**СОДЕРЖАНИЕ**

Введение 5

1 Описание работы парикмахерской 6

2 Постановка задачи и обзор методов ее решения 8

2.1 Постановка задачи 8

2.2 Обзор методов решения поставленных задач 9

3 Моделирование программного средства 11

3.1 Функциональное моделирование на основе стандарта IDEF011

3.2 UML моделирование 21

4 Информационная модель системы и её описание 26

5 Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы 27

6 Руководство пользователя 28

7 Результаты тестирования разработанной системы 37

Заключение 38

Список использованных источников 39

Приложение А (Обязательное) Листинг алгоритмов, реализующих бизнес-логику 40

Приложение Б (Обязательное) Листинг основных элементов программы 46

Приложение В (Обязательное) Листинг скрипта генерации базы данных 49

**ВВЕДЕНИЕ**

Профессия парикмахер зародилась еще в древности. Тогда специалист занимался не только прическами, но и являлся лекарем. Сами волосы в те времена считались средством связи человек с высшим миром, а парикмахер через стрижки очищал духовную сущность человека. Позже профессия пережила кризис и длительное время была обесценена. И только с XX века парикмахер вновь становится востребованным на рынке труда.

Парикмахер — это специалист по созданию стиля человека с помощью прически и парика. Человек этой профессии должен знать технологию выполняемой процедуры, но, помимо этого, в работе важна креативность эстетический и художественный вкус при создании особенного стиля для каждого клиента. [1]

Целью данного курсового проекта является автоматизация работы парикмахерской, повышение эффективности работы и снижение количества ошибок во время добавления данный о клиентах.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. исследовать сферу работы парикмахерской (выяснить основные понятия, бизнес-направления);
2. создать систему для работы с данными;
3. сформировать UML диаграммы;
4. разработать удобный и понятный интерфейс, реализованный для работы с услугами парикмахерской;
5. протестировать разработанное программное приложение;

**1 ОПИСАНИЕ РАБОТЫ ПАРИКМАХЕРСКОЙ**

Парикмахерская индустрия стремится к безопасной работе сотрудников, предпринимателей и всех работников отрасли. Отрасль стремится удовлетворять потребности клиентов, обеспечивая безопасное и ответственное обслуживание в парикмахерских. Внешний вид имеет большое значение для благополучия людей, и красивая прическа играет в этом важную роль. Таким образом, парикмахер выполняет важную социальную функцию в нашем обществе. Неслучайно парикмахер часто воспринимается как необходимая часть жизни и как человек, приносящий удачу.

Парикмахер делает разные виды стрижек, окраску, химическую завивку и укладку, занимается лечением волос, наращиванием, ламинированием, корректирует форму прически, а также усов и бороды у мужчин, плетет косы, дреды и другие замысловатые прически. Некоторые парикмахеры специализируются на постижерных работах — изготовлении париков.

Парикмахер может быть специалистом по женским, мужским или детским прическам, или парикмахером-универсалом. Но настоящие волшебники — это парикмахеры-стилисты. Они не просто приводят волосы в порядок, но и меняют образ, они способны с помощью прически подчеркнуть достоинства и скрыть недостатки клиента.

Работа парикмахера предполагает непосредственный контакт с волосами. В их ежедневные обязанности входят:

1. Стрижка волос. Это могут быть как женские, так и мужские прически. Происходят они с использованием инструментов, которые должны быть в наличии у каждого мастера. Есть техники простой модельной стрижки с филировкой перед завершением. Существует и стрижка горячими инструментами и т.д. Всеми процессами необходимо владеть в совершенстве.
2. Укладка волос. Современное парикмахерское искусство предлагает десятки способов придания локонам формы. Выделяются среди них самые популярные: завивка, выпрямление, укладка феном и дифузором, гофрирование, укладка специальными средствами и т.д.
3. Восстановление волос. К восстановительным процедурам относят не только маски. Это ламинирование, кератинирование, мезотерапия и многое другое. Существует масса современных процедур, которыми в совершенстве должен владеть мастер-парикмахер.
4. Плетение и создание причесок. Хороший парикмахер умеет не только работать с распущенными волосами, но и сплетать локоны в причудливые и роскошные прически. Здесь вы сможете проявить свою фантазию и создать на голове у клиентки настоящий шедевр.
5. Окрашивание волос. Существует масса современных технологий покраски. Они направлены не только на изменение цвета, но и на минимальное повреждение при этом. Различные виды тонирования, мелирования, колорирования, обесцвечивания и т.д. Весь спектр работ должен быть в арсенале хорошего мастера.
6. Уход за волосами. Локоны легко повреждаются, поэтому после стрижки или окраски они нуждаются в тщательном уходе. Парикмахер должен определить тип волос и правильно подобрать средство по уходу за ними в салоне. Для этого используют профессиональную продукцию.[2]

**Парикмахер стилист визажист** – более сложное понятие, подчеркивающее то, что мастер прошел курсы стилистики и макияжа. Парикмахер стилист в процессе своей работе должен сделать то, о чем мечтает практически каждый человек: подобрать именно ту стрижку, тот цвет волос, порой очень неожиданные, которые сделают человека ярким, жизнерадостным и красивым, создадут основу его имиджа. Услуги стилиста парикмахера ценятся намного выше, так как малое количество мастеров решаются совершенствовать свое мастерство в создании новых образов. На это нужны деньги и время. Поэтому стилистами становятся действительно профессиональные парикмахеры, имеющие отличный вкус и чувство стиля.

**Парикмахер колорист** – это человек, специализирующийся на окрашивании волос. Он знает, кому какую краску лучше использовать, чтобы не повредить волосы, какой интенсивности оттенок необходим именно для вашего лица, фигуры. На курсах колористов обучают различным комбинациям красок, созданию плавных переходов, эффекта выгоревших волос и многим другим интересным способам выделить владельца прически. Парикмахер визажист, как правило, специализируется на свадьбах и вечерних образах. Очень удобно, сделав вызов парикмахера на дом, получить и мастера макияжа в одном лице.

Стоимость парикмахерских услуг зависит от уровня салона, опыта и мастерства парикмахера, стоимости и качества используемой косметики и оборудования. Парикмахерские услуги на дому, как правило, на порядок ниже, так как в этом случае не берется стоимость за аренду помещения, коммунальные услуги и многие другие факторы. Если у вас есть личный парикмахер, которому вы полностью доверяете собственную шевелюру – то вы настоящий счастливчик![3]

**2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ**

**2.1 Постановка задачи**

Исходя из требований технического задания необходимо создать программу для автоматизации работы парикмахерской. Приложение должно быть выполнено в архитектуре клиент-сервер с организацией взаимодействия с базой данных на объектно-ориентированном языке Java.

В данном программном средстве будет разрешена работа как администратору, так и обычному пользователю.

В возможности администратора будет входить управление учётными записями пользователей, а также добавление и редактирование данных о парикмахерской.

Так как обычными пользователями будут являться сотрудники парикмахерской, для пользователей данного типа будет реализована возможность добавления услуг парикмахерской в базу данных, добавления новой записи на прием с выбором нужной процедуры.

Интерфейс данного приложения должен быть интуитивно понятным для того, чтобы сотрудникам было просто работать с данным приложением. Приложение должно запускаться без использования интегрированных средств разработки.

Также необходимо отладить и протестировать разработанное приложение для того, чтобы была гарантия качественной работы в дальнейшем.

**2.2 Обзор методов решения поставленных задач**

Данное программное приложение будет разработано на объектно-ориентированном языке программирования Java.

Разработанная система будет использовать MySQL .

В разработке данной программной системы будут применены паттерны проектирования Model-View-Controller и паттерн проектирования Фасад.

Паттерны проектирования - это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае.

Model-view-controller (MVC)– шаблон проектирования, с помощью которого его компоненты (модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем) разделены на три отдельных так, что модификация одного из них оказывает минимальное воздействие на остальные. Модель предоставляет данные и методы работы с ними, реагирует на запросы, изменяя своё состояние. Представление отвечает за визуализацию. Часто в качестве представления выступает форма с графическими элементами. Контроллер обеспечивает ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Схема взаимодействия компонентов приведена на рисунке 2.1.

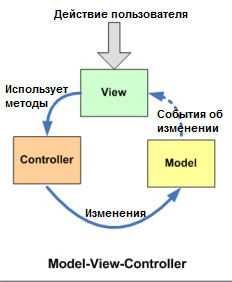


Рисунок 2.1 – Схема работы паттерна проектирования MVC

Главным преимуществом концепции MVC является разделение логики управления приложения, получения данных и их отображения. [4]

Фасад — это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку.

Паттерн Фасад инкапсулирует сложную подсистему в единственный интерфейсный объект. Это позволяет сократить время изучения подсистемы, а также способствует уменьшению степени связанности между подсистемой и потенциально большим количеством клиентов. С другой стороны, если фасад является единственной точкой доступа к подсистеме, то он будет ограничивать возможности, которые могут понадобиться "продвинутым" пользователям.

Схема работы паттерна проектирования Фасад показана на рисунке 2.2.



Рисунок 2.2 – Схема работы паттерна проектирования Фасад

Как и все паттерны проектирования Фасад имеет свои преимущества и недостатки. К преимуществам относится то, что этот паттерн изолирует клиентов от компонентов сложной подсистемы, а к недостаткам – рискует стать божественным объектом, привязанным ко всем классам программы.

**3 МОДЕЛИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Функциональное моделирование на основе стандарта IDEF0**

BPwin - мощный инструмент моделирования, разработанный фирмой Computer Associates Technologies который используется для анализа, документирования и реорганизации сложных бизнес-процессов. Он включает в себя следующие методологии: IDEF0, DFD и IDEF3.

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. [5]

IDEF0 — методология функционального моделирования (англ. function modeling) и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является ее акцент на соподчиненность объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность (поток работ).

Функциональная модель IDEF0 представляет собой набор блоков, каждый из которых представляет собой «черный ящик» со входами и выходами, управлением и механизмами, которые детализируются (декомпозируются) до необходимого уровня. Наиболее важная функция расположена в верхнем левом углу. А соединяются функции между собой при помощи стрелок и описаний функциональных блоков. При этом каждый вид стрелки или активности имеет собственное значение. Данная модель позволяет описать все основные виды процессов, как административные, так и организационные. [6]

На контекстной диаграмме верхнего уровня (рисунок 3.1.1) представлена функциональная модель «Автоматизировать работу парикмахерской», а также определены потоки входных и выходных данных, механизмы ограничения и управления данными.

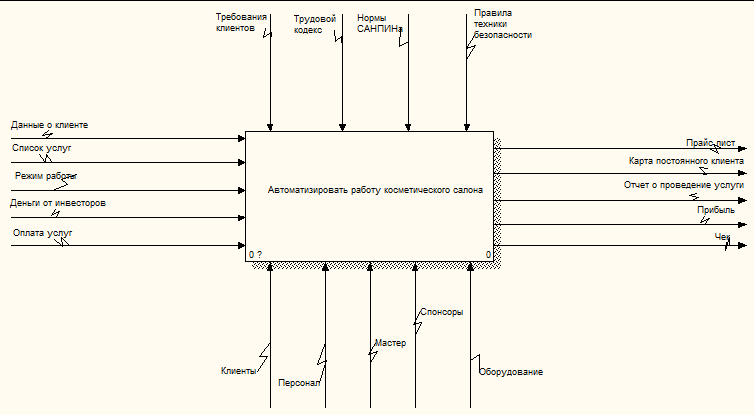


Рисунок 3.1.1 – Контекстная диаграмма верхнего уровня

Входной поток включает в себя данные о клиенте, список услуг, режим работы, деньги от инвесторов, оплата услуг. После соответствующей обработки в выводном потоке имеем прайс-лист, карту постоянного клиента, отчет о проведении услуг, прибыль, чек. В роли управляющих механизмов выступают требования клиентов, трудовой кодекс, трудовой кодекс, нормы САНПИНа, правила техники безопасности. Механизмами являются клиенты, персонал, спонсоры, мастер, оборудование.

На рисунке 3.1.2 отображена декомпозиция контекстной диаграммы, состоящая из трёх блоков: «Записать на приём», «Оказать услуги» и «Оплатить оказанные услуги».

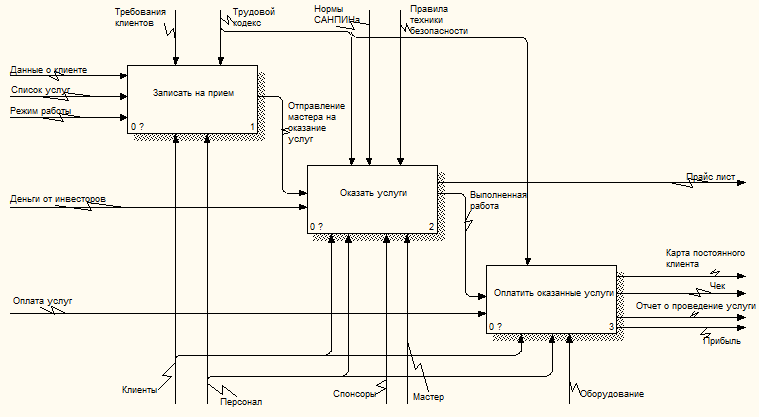


Рисунок 3.1.2 – Декомпозиция контекстной диаграммы верхнего уровня

Первый компонент данной декомпозиции («Записать на приём») подразумевает под собой запись клиента на приём в парикмахерскую.

Декомпозиция данного блока отображена на рисунке 3.1.3. Она представлена тремя компонентами: «Принять данные о клиенте», «Согласовать дату, время» и «Занести информацию в БД».

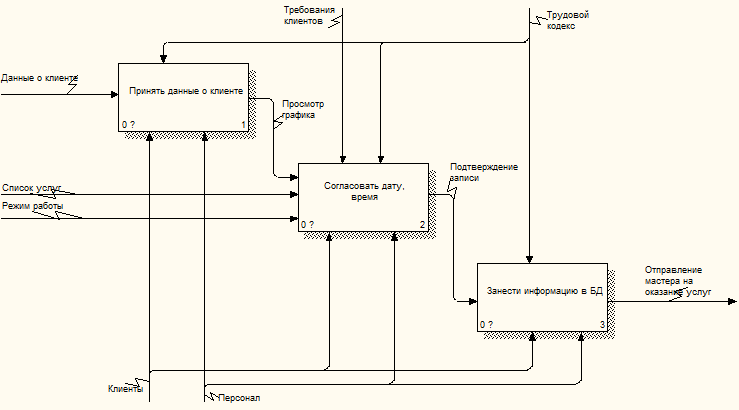


Рисунок 3.1.3 – Декомпозиция блока «Записать на прием»

Второй компонент декомпозированной контекстной диаграммы верхнего уровня («Оказать услуги») подразумевает под собой разработку процедуры оказания услуг.

Декомпозиция данного блока отображена на рисунке 3.1.4. Она представлена тремя компонентами: «Проконсультировать клиента», «Подготовить средства и оборудования для услуги» и «Провести услуги».

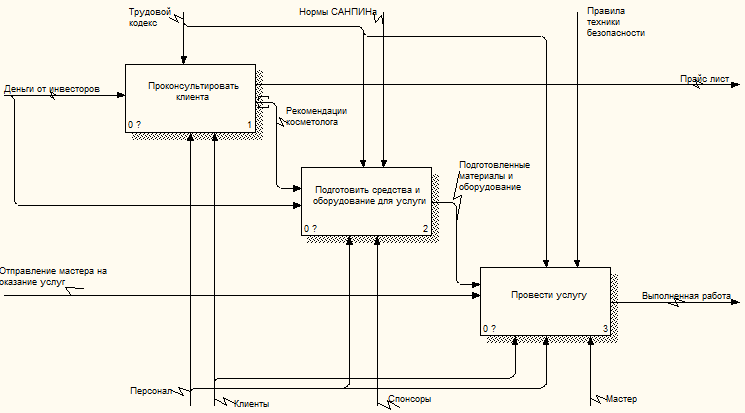


Рисунок 3.1.4 – Декомпозиция блока «Оказать услуги»

Третий компонент декомпозированной контекстной диаграммы верхнего уровня («Оплатить оказанные услуги») подразумевает под собой оплату всех проведенных процедур в парикмахерской.

Декомпозиция данного блока отображена на рисунке 3.1.5. Она представлена тремя компонентами: «Рассчитать стоимость», «Оплатить за услуги» и «Получить чек».

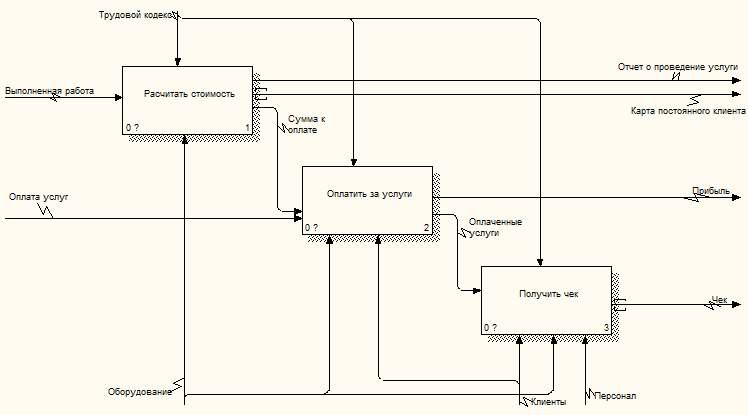


Рисунок 3.1.5 – Декомпозиция блока «Оплатить оказанные услуги»

На рисунке 3.1.6 отображена декомпозиция третьего уровня (блок «Записать на прием») первого компонента («Принять данные о клиенте»), состоящая из трёх блоков: «Записать личные данные клиента», «Записать вид услуги» и «Указать дату и время».

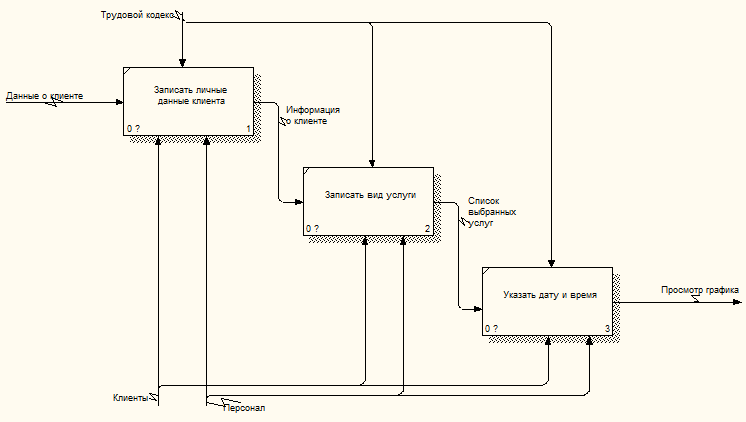


Рисунок 3.1.6 – Декомпозиция блока «Принять данные о клиенте»

На рисунке 3.1.7 отображена декомпозиция третьего уровня (блок «Записать на прием») второго компонента («Согласовать дату, время»), состоящая из четырех блоков: «Проверить наличие места на указанную дату и время», «Оставить выбранную дату», «Выбрать другую дату» и «Подготовить информацию для базы данных».

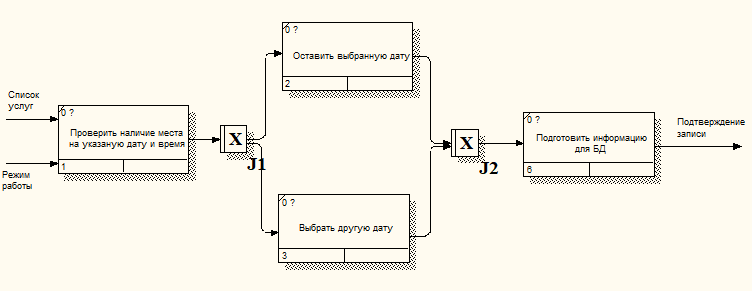


Рисунок 3.1.7 – Декомпозиция блока «Согласовать дату, время»

На рисунке 3.1.8 отображена декомпозиция третьего уровня (блок «Записать на прием») третьего компонента («Занести информацию в базу данных»), состоящая из трех блоков: «Проверить заполнение всех блоков», «Проверить правильность введенных данных» и «Сохранить базу данных».

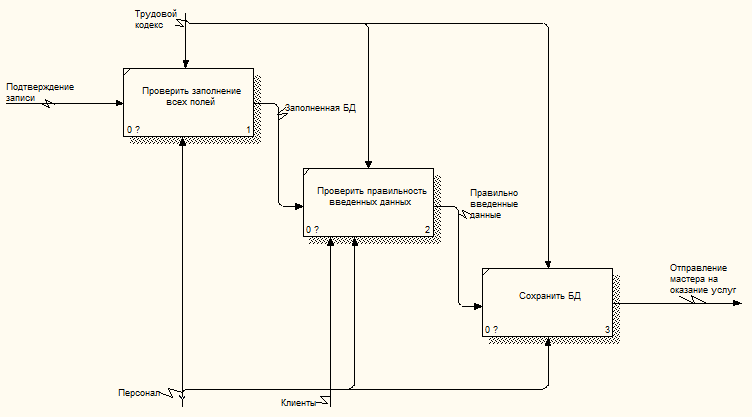


Рисунок 3.1.8 – Декомпозиция блока «Занести информацию в базу данных»

На рисунке 3.1.9 отображена декомпозиция третьего уровня (блок «Оказать услуги») первого компонента («Проконсультировать клиента»), состоящая из трех блоков: «Осмотреть клиента», «Дать рекомендации» и «Составить список рекомендованных средств».

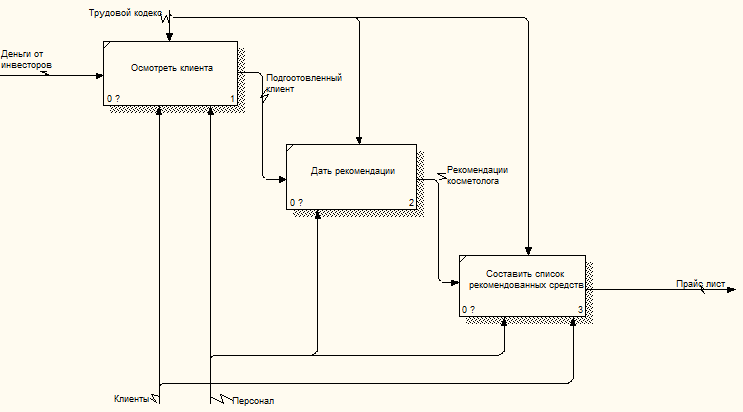


Рисунок 3.1.9 – Декомпозиция блока «Проконсультировать клиента»

На рисунке 3.1.10 отображена декомпозиция третьего уровня (блок «Оказать услуги») второго компонента («Подготовить средства и оборудования для услуги»), состоящая из трех блоков: «Подготовить необходимые средства для процедуры», «Продезинфицировать инструменты» и «Проверить исправность оборудования».

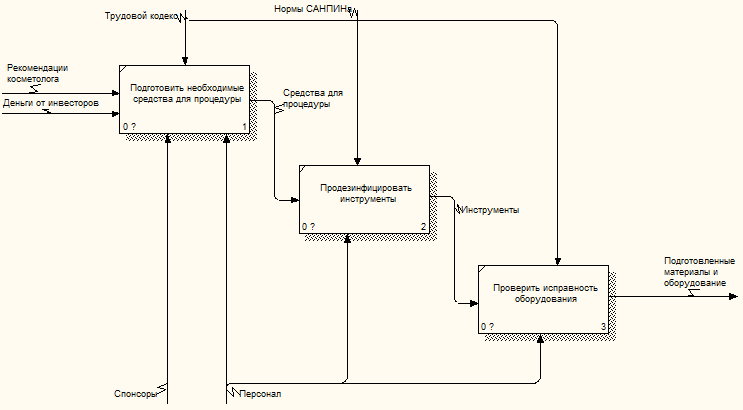


Рисунок 3.1.10 – Декомпозиция блока «Подготовить средства и оборудования для услуги»

На рисунке 3.1.11 отображена декомпозиция третьего уровня (блок «Оказать услуги») третьего компонента («Провести услугу»), состоящая из трех блоков: «Подготовить клиента к процедуре», «Проверить аллергическую реакцию» и «Провести процедуру».

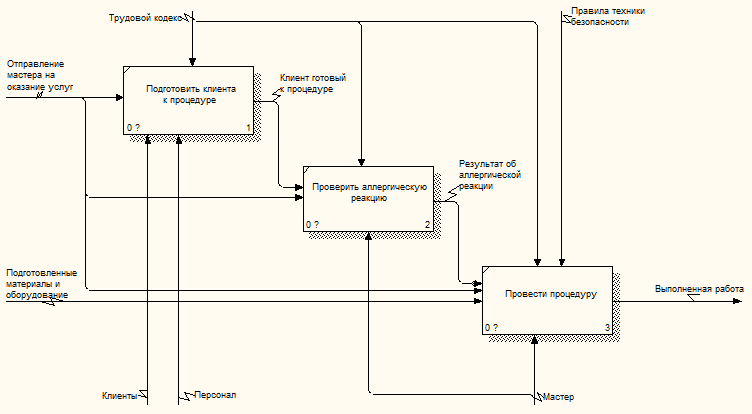


Рисунок 3.1.11 – Декомпозиция блока «Провести услугу»

На рисунке 3.1.12 отображена декомпозиция третьего уровня (блок «Оплатить оказанные услуги») первого компонента («Рассчитать стоимость»), состоящая из трех блоков: «Подсчитать общую стоимость всех сделанных процедур», «Учесть скидку при её наличие» и «Подсчитать конечную стоимость».

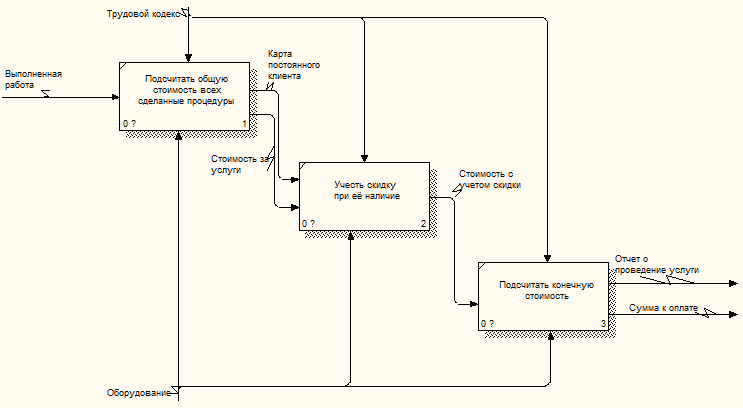


Рисунок 3.1.12 – Декомпозиция блока «Рассчитать стоимость»

На рисунке 3.1.13 отображена декомпозиция третьего уровня (блок «Оплатить оказанные услуги») второго компонента («Оплатить за услуги»), состоящая из четырех блоков: «Определить способ оплаты», «Оплатить картой», «Оплатить наличными» и «Осуществить оплату».

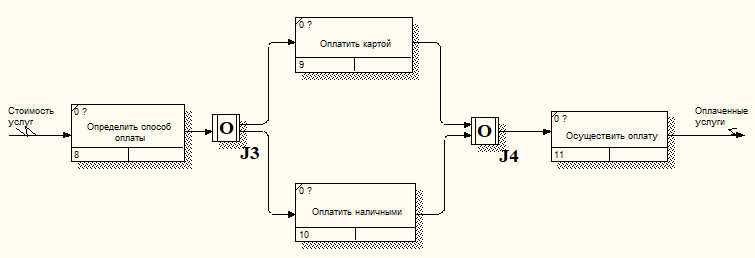


Рисунок 3.1.13 – Декомпозиция блока «Оплатить за услуги»

На рисунке 3.1.14 отображена декомпозиция третьего уровня (блок «Оплатить оказанные услуги») третьего компонента («Получить чек»), состоящая из трех блоков: «Проверить данные чека», «Напечатать чек» и «Выдать чек».

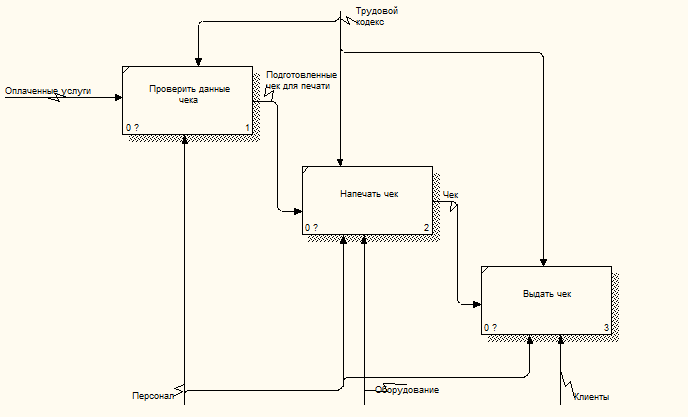


Рисунок 3.1.14 – Декомпозиция блока «Получить чек»

Функциональные модели на основе стандарта IEDF0 показывают методику работы парикмахерской в жизни, а для того, чтобы увидеть работу парикмахерской используя программное средство, разрабатываются UML диаграммы.

**3.2 UML моделирование**

UML – графическая нотация, предназначенная для описания и моделирования процессов, протекающих в ходе разработки.

Были разработаны следующие диаграммы: диаграмма вариантов использования, диаграмма состояний, диаграммы последовательностей, диаграмма развёртывания, диаграмма компонентов, диаграмма классов.

Диаграмма вариантов использования включает в себя актеров. В приложении актерами являются директор сети и сотрудники парикмахерских салонов. Вариантами использования системы являются: регистрация в системе, работа с информацией о услугах, работа с информацией о салонах, работа с информацией о мастерах, просмотр списка добавленных клиентов и другие (рисунок 3.2.1).

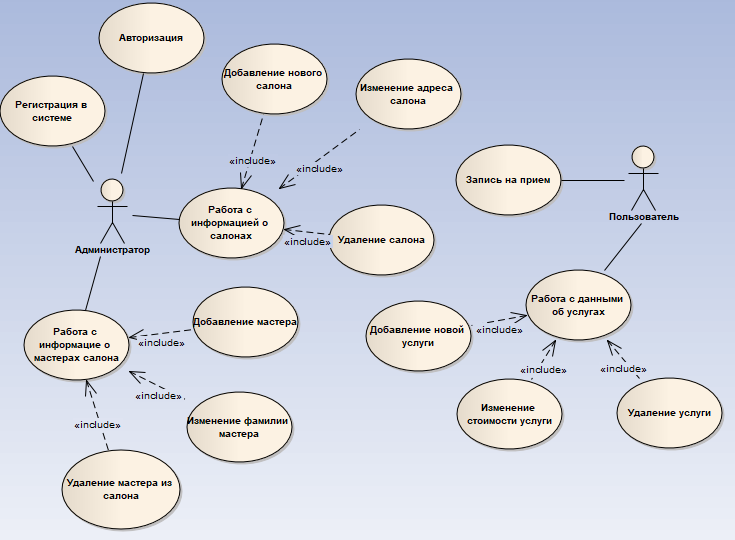


Рисунок 3.2.1 – Диаграмма вариантов использования

Диаграмма состояния показывает все состояния, в которых находится запись на прием в системе (рисунок 3.2.2).

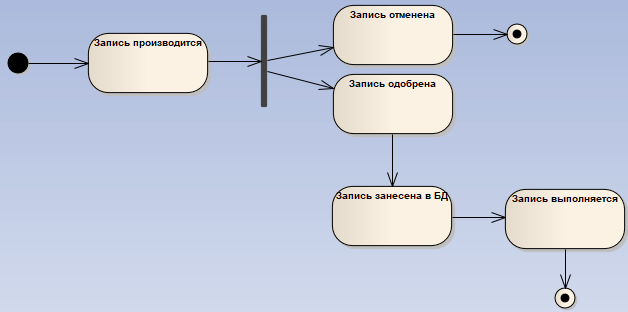


Рисунок 3.2.2 – Диаграмма состояния

Далее представлена диаграмма последовательности, которая демонстрирует как сотрудник парикмахерской может осуществить добавление новой записи на прием в базу данных (рисунок 3.2.3).

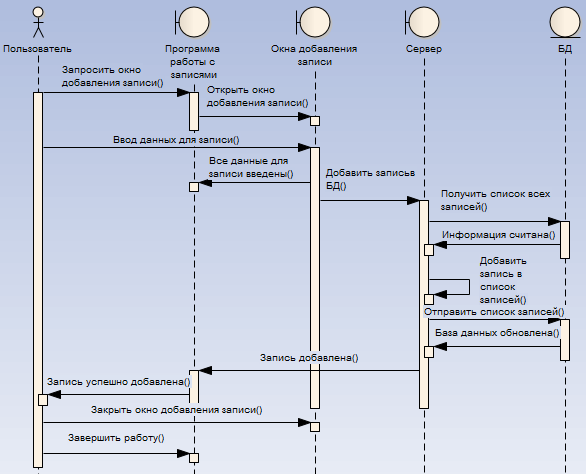


Рисунок 3.2.3 – Диаграмма последовательности

На рисунке 3.2.4 представлена диаграмма компонентов системы.

Диаграмма состоит из шести взаимодействующих компонентов: сотрудник, система работы с данными, база услуги парикмахерской, база клиентов, база мастеров, база записей на прием в парикмахерскую.

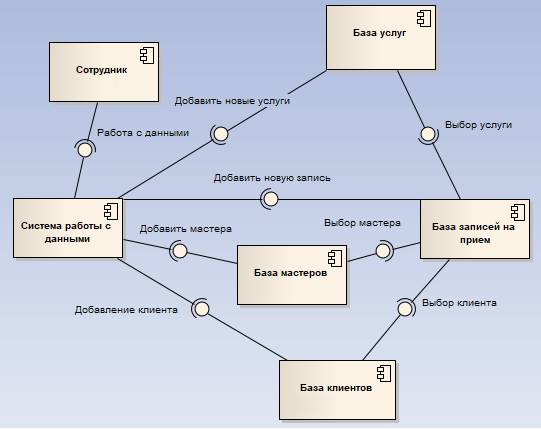


Рисунок 3.2.4 – Диаграмма компонентов

Диаграмма развертывания (рисунок 3.2.5) описывает процесс взаимодействия узлов системы.

Узел База данных содержит компонент Database, который специфицируется артефактом salon. Узлы соединены по средством Ethernet. Среды выполнения взаимодействуют с помощью протокола TCP/IP.

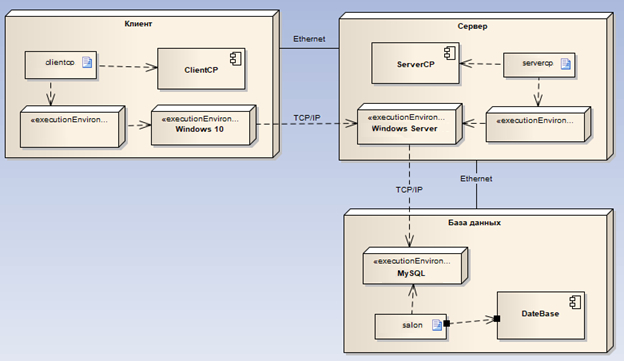


Рисунок 3.2.5 – Диаграмма развертывания системы

На рисунке 3.2.6 представлена диаграмма классов парикмахерской

Диаграмма состоит из пяти классов: Salon, Service, User, Employee и Record.

Для обеспечения удобства работы с данными о парикмахерской, услугах, пользователях и сотрудниках используются интерфейсы Print.

В классе Salon содержится название парикмахерской, город и адрес размещение её и номер телефона.

В классе Service содержится название услуги, её стоимость и тип услуги.

Класс User содержит логин, пароль и роль пользователя, с помощью этих данных осуществляется доступ в систему.

В классе Employee содержится фамилия и имя сотрудника парикмахерской.

Класс Record содержит номер записи в парикмахерскую, дату и время записи.



Рисунок 3.2.6 – Диаграмма классов

Таким образом UML диаграммы наглядно демонстрируют работу программного средства. А для того, чтобы продемонстрировать структуру данные проектируется информационная модель системы.

**4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЁ ОПИСАНИЕ**

AllFusion ERwin Data Modeler (ранее ERwin) — CASE-средство для проектирования и документирования баз данных, которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания.

AllFusion ERwin Data Modeler (ERwin) позволяет наглядно отображать сложные структуры данных. Продукт улучшает коммуникацию организации, обеспечивая совместную работу администраторов и разработчиков баз данных, многократное использование модели, а также наглядное представление комплексных активов данных в удобном для понимания и обслуживания формате. [7]

На рисунке 4.1 представлена логическая модель работы парикмахерской.

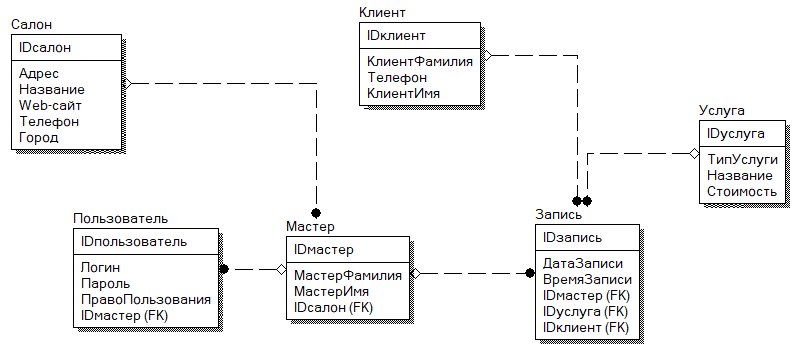


Рисунок 4.1 – Логическая модель работы парикмахерской

Отношение находится в третьей нормальной форме тогда и только тогда, когда находится во второй нормальной форме и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы. [8]

Разработанная модель соответствует вышеперечисленным требованиям, следовательно, находится в третьей нормальной форме.

**5 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ**

Рассмотрим алгоритм, реализующий редактирований данных о клиенте. Данный алгоритм продемонстрирован на рисунке 5.1.

Метод редактирования данных заключается в том, что сервер посылает клиенту сообщение о том, что необходимо редактировать выбранное поле из таблицы, отображающей данные о клиенте. Клиент получает это сообщение и пользователь редактирует необходимое поле, отправляет данные на сервер. Сервер считывает их из потока вывода и обновляет данные в базе данных. После чего выводится таблица с новыми данными.



Рисунок 5.1 – Блок-схема алгоритма редактирования данных о клиенте

**6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Разработанное приложение состоит из двух частей: клиент и сервер. Данное приложение можно запустить без среды разработки. Для этого оно строится из двух файлов: servercp (для сервера) и clientcp (для клиента).

Для начала работы необходимо запустить сервер. При успешном запуске появится сообщение о там, что сервер запущен. После запуска клиентской части, отобразится сообщение о том, что первый клиент подключился (рисунок 7.1).

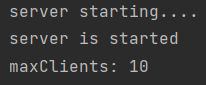


Рисунок 6.1 – Статистика подключений к серверу

При первом входе в систему по экране появится окно для авторизации. Это показано на рисунке 6.2.

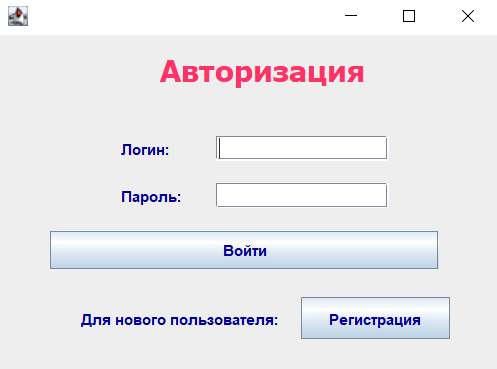


Рисунок 6.2 – Авторизация

Пользователь обладающий логином и паролем админа заходит в меню администратора, где может добавлять новые парикмахерские и добавлять новых мастеров. Это продемонстрированно на рисунке 6.3.

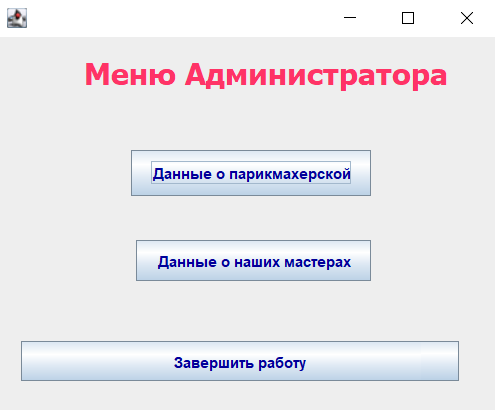


Рисунок 6.3 – Меню администратора

При выборе первого поля в меню администратора открывается окно работы с данными о парикмахерской. В возможности администратора входит добавление новой парикмахерской, редактирование данных и удаление. Это показано на рисунке 6.4.

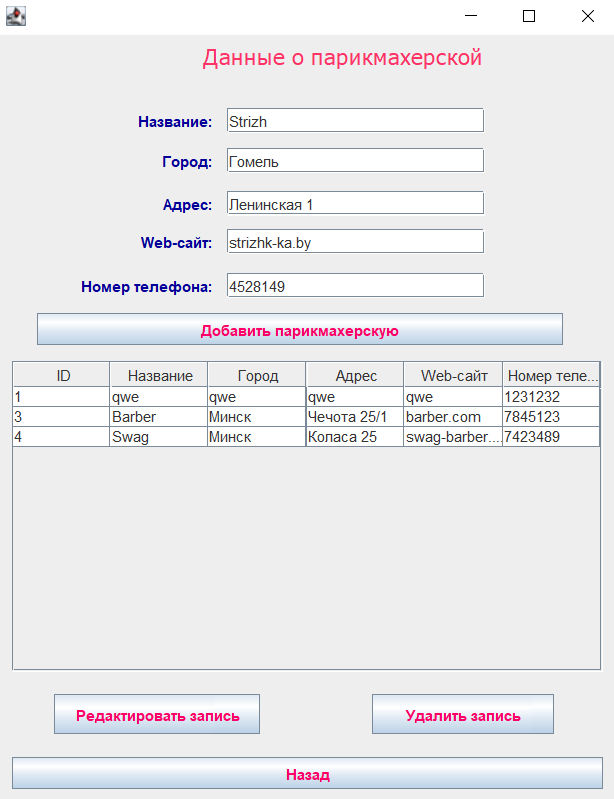


Рисунок 6.4 – Работа с данными о парикмахерской

При нажатии на кнопку “Добавить парикмахерскую” парикмахерская будет добавлена в базу данных (рисунок 6.5).

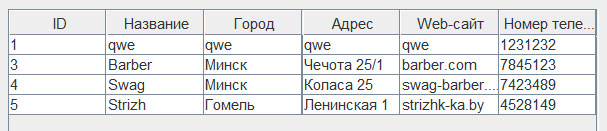


Рисунок 6.5 – Добавление новой парикмахерской

Если администратор нажмет на кнопку “Редактировать запись”, передним откроется диалоговое окно для редактирование выбранной запись. Это продемонстрированно на рисунке 6.6.

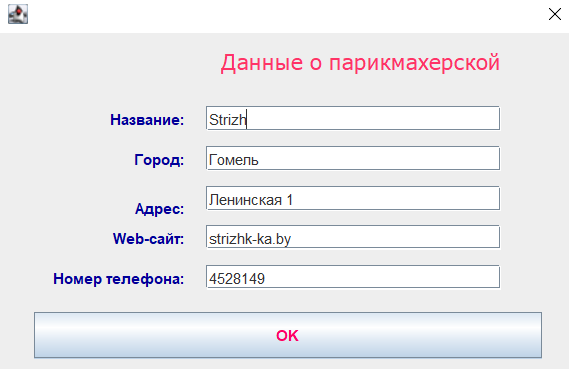


Рисунок 6.6 – Диалоговое окно редактирование данных о парикмахерской

При нажатии на кнопку “Назад” администратор вернется в главное меню. Если администратор выберет в меню поле “Данные о наших мастерах”, то перед ним откроется окно для работы с этими данными, где он сможет добавлять, редактировать и изменять данные. Это продемонстрированно на рисунке 6.7.

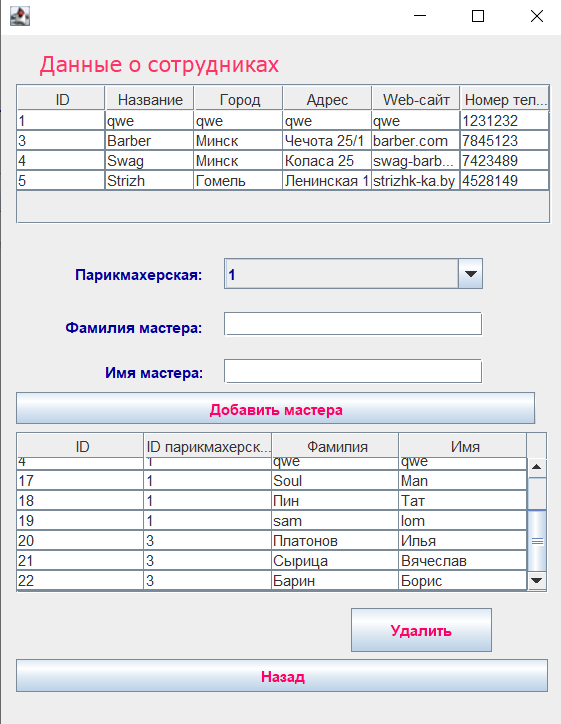


Рисунок 6.7 – Работа с данными о мастерах

При входе клиента в программу перед ним откроется окно авторизации. Это показано на рисунке 6.8.

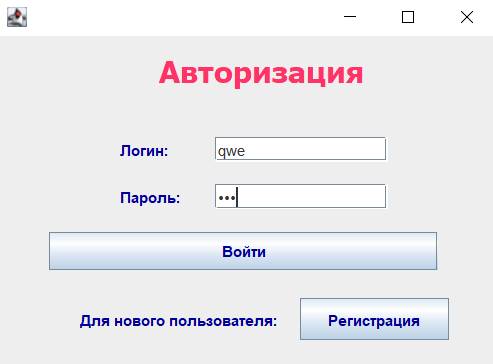


Рисунок 6.8 – Авторизация

При нажатии на кнопку “Регистрация” откроется окно для добавление нового пользователя, который будет зарегистрирован в системе с ролью пользователя. Это продемонстрированно на рисунке 6.9.

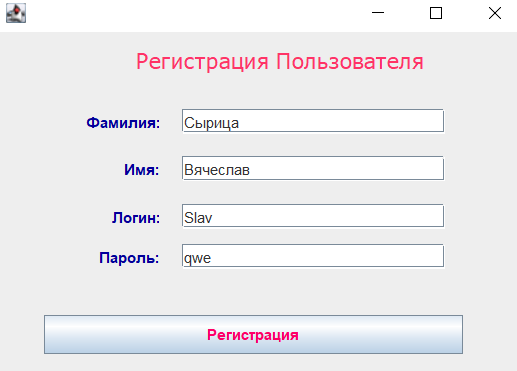


Рисунок 6.9 – Регистрация пользователя

После успешной регистрации откроется главное меню пользователя. В обязанности пользователя входит работа с записью на прием, работа с данными об услугах. Это показано на рисунке 6.10.

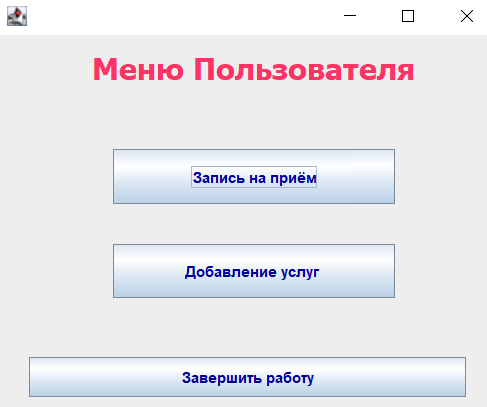


Рисунок 6.10 – Меню пользователя

При нажатии на кнопку “Запись на прием” откроется окно для работы с записью на прием. В этом окне сначала необходимо произвести добавление данных о клиенте, после нажатия на кнопку “Добавить клиента” клиент будет добавлен в базу данных. Данные о клиенте можно редактировать, также можно при необходимости удалить клиента из базы данных. После добавления клиента можно производить уже непосредственно саму запись в парикмахерскую. Для осуществления записи необходимо из выпадающих списков выбрать клиента, которого необходимо записать на приём, вид услуги, которая будет произведена, и мастера, который будет проводить процедуру. Это показано на рисунке 6.11.

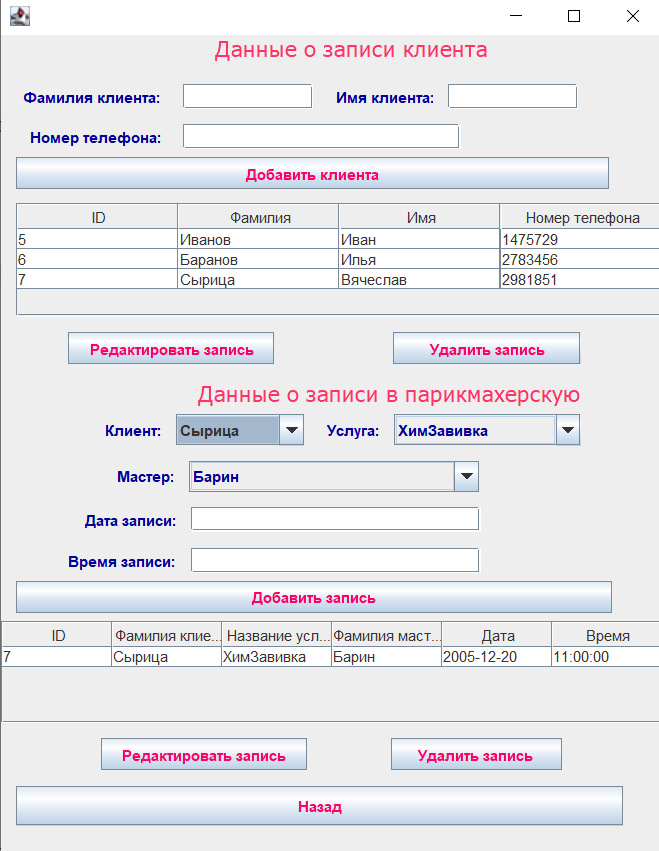


Рисунок 6.11 – Работа с данными о записи на прием

При нажатии на кнопку “Редактировать запись” появится диалоговое окно для редактирования записи на прием. Это показано на рисунке 6.12.

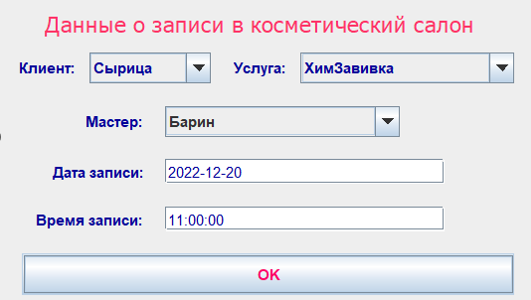


Рисунок 6.12 – Редактирование записи на приём

После нажатия на кнопку “OK” диалоговое окно будет закрыто, а данные таблице будут обновлены. Это продемонстрированно на рисунке 6.13.

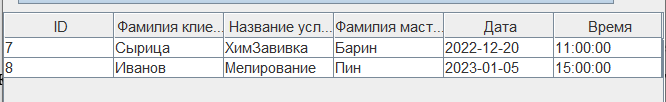


Рисунок 6.13 – Обновленная таблица о записях на приём

При выборе кнопке “Назад” откроется главное меню пользователя. Если нажать на кнопку “Добавление услуг” откроется окно для работы с услугами паркимахерской. В этом окне можно добавлять новые услуги, редактировать и удалять услуги (рисунок 6.14).

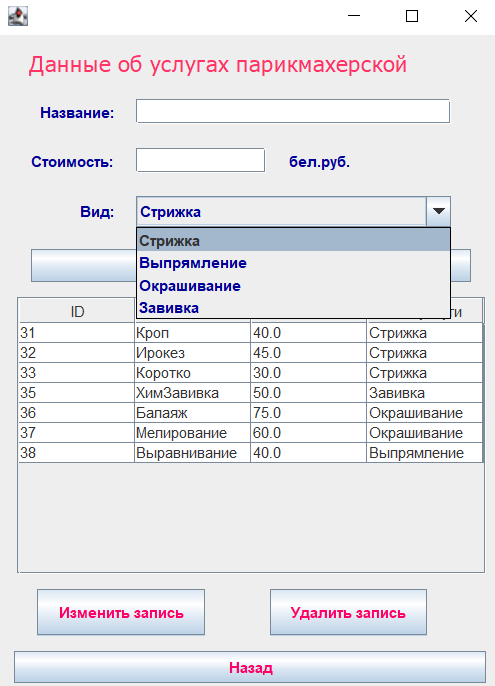


Рисунок 6.14 – Работа с данными об услугах

**7 РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ СИТСЕМЫ**

В разработанном программном средстве постоянно осуществляется ввод данных. Так как данным приложением будут пользоваться обычные пользователи, необходимо провести его тестирование.

При запуске клиентской части приложения, пользователю необходимо проходит авторизацию. Если в окне авторизации пользователь ввел неверный логин, которого не существуют в базе данных, то появится окно, выводящее информацию об ошибке (рисунок 7.1).

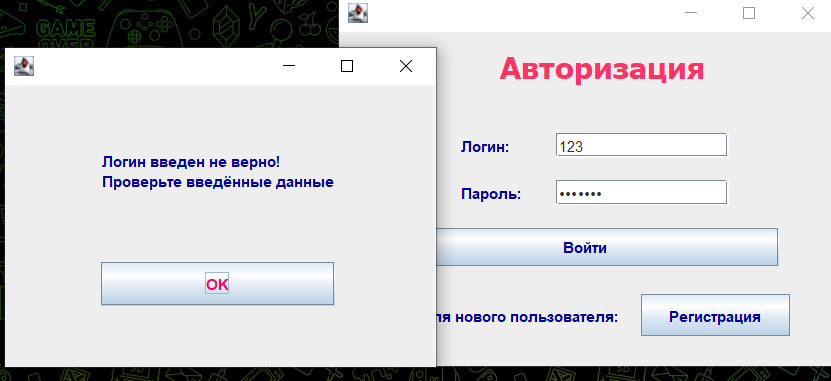
****

Рисунок 7.1 – Окно с информацией о неверном логине

В случае ввода неверного пароля, появится окно с сообщение о вводе неверного пароля. Это продемонстрированно на рисунке 7.2.

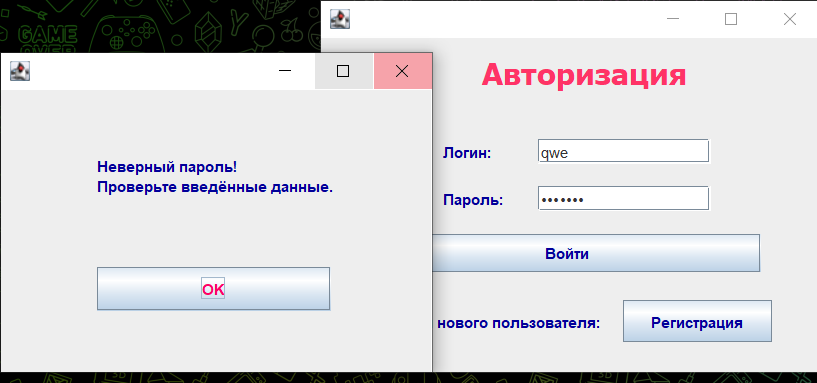
****

Рисунок 7.2 – Окно с информацией о неверном пароле

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была детально исследована работа парикмахерской и реализовано программное приложение, позволяющее производить работу с данными об услугах, о клиентах и о записях на приём. Реализация программного продукта позволяет облегчить процедуру добавления данных парикмахерской. Следовательно, основная задача курсового проекта была успешно выполнена.

В качестве графического представления некоторого количества классов и связей между ними была построена диаграмма классов, которая позволяет наглядно посмотреть на связи между отдельными классами.

Программа удовлетворяет основным характеристикам, которые были заявлены. Оно удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области.

При разработке данной программы была учтена логика пользователя и проанализированы вероятные ошибки ввода, с которыми можно столкнуться. Поэтому в программе реализованы проверки на ввод данных, и при неправильном вводе пользователь получает оповещение-подсказку, помогающую ему сориентироваться и исправить ввод.

Для наглядности работы парикмахерской была разработана функциональная модель, а для наглядности работы программы была описана работа всей программы.

Программа легко адаптируется под другую предметную область и в дальнейшем может легко редактироваться и совершенствоваться. Это дает возможность внедрить данный программный продукт в другую среду, близкую по выполняемым задачам данной области.

Подводя итог, можно с уверенностью сказать, что цели и задачи, поставленные перед данной работой, успешно достигнуты и выполнены.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Kp.ru [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.kp.ru/putevoditel/obrazovanie/parikmakher/

[2] Edunews [Электронный ресурс]. – Электронные данные. – Режим доступа: https://edunews.ru/professii/obzor/moda/parikmaher.html

[3] Firmika [Электронный ресурс]. – Электронные данные. – Режим доступа: https://beauty.firmika.ru/\_\_stilist\_parikmaher

[4] Cpp-reference [Электронный ресурс] – Электронные данные. – Режим доступа: http://cpp-reference.ru/patterns/structural-patterns/facade/

[5] Intellect.ml [Электронный ресурс] – Электронные данные. – Режим доступа: <https://intellect.ml/metodologiya-idef0-funktsionalnoe-modelirovanie-biznes-protsessov-7318>

[6] Habr.com [Электронный ресурс] – Электронные данные. – Режим доступа: <https://habr.com/company/trinion/blog/322832/>

[7] Wikipedia [Электронный ресурс] – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/ERwin\_Data\_Modeler

[8] Alexvolkov.ru [Электронный ресурс] – Электронные данные. – Режим доступа: http://alexvolkov.ru/database-normalization-third-normal-form.html

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг алгоритмов, реализующих бизнес-логику**

//Класс работы с данными о записи в парикмахерскую

package control;

import beatysalon.Record;

import beatysalon.Client;

import servercp.ServerWork;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.Map;

public class InsertRecordTable {

private ServerWork sw;

private DateBase mdbc;

private java.sql.Statement stmt;

private String query;

private Statement statement;

private java.sql.PreparedStatement preparedStmt;

public Map<String, String> stringmap;

private Record record;

private Client client;

public InsertRecordTable(ServerWork sw) {

try {

this.sw = sw;

mdbc = new DateBase();

Connection conn = mdbc.getConnection();

stmt = conn.createStatement();

} catch (Exception e) {

System.out.println("InsertRecordTable = " + e);

}

}

public String quotate(String content) {

return " '" + content + "' ";

}

public void AddClientButtonActionPerformed() throws IOException, ClassNotFoundException, SQLException {

client = new Client();

query = "INSERT INTO clients VALUES(NULL,?,?,?)";

preparedStmt = mdbc.getConnection().prepareStatement(query);

stringmap = sw.receiveStringMap();//считывание данных потока ввода

client.stringmap(stringmap);

Продолжение приложения А

sw.serverMessage("stringmap recieved! ");

preparedStmt.setString(1, client.getClientSurname());

preparedStmt.setObject(2, client.getClientName());

preparedStmt.setString(3, client.getClientPhoneNumber());

preparedStmt.execute();

}

public void AddRecordButtonActionPerformed() throws IOException, ClassNotFoundException, SQLException {

record = new Record();

query = "INSERT INTO records VALUES(NULL,?,?,?,?,?)";

preparedStmt = mdbc.getConnection().prepareStatement(query);

stringmap = sw.receiveStringMap();//считывание данных потока ввода

record.stringmap(stringmap);

sw.serverMessage("stringmap recieved! ");

preparedStmt.setString(1, record.getClientSurname());

preparedStmt.setString(2, record.getServiceName());

preparedStmt.setString(3, record.getEmployeeSurname());

preparedStmt.setString(4, record.getRecordDate());

preparedStmt.setString(5, record.getRecordTime());

preparedStmt.execute();

}

public void DeleteClientButtonActionPerformed() throws IOException, ClassNotFoundException {

sw.stringmap = sw.receiveStringMap();

String ClientID = sw.stringmap.get("ClientID");

String insertStr = "";

try {

insertStr = "DELETE FROM clients WHERE ClientID=" + ClientID;

int done = stmt.executeUpdate(insertStr);

} catch (SQLException e) {

System.out.println("delete = " + e);

sw.sendMessage("Error occured in deleting data: " + e);

}

}

public void DeleteRecordButtonActionPerformed() throws IOException, ClassNotFoundException {

sw.stringmap = sw.receiveStringMap();

String RecordID = sw.stringmap.get("RecordID");

String insertStr = "";

try {

insertStr = "DELETE FROM records WHERE RecordID=" + RecordID;

int done = stmt.executeUpdate(insertStr);

} catch (SQLException e) {

Продолжение приложения А

System.out.println("delete = " + e);

sw.sendMessage("Error occured in deleting data: " + e);

}

}

public void EditClientButtonActionPerformed() {

sw.stringmap = sw.receiveStringMap();

String ClientID = sw.stringmap.get("ClientID");

String ClientSurname = sw.stringmap.get("ClientSurname");

String ClientName = sw.stringmap.get("ClientName");

String ClientPhoneNumber = sw.stringmap.get("ClientPhoneNumber");

try {

String insertStr = "UPDATE clients SET ClientSurname ="

+ quotate(ClientSurname) + "WHERE ClientID =" + ClientID;

int done = stmt.executeUpdate(insertStr);

insertStr = "UPDATE clients SET ClientName ="

+ quotate(ClientName) + "WHERE ClientID =" + ClientID;

done = stmt.executeUpdate(insertStr);

insertStr = "UPDATE clients SET ClientPhoneNumber ="

+ quotate(ClientPhoneNumber) + "WHERE ClientID =" + ClientID;

done = stmt.executeUpdate(insertStr);

} catch (SQLException e) {

sw.sendMessage("Error occured in editing data:" + e);

}

}

public void EditRecordButtonActionPerformed() {

sw.stringmap = sw.receiveStringMap();

String RecordID = sw.stringmap.get("RecordID");

String ClientSurname = sw.stringmap.get("ClientSurname");

String ServiceName = sw.stringmap.get("ServiceName");

String EmployeeSurname = sw.stringmap.get("EmployeeSurname");

String RecordDate = sw.stringmap.get("RecordDate");

String RecordTime = sw.stringmap.get("RecordTime");

try {

String insertStr = "UPDATE records SET ClientSurname="

+ quotate(ClientSurname) + "WHERE RecordID =" + RecordID;

int done = stmt.executeUpdate(insertStr);

insertStr = "UPDATE records SET ServiceName="

+ quotate(ServiceName) + "WHERE RecordID =" + RecordID;

done = stmt.executeUpdate(insertStr);

insertStr = "UPDATE records SET EmployeeSurname="

+ quotate(EmployeeSurname) + "WHERE RecordID =" + RecordID;

done = stmt.executeUpdate(insertStr);

insertStr = "UPDATE records SET RecordDate="

+ quotate(RecordDate) + "WHERE RecordID =" + RecordID;

done = stmt.executeUpdate(insertStr);

insertStr = "UPDATE records SET RecordTime="

+ quotate(RecordTime) + "WHERE RecordID =" + RecordID;

done = stmt.executeUpdate(insertStr);

} catch (SQLException e) {

Продолжение приложения А

sw.sendMessage("Error occured in editing data:" + e);

}

}

public void getResultFromClient() {

ResultSet rs = null;

try {

rs = stmt.executeQuery("SELECT \* FROM clients");

ArrayList<String[]> resultSets = new ArrayList<>();

while (rs.next()) {

String[] row = {

rs.getString("ClientID"),

rs.getString("ClientSurname"),

rs.getString("ClientName"),

rs.getString("ClientPhoneNumber")

};

resultSets.add(row);

}

sw.sendResultSet(resultSets);

rs.close();

} catch (SQLException e) {

System.out.println("getResultFromClient = " + e);

}

}

public void getResultFromRecord() {

ResultSet rs = null;

try {

rs = stmt.executeQuery("SELECT \* FROM records");

ArrayList<String[]> resultSets = new ArrayList<>();

while (rs.next()) {

String[] row = {

rs.getString("RecordID"),

rs.getString("ClientSurname"),

rs.getString("ServiceName"),

rs.getString("EmployeeSurname"),

rs.getString("RecordDate"),

rs.getString("RecordTime")

};

resultSets.add(row);

}

sw.sendResultSet(resultSets);

rs.close();

} catch (SQLException e) {

System.out.println("getResultFromRecord = " + e);

}

}

Продолжение приложения А

public void getIDsFromClient() {

ResultSet rs = null;

try {

rs = stmt.executeQuery("SELECT ClientSurname FROM clients");

ArrayList<String[]> resultSets = new ArrayList<>();

while (rs.next()) {

String[] row = {

rs.getString("ClientSurname")

};

resultSets.add(row);

}

sw.sendResultSet(resultSets);

rs.close();

} catch (SQLException e) {

System.out.println("getIDsFromClient = " + e);

}

}

public void getIDsFromService() {

ResultSet rs = null;

try {

rs = stmt.executeQuery("SELECT ServiceName FROM service");

ArrayList<String[]> resultSets = new ArrayList<>();

while (rs.next()) {

String[] row = {

rs.getString("ServiceName")

};

resultSets.add(row);

}

sw.sendResultSet(resultSets);

rs.close();

} catch (SQLException e) {

System.out.println("getIDsFromService = " + e);

}

}

public void getIDsFromEmployee() {

ResultSet rs = null;

try {

rs = stmt.executeQuery("SELECT EmployeeSurname FROM employees");

ArrayList<String[]> resultSets = new ArrayList<>();

while (rs.next()) {

String[] row = {

rs.getString("EmployeeSurname")

};

Продолжение приложения А

resultSets.add(row);

}

sw.sendResultSet(resultSets);

rs.close();

} catch (SQLException e) {

System.out.println("getIDsFromEmployee = " + e);

}

}

}

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Листинг основных элементов программы**

//Класс реализующий работу сервера

package servercp;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import static java.lang.Thread.NORM\_PRIORITY;

import java.net.ServerSocket;

import java.net.Socket;

public class ServerCP extends Thread {

private static ServerSocket serverSocket;

private final Socket clientAccepted;

public static ObjectInputStream sois;

public static ObjectOutputStream soos;

private static int numOfClients;

public static void main(String[] arg) {

sois = null;//объявление байтового потока ввода

soos = null;//объявление байтового потока вывода

serverSocket = null; //объявление объекта класса Socket

try {

ServerWork.maxClients = 10;

System.out.println("server starting....");

serverSocket = new ServerSocket(2525);//создание сокета сервера

System.out.println("server is started");

for (int i = 0; i < arg.length; i++) {

if (arg[i].equals("max")) {

ServerWork.maxClients = Integer.parseInt(arg[i + 1]);

}

if (arg[i].equals("logging")) {

System.out.println("logging is on");

ServerWork.logging = true;

}

}

System.out.println("maxClients: " + ServerWork.maxClients);

while (true) {

// ждём нового подключения, после чего запускаем обработку клиента

// в новый вычислительный поток и увеличиваем счётчик на единичку

ServerCP serverCP = new ServerCP(serverSocket.accept());

Продолжение приложения Б

}

} catch (IOException | NumberFormatException e) {

System.out.println("init error: " + e);

} // вывод исключений

finally {

try {

serverSocket.close();//закрытие сокета сервера

} catch (IOException e) {

//вызывается метод исключения е

}

}

}

public ServerCP(Socket s) {

// копируем данные

this.clientAccepted = s;

numOfClients++;

System.out.println("client '" + numOfClients + "' is joined");

// и запускаем новый вычислительный поток (см. ф-ю run())

setDaemon(true);

setPriority(NORM\_PRIORITY);

start();

}

@Override

public void run() {

try {

// из сокета клиента берём поток входящих данных

sois = new ObjectInputStream(clientAccepted.getInputStream());

// и оттуда же - поток данных от сервера к клиенту

soos = new ObjectOutputStream(clientAccepted.getOutputStream());

ServerWork serverWork = new ServerWork(sois, soos, numOfClients);

serverWork.runServerWork();

System.out.println("client '" + numOfClients + "' disconnected");

numOfClients--;

} catch (IOException e) {

System.out.println("init error: " + e);

}

}

}

//Класс реализующий подключение базы данных

package control;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DateBase extends Configs {

Продолжение приложения Б

private Connection connection;

public Connection getConnection() throws ClassNotFoundException, SQLException, IOException {

getConfigs();

String connectionString = "jdbc:mysql://" + dbHost + ":"

+ dbPort + "/" + dbName;

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

connection = DriverManager.getConnection(connectionString, dbUser, dbPass);

return connection;

}

}

//Класс реализующий конфигурации данных для бд

package control;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

public class Configs {

protected String dbHost; //= "localhost"

protected String dbPort; //= "3306"

protected String dbUser; //= "root"

protected String dbPass; //= "08210911ijMuM"

protected String dbName; //= "salon"

/\*\*

\*

\*/

protected void getConfigs() throws FileNotFoundException, IOException {

FileReader reader = new FileReader("src/control/config.txt");

Scanner scanner = new Scanner(reader);

this.dbHost = scanner.nextLine();

this.dbPort = scanner.nextLine();

this.dbUser = scanner.nextLine();

this.dbPass = scanner.nextLine();

this.dbName = scanner.nextLine();

reader.close();

}

}

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Листинг скрипта генерации базы данных**

CREATE SCHEMA `salon` DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci ;

CREATE TABLE `salon`.`salons` (

`SalonID` INT NOT NULL AUTO\_INCREMENT,

`SalonName` VARCHAR(45) NOT NULL,

`SalonCity` VARCHAR(45) NULL,

`SalonLocation` VARCHAR(45) NULL,

`SalonWebSite` VARCHAR(60) NOT NULL,

`SalonPhoneNumber` INT NOT NULL,

PRIMARY KEY (`SalonID`) ,

UNIQUE INDEX `SalonID\_UNIQUE` (`SalonID` ASC) ,

UNIQUE INDEX `SalonName\_UNIQUE` (`SalonName` ASC)

);

CREATE TABLE `salon`.`clients` (

`ClientID` INT NOT NULL AUTO\_INCREMENT,

`ClientSurname` VARCHAR(45) NOT NULL,

`ClientName` VARCHAR(45) NULL,

`ClientPhoneNumber` INT NOT NULL,

PRIMARY KEY (`ClientID`, `ClientSurname`),

UNIQUE INDEX `ClientID\_UNIQUE` (`ClientID` ASC),

UNIQUE INDEX `ClientSurname\_UNIQUE` (`ClientSurname` ASC)

);

CREATE TABLE `salon`.`service` (

`ServiceID` INT NOT NULL AUTO\_INCREMENT,

`ServiceName` VARCHAR(45) NOT NULL,

`ServiceCost` DOUBLE NOT NULL,

`ServiceType` VARCHAR(45) NOT NULL,

PRIMARY KEY (`ServiceID`, `ServiceName`),

UNIQUE INDEX `ServiceID\_UNIQUE` (`ServiceID` ASC),

UNIQUE INDEX `ServiceName\_UNIQUE` (`ServiceName` ASC)

);

CREATE TABLE `salon`.`employees` (

`EmployeeID` INT NOT NULL AUTO\_INCREMENT,

`SalonID` INT NOT NULL,

`EmployeeSurname` VARCHAR(45) NOT NULL,

`EmployeeName` VARCHAR(45) NULL,

PRIMARY KEY (`EmployeeID`, `EmployeeSurname`) ,

UNIQUE INDEX `EmployeeID\_UNIQUE` (`EmployeeID` ASC) ,

UNIQUE INDEX `EmployeeSurname\_UNIQUE` (`EmployeeSurname` ASC) ,

INDEX `fx\_employees\_salons\_idx` (`SalonID` ASC) ,

CONSTRAINT `fx\_employee\_salons` FOREIGN KEY (`SalonID`)

REFERENCES `salon`.`salons` (`SalonID`)

Продолжение приложения В

ON DELETE NO ACTION ON UPDATE NO ACTION

);

CREATE TABLE `salon`.`users` (

`UserID` INT NOT NULL AUTO\_INCREMENT,

`EmployeeID` INT NULL,

`UserLogin` VARCHAR(45) NOT NULL,

`UserPassword` VARCHAR(45) NOT NULL,

`UserRigths` ENUM('admin', 'user') NOT NULL DEFAULT 'user',

PRIMARY KEY (`UserID`) ,

UNIQUE INDEX `UserLogin\_UNIQUE` (`UserLogin` ASC),

INDEX `fx\_users\_employees\_idx` (`EmployeeID` ASC),

CONSTRAINT `fx\_users\_employees` FOREIGN KEY (`EmployeeID`)

REFERENCES `salon`.`employees` (`EmployeeID`)

ON DELETE NO ACTION ON UPDATE NO ACTION

);

CREATE TABLE `salon`.`records` (

`RecordID` INT NOT NULL AUTO\_INCREMENT,

`ClientSurname` INT NOT NULL,

`ServiceName` INT NOT NULL,

`EmployeeSurname` INT NOT NULL,

`RecordDate` DATE NOT NULL,

`RecordTime` TIME NOT NULL,

PRIMARY KEY (`RecordID`) ,

UNIQUE INDEX `RecordID\_UNIQUE` (`RecordID` ASC) ,

INDEX `fx\_records\_clients\_idx` (`ClientSurname` ASC) ,

INDEX `fx\_records\_service\_idx` (`ServiceName` ASC) ,

INDEX `fx\_records\_employees\_idx` (`EmployeeSurname` ASC) ,

CONSTRAINT `fx\_records\_clients` FOREIGN KEY (`ClientSurname`)

REFERENCES `salon`.`clients` (`ClientSurname`)

ON DELETE CASCADE ON UPDATE NO ACTION,

CONSTRAINT `fx\_records\_service` FOREIGN KEY (`ServiceName`)

REFERENCES `salon`.`service` (`ServiceName`)

ON DELETE CASCADE ON UPDATE NO ACTION,

CONSTRAINT `fx\_records\_employees` FOREIGN KEY (`EmployeeSurname`)

REFERENCES `salon`.`employees` (`EmployeeSurname`)

ON DELETE NO ACTION ON UPDATE NO ACTION

);