return null;
  }

  private Medicine getMedicine(Integer recordID) {
   return em.find(Medicine.class, recordID);
  }

// ------------------------------------------------------------
        Integer getLastID(String namedQuery) {
   Query query = em.createNamedQuery(namedQuery);
   Object result = query.getSingleResult();
   return getLastID((Integer)result );
  }

  Integer getLastID(Integer lastID) {
   if(lastID==null) return 1;
   else return lastID + 1;
  }
}
```

10 Список литературы

1. Р. Мюллер. Базы данных и UML: Проектирование.– Лори, 2002г. 432 с.
2. Фельдман С.К. Система программирования Java без секретов: Как создать безопасное приложение с "нуля". – Новый издательский дом" , 2005 г. , 347 с.
3. Дейтел П.Дж., Дейтел Х.М. Как програмировать на Java. Книга 2. Файлы, сети, базы данных. – "Бином" · 2005 г., 672 с.
4. Немного о Java [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.java.com/ru/download/faq/whatis_java
5. Клиент-сервер [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.mstu.edu.ru/study/materials/zelenkov/ch
6. Понятие клиент-серверных систем [Электронный ресурс]. – Электронные данные. – Режим доступа: http://bourabai.kz/dbt/client1.htm
7. [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.bytemag.ru/articles/detail.php?ID=6547

11 Лист задания

Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный университет информатики и радиоэлектроники
Факультет непрерывного и инновационного обучения
Кафедра проектирования информационно-компьютерных систем

«УТВЕРЖДАЮ» Заведующий кафедрой _____ В.В. Хорошко «___» _____ 2020

З А Д А Н И Е

к курсовой работе по дисциплине «Современные технологии проектирования информационных систем»

Фамилия, имя, отчество Букляревич Галина Владимировна

группа 784371

1.Тема проекта: Разработка информационной системы для предметной области «Виртуальная аптека»

2.Сроки сдачи студентом законченного проекта: 3 мая 2020 г.

3.Исходные данные к проекту:

3.1.Описание к выполнению

3.2.Язык и среда программирования – на выбор студента. Однако разработанное программное обеспечение должно быть реализовано на объектно-ориентированном языке.

3.3.В реализации программного обеспечения учесть возможность использования сервера.

3.4.Пояснительную записку и графический материал выполнять по СТП БГУИР 01-2013.

3.5.Другие требования уточняются студентом в процессе работы.

4. Содержание расчётно-пояснительной записки (перечень подлежащих разработке вопросов):

Титульный лист. Заполненный бланк задания с приложением. Содержание (1-2 *стр.*)

Введение (*1 – 3 стр. Актуальность темы курсовой работы; цель и перечень задач, которые планируется решить; детальная постановка задачи*).

4.1.Описание проекта (*10 – 15 стр. Описание серверной и клиентской части разрабатываемого проекта*).

4.2.Обоснование выбора технологий (*7-15 стр. Технологии программирования, используемые для решения поставленных задач. Реализация объектно-ориентированных технологий программирования в современных программно-математических средах*).

4.3.Инструментарий (*5-7 стр. Обоснование используемых инструментов. Использование системы контроля версий GIT. Обязательна ссылка на репозиторий с проектом, например github.com*.).

4.4.Архитектурный шаблон проектирования MVC (*5-7 стр. Разработка схемы алгоритма, диаграммы последовательности и диаграммы состояний (схемы в Приложении) с детальными пояснениями каждого компонента шаблона проектирования MVC или его модификаций*).

4.5.Шаблон проектирования практических решений (*7-10 стр. Использование шаблонов проектирования практических решений для решения практических задач*).

Заключение (*1 стр. Выводы по курсовой работе*).

Список литературных источников (*1, 2 стр. Перечень литературы и интернет-источников, которые были реально использованы при выполнении курсовой работы).*

Приложения (*3 и более стр. Ведомость документов, листинг программного кода и др.).*

5.Перечень графического материала (с указанием обязательных чертежей и графиков):

5.1.Структура графического пользовательского интерфейса (формат А3 или несколько А4)

5.2.Схема алгоритма (формат А3 или несколько А4)

5.3.Диаграмма последовательности (формат А3 или несколько А4)

5.4.Диаграмма состояний (формат А3 или несколько А4)

6.Консультант по работе: Михалькевич Александр Викторович

7.Дата выдачи задания:

8.Календарный график работы над проектом на весь период проектирования:

№ п/п	Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание
1.	1-я опроцентовка (пп. 4.1, 4.2, 5.1)	04.03.2020	40%
2.	2-я опроцентовка (пп. 4.3, 4.4, 5.2, 5.3)	01.04.2020	70%...80%
3.	3-я опроцентовка (пп. 4.5, 4.6, 5.4, приложения)	29.04.2020	95%
4.	Сдача на проверку и защита курсового проекта	03.05.2020	100%
5.	Защита курсового проекта	10-11.05.2020	Согласно графику

Руководитель _____

А.В.Михалькевич

Задание принял к исполнению

Заключение

В ходе выполнения курсового проекта была разработана простейшая электронная система учета поставок медпрепаратов. Программа реализована с использованием языка программирования Java, JSF, технологий EJB v3, сервера приложения JBoss и сервера базы данных MySQL. ПО представляет собой образец реального корпоративного J2EE приложения, функционирующего на основе сетевых технологий.. Применение архитектуры веб-клиента в разработанной подсистеме предоставляет пользователям возможность с различных компьютеров обращаться к серверу за необходимой информацией. Сервер предоставляет возможность для клиента для работы с базой данных. Он выступает в качестве посредника между клиентом и базой данных. Он принимает запросы от клиента, их обрабатывает и направляет в базу данных. Также он перенаправляет запросы назад клиенту. Клиент имеет возможность работы с информацией, хранимой в базе данных. Такая информация представлена в виде таблиц базы данных. Клиент может просматривать, редактировать, осуществлять поиск и т.д. необходимой ему информации. Применение базы данных в качестве хранилища информации позволяет оптимально и эффективно хранить информацию, ее структурировать. Кроме того, в результате выполнения данного курсового проекта были получены дополнительные навыки в работе с языком JAVA, UML, были получены знания о различных методах построения функциональных и информационных моделей. В ходе тестирования разработанного ПО было установлено, что оно работает корректно и соответствует заявленным функциональным требованиям. В целом можно считать, что цель курсовой работы достигнута. Проект доступен по ссылке:
https://drive.google.com/open?id=10Yc2Rbancnyvs2ZODEhuf1P5QgR0AKij

Список использованных источников

Приложения