

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ УТ

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление

09.03.02 «Информационные системы и технологии»

подготовки

БАКАЛАВРСКАЯ РАБОТА

на тему

**«Web-приложение салона красоты с локальным
хранилищем на стороне клиента»**

Студент

(подпись, дата) Духачева Анастасия Дмитриевна
(ФИО полностью)

Руководитель

(подпись, дата) Макарычев П.П.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата) Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись) Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись) Попова Н.А.
(фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Кафедра

Вычислительной техники

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

А.Ю. Козлов

«25» декабря 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Духачева Анастасия Дмитриевна гр. 20ВИ1 факультета вт
<Фамилия, имя, отчество полностью>
2. Тема работы «Web-приложение салона красоты с локальным хранилищем на стороне клиента»
- Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____
2. Руководитель работы Макарычев П.П.
3. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать web-приложение учета и анализа услуг для информационной системы салона красоты. Среда и средства разработки: C#, ASP.NET, .NET Framework 4.6, MySqlLeech, MySql + phpMyAdmin. Среда функционирования: Windows 7 и выше. Основные функции: - учет услуг, - учет реализации услуг, - прогноз продажи услуг, - ведение аналитики по реализации услуг специалистами. Особенности (особые функции, особые требования): Бесплатное ПО, наличие API для получение информации об свободных записях, изменении количества записей, добавления новых записей; многопользовательское приложение

4. Перечень подлежащих разработке вопросов.

1. Анализ предметной области и постановка задачи.

2. Анализ требований на разработку.

3. Разработка программных средств.

4. Тестирование программных средств.

5. Оформление ВКР.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области и постановка задачи	15%	08.04.2024-13.04.2024	
2.	Анализ требований на разработку	20%	15.04.2024-25.04.2024	
3.	Разработка программных средств	30%	26.04.2024-12.05.2024	
4.	Тестирование программных средств	20%	13.05.2024-22.05.2024	
5.	Оформление ВКР	15%	23.05.2024-03.06.2024	

Дата выдачи задания "21" декабря 2023г.

Руководитель бакалаврской работы _____

(подпись, дата)

Задание к исполнению принял студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____

(подпись, дата)

Реферат

Пояснительная записка содержит 78 листов, 18 рисунков, 5 таблиц, 15 использованных источников, 7 листингов и 3 приложения.

АИС, ASP.NET, VISUAL STUDIO 2022, C#, УЧЕТ РАСПИСАНИЯ, ИНФОРМАЦИОННАЯ СИСТЕМА.

Предмет работы: разработка информационной системы автоматизации работы салона красоты.

Объект исследования: веб приложение, обеспечивающее деятельность салона красоты.

Цель работы: разработать информационную систему, которая автоматизирует деятельность салона красоты.

Средства разработки: C#, ASP.NET, .NET Framework 4.8, MySqlGDB, JSONExtended, MySql.

Результаты работы: спроектировано, разработано, отлажено и протестировано веб приложение учета и анализа работы для информационной системы салона красоты.

Область применения: салоны красоты, занимающиеся оказанием услуг в beauty сфере.

					ПГУ 09.03.02 – 08ВР201.06 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Web-приложение салона красоты с локальным хранилищем на стороне клиента» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Духачева А.Д.					4	78
Провер.		Макарычев П.П.				группа 20ВИ1		
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	11
1 Аналитическая часть.....	12
1.1 Анализ предметной области	12
1.1.1 Требования к составу и параметрам технических средств.....	13
1.1.2 Описание функциональности контроля работы салона.....	14
1.2 Анализ информационных систем аналогичного назначения.....	14
1.2.1 «Точка красоты»	14
1.2.2 «Пальчики»	15
1.2.3 «Cocos Studio»	16
1.2.4 Обоснование для разработки веб приложения системы работы салона красоты.....	18
1.3 Анализ требований к программной реализации.....	21
1.3.1 Список функциональных требований.....	21
1.3.2 Список нефункциональных требований.....	24
1.3.3 Требования к надежности	24
1.3.4 Условия эксплуатации.....	24
2 Проектирование программного обеспечения	25
2.1 Проектирование пользовательского интерфейса	25
2.2 Анализ бизнес процессов организации в нотации IDEF0	27
2.3 Проектирование модели базы данных.....	28
2.3.1 Проектирование физической и инфологической моделей базы данных с помощью Erwin	28
2.4 Проектирование поведения ИС.....	30
2.4.1 Диаграмма состояний	30
2.4.2 Диаграмма деятельности.....	31
2.4.3 Диаграмма последовательности	31
2.4.4 Диаграмма коммуникаций	32

2.4.5	Диаграмма компонентов	33
2.4.6	Диаграмма развертывания.....	33
3	Разработка приложения.....	35
3.1	Программные средства.....	35
4	Отладка и тестирование	38
4.1	Тестирование программного обеспечения.....	38
4.1.1	Метрики программного средства.....	42
4.1.2	Расчет метрик кода с помощью встроенного средства Visual Studio	42
4.1.3	Определение метрик компонента серверной части приложения.....	43
	Заключение	46
	Список использованных источников	47
	Приложение А (рекомендуемое) Диаграммы UML	49
	Приложение Б (рекомендуемое) Листинг программы	56
	Приложение В (рекомендуемое) Результаты тестирования	77

Введение

Эффективность работы салона красоты зависит напрямую от качества связи с клиентом. Успех ведения бизнеса в этой сфере зависит не только от профессионализма мастеров, а от доступности информации о сервисах, удобства записи, взаимодействия с салонами. В процессе этого процесса играет ключевая роль интернет-платформа, позволяющая салонам красоты эффективно позиционироваться на рынок и строить долгосрочные взаимоотношения с клиентом.

Разработка веб приложения для салонов красоты – современный способ взаимодействия клиентов и передачи им актуальных сведений о услугах и специалистах салона.

Интернет становится неотъемлемым частью современной жизни, а поиск информации о услугах салона красоты чаще всего начинается в интернете. Первые впечатления от салона формируют потенциальный клиент, когда он знакомится с его веб-приложением или приложениями.

Разработка Web-приложения по поиску услуг и специалистов салонов красоты, а также дальнейшей записи на прием, является важным шагом на пути повышения лояльности клиентов, а также автоматизации бизнес-процессов. Приложение предназначено для того, чтобы клиенты могли получить удобный инструмент записи в интернете, ознакомиться с перечень услуг, цены, информацию о специалистах, получить специальные предложения и акции.

Функции приложения могут включать предоставление электронного каталога услуг, фото специалистов, просмотр посещений.

Особенность работы салона красоты заключается в широком ассортименте услуг, который должен быть структурирован и представлен пользователю удобно. Важно предусматривать возможность фильтра услуг по разным характеристикам: тип, классификации, стоимость, специалист.

1 Аналитическая часть

1.1 Анализ предметной области

Администратор веб приложения выполняет функцию управляющего салоном красоты, контролирует актуальность информации о услугах и специалистах и взаимодействует с клиентами по системе онлайн записи.

Web-приложение предназначено для того, чтобы предоставлять клиентам актуальные сведения о услугах салонов красоты, их мастерах, графике работы и осуществлять онлайн запись и обратную связь.

Разработанное приложение предназначено для двух групп пользователей:

- клиенты салонов красоты, получающие возможность познакомиться с ассортиментом услуг, подобрать подходящий мастер, записаться в удобное время и оставить отзыв на посещение;
- администратор салон красоты, получающий инструмент для контроля за информацией о сервисах и специалистов, отслеживает записи и ведёт базу клиентов.

В системе информационной поддержки салонов красоты должны быть реализованы следующие основные функции:

- контроль каталога услуг, где система дает возможность добавлять, изменять и удалить информацию о салонных услугах, в том числе о их описании, стоимости, длительности и фотографиях работ;
- контроль профилей специалистов, где предлагается создание и редактирование профилей специалистов с указанием своей специализации, профессионального опыта, графики работы и возможностью добавление фотографий работ, отзывов клиента;
- система онлайн записи, где клиент может самостоятельно выбрать услугу, исполнителя, дату посещения и время проведения записи, осуществить запись онлайн, после чего получить подтверждение и напоминание.

Основные цели разработки web-приложения:

- создание единой информационной среды взаимодействия салонов красоты и клиентов;
- предлагает клиентам удобный инструмент для записи и получения информации о сервисах;
- повышение эффективности работы администратора салона при приеме заявок, ведении графика и взаимодействия с клиентом.

Внедрение этой системы позволит:

- повысить лояльность клиентов благодаря удобству и доступной информации;
- увеличение количества записей, благодаря упрощенному процессу оформления брони;
- оптимизация работы администраторов в салоне, освобождение их от рутины.

1.1.1 Требования к составу и параметрам технических средств

Для работы с Web-приложением рекомендуются следующие программные и технические средства:

- операционная система семейства Windows/Linux;
- современный браузер (рекомендуются Google Chrome, Firefox);
- одноядерный процессор Intel или совместимый с частотной ядра не менее 500МГц;
- объем свободной оперативной памяти 512 Мб;
- объем необходимой памяти на жестком диске 500 Мб;
- стандартный VGA-монитор или совместимый, с поддерживаемым разрешением от 1024x768;
- 3D-ускоритель с объемом оперативной памяти больше 32 Мб;
- стандартная клавиатура;
- манипулятор «мышь».

1.1.2 Описание функциональности контроля работы салона

В настоящее время существует много информационных систем контроля работы салона красоты.

Внедрение автоматизированной информационной системы позволит:

- снизить случайные ошибки связанных с ручным отслеживанием записей;
- автоматизировать обработку информации в системе;
- получить доступ к списку записей к специалисту из любой точки, где есть компьютер, имеющий выход в Интернет.

1.2 Анализ информационных систем аналогичного назначения

1.2.1 «Точка красоты»

Точка красоты – специальная система, которая предназначена для автоматизации ключевых деловых процессов в косметических салонах и центрах красоты. Платформа предлагает инструменты управления сервисом, ведения базы клиентов, контроля за финансами и оценки эффективности деятельности. Внедрение точки красоты дает возможность салонам улучшить качество сервиса, оптимизировать затраты и повысить доход.

Функции управления сервисами:

- создать и редактировать каталог услуг, содержащий подробное описание, указание цены и длительности;
- формирование листов скидок с гибкими системами акций и скидок;
- ведение графики работы мастеров, с возможностью записи клиентов онлайн.

Функции работы с заказчиками:

- создать и вести единую базу клиентов, содержащую историю посещения и личные данные;

- сегментация базы клиентов по разным параметрам, чтобы провести таргетированные маркетинговые кампании;

- возможность отправлять SMS и электронные письма с информацией о акциях, спецпредложениях и напоминаниями о записях.

Финансовая аналитика и финансовые показатели:

- контроль за доходами и расходами салона с формированием подробного отчета;

- расчет зарплаты сотрудников в зависимости от различных мотивационных систем;

- анализируйте эффективность работы салона в соответствии с ключевыми показателями KPI.

Система «Точка красоты» является эффективным решением автоматизации работы косметических салонов и центров красоты, направленным на повышения эффективности, повышения прибыли, формирование клиентской лояльности. Однако, прежде чем внедрить эту систему, необходимо внимательно анализировать потребности в салоне и оценивать экономическую целесообразность ее применения, так как система не является дешевой в использовании.

1.2.2 «Пальчики»

«Пальчики» – собственная система контроля, разработана и реализована в известной салонной сети. Система поможет оптимизировать бизнес-процессы, централизованно управлять и контролировать деятельность всей салонной сети. Однако, несмотря на многочисленные преимущества, система для пальцев, как и всякая другая, обладает своими недостатками.

Функции управления сервисами и сотрудниками:

- создать и редактировать каталог услуг, в котором указаны стоимость, длительность, необходимые материалы;

- формирование профиля мастера с указанием квалификации, опыт работы, график работы, фотографии работ;
- анализ популярности услуг и специалистов, мониторинг сезонных трендов;
- планирование работы персонала, контроль загрузки работников.

Функции управления записями на услуги:

- онлайн запись клиентов на услуги с выбором оптимального времени, мастеров и салонов;
- контроль расписания салонов, контроля за загрузкой, предотвращения блокировок;
- автоматически напоминает клиенту о предполагаемой записи на веб-приложение.

Финансовая аналитика и финансовые показатели:

- управление финансовыми показателями салонной сети: выручкой, расходами, прибыли;
- анализирующая эффективность работы отдельных офисов и сетей в целом;
- формирование отчетов по реализации услуг, работам мастеров и популярности сервисов.

Система контроля «Пальчики» является примером специального решения, которое создано под конкретные нужды крупной салонной сети. Система эффективно управляет ключевыми процессами бизнеса, но обладает и своими недостатками, характерными для подобных систем.

1.2.3 «Cocos Studio»

Компания Cocos Studio, сеть косметических салонов, которые предлагают услуги и косметику класса люкса, использует свою систему контроля, адаптированную к специфике работы с премиальными сегментами.

Функции контроля ассортимента и поставщиков:

- система предназначена для работы с ограниченным спектром премиальных марок, в том числе нишевых и редких марок;
- система обеспечивает оперативное реагирование на появление новых услуг и приобретение эксклюзивных партий услуг;
- обеспечивает достоверность и качество продукции и позволяет получить выгодное сотрудничество;
- система обеспечивает соблюдение особых условий хранения и доставки премиальных косметики, чтобы сохранить ее качество.

Работа с заказчиками и повышение лояльности:

- дает возможность вести подробную базу клиента, следить за его предпочтениями, историей посещений, предоставить персональный сервис;
- система дает возможность планировать и проводить презентации новых услуг, семинары с участием ведущих экспертов, другие мероприятия для VIP клиентов.

Финансовые контрольные и аналитические работы:

- контроль бюджета с учетом высоких стоимостей услуг: система эффективно планирует расходы на покупку премиальных косметических средств;
- аналитическая рентабельность и продажи: дает возможность оценить эффективность работы различных брендов и услуг, корректировку ассортиментной политики.

Системы управления Cocos Studio являются примером адаптации ИТ-решений под конкретные бизнес-задания. Система обеспечивает успешное позиционирование Cocos Studio в сегменте премиум и обеспечение высокого уровня услуг. Однако, чтобы сохранить конкурентные преимущества, необходимо постоянно совершенствовать и развивать систему с учетом динамики рынка, изменений потребительского спроса.

1.2.4 Обоснование для разработки клиент серверного приложения системы работы салона красоты

В таблице 1 представлена сравнительная характеристика рассмотренных систем работы салона красоты, таких как: «Точка красоты» – сеть салонов красоты, «Пальчики» – сеть доступных салонов красоты и «Cocos Studio» – премиальная сеть салонов красоты.

Таблица 1 – Сравнительная характеристика рассмотренных систем

Характеристика	Точка красоты	Пальчики	Cocos Studio
1	2	3	4
Целевая аудитория	Сеть салонов красоты	Крупная сеть салонов красоты	Премиальная сеть салонов красоты
Управление ассортиментом услуг	Базовое управление ассортиментом, категоризация услуг, анализ продаж	Расширенное управление ассортиментом, анализ сезонных трендов, фокус на популярность услуг	Фокус на селективный ассортимент премиальных и нишевых брендов материалов необходимых для оказания услуг, работа с эксклюзивными контрактами
Управление работой сотрудников	Создание и отправка заказов, отслеживание статуса, ведение базы специалистов, сравнение цен	Формирование заказов с учетом потребностей салонов, отслеживание статуса, ведение базы специалистов, сравнение цен	Прямое сотрудничество с сторонними специалистами, строгий контроль качества оказываемых услуг и квалификации специалистов

Продолжение таблицы 1

1	2	3	4
Контроль расходов	Установка бюджетов, мониторинг расходов по категориям, анализ эффективности работы	Установка бюджетов для каждого салона и сети, мониторинг расходов, анализ эффективности работы	Управление бюджетом с учетом высокой стоимости материалов, анализ эффективности работы с фокусом на рентабельность и оборачиваемость
Дополнительные функции	Интеграция с кассовыми системами, генерация отчетов, работа с несколькими салонами	Интеграция с кассовыми системами и программами лояльности, генерация отчетов	Интеграция с CRM-системой, организация мероприятий и презентаций
Преимущества	Оптимизация работы, контроль расходов, улучшение ассортимента	Эффективное управление записями для большой сети	Экспертиза в премиальном сегменте, эксклюзивный ассортимент, п
Недостатки	Ограниченный функционал работы системы	Сложность системы, недостаточная гибкость, интеграция с другими системами	Высокая стоимость работы, сложность настройки, риск неликвидов
Стоимость	Средняя	Высокая	Очень высокая
Гибкость	Низкая	Невысокая	Высокая
Интеграция	Отсутствует	Проблемы с интеграцией	Интеграция с различными сервисами

В рассмотренных аналогичных системах после анализа выявлено следующее:

- внешний вид и устройство сайтов не позволяют пользователю сходу освоиться и начать записываться на услуги;
- для созданий новых записей зачастую нужна регистрация пользователя и создание личного кабинета;
- устаревшая система контроля над временем сотрудников.

Лучшим решением выше выявленных сложностей является разработка автоматизированной информационной системы с использованием технологии ASP.NET [1], потому что ASP.NET – это один из самых популярных фреймворков для разработки сайтов. Инфраструктура ASP.NET MVC [5] представляет собой последнюю версию веб-платформы ASP.NET от Microsoft. У нее есть множество преимуществ, которых нет в классической платформе для веб-разработки ASP.NET Web Forms.

Встроенные методы представлений генерируют ясный и соответствующий стандартам код разметки в формате HTML. Таким образом ASP.NET MVC [5] предлагает мощную систему маршрутизации URL, позволяющую создавать удобные и приятные в смысловом плане URL-адреса. Компоненты ASP.NET MVC расширяемы и тестируемы, она обеспечивает расширенную поддержку разработки через тестирование. Огромным преимуществом является также мощная полнофункциональная среда разработки – Microsoft Visual Studio.

Для платформы .NET на сегодняшний день существуют сотни тысяч бесплатных библиотек и компонентов, расширяющих функционал стандартного языка, доступных для скачивания. Также присутствует удобная система документации и база данных чуть ли не по всем вопросам, которые могут возникнуть в процессе разработки, что так же является огромный плюсом в сторону данного инструмента разработки.

Ожидаемый результат разработки системы:

- современность – web приложение, разработанное с помощью современных инструментов разработки;
- понятный интерфейс – имеется ввиду то, что пользователь сможет без особых усилий понять, как функционируют элементы системы и для чего они нужны;
- комфорт – пользователь сможет просматривать каталог услуг и оформлять заявку на запись без необходимости совершения лишних действий, в том числе и регистрации;
- непротиворечивость и достоверность данных;
- низкая стоимость – так же является неоспоримым преимуществом.

1.3 Анализ требований к программной реализации

Спроектированная и полностью разработанная АИС [2] является целью выпускной квалификационной работы.

После разработки и последующего внедрения АИС предполагаются следующие достоинства:

- улучшение эффективности выполнения функций;
- неизменный контроль над процессом формирования и управление рабочим расписанием специалистов салона;
- увеличение количества категорий услуг;
- добавление информационных и справочных разделов.

1.3.1 Список функциональных требований

Автоматизированная информационная система работы салона красоты должна выполнять необходимый функционал:

- ведение базы данных (запись, чтение, модификация, удаление);
- хранение информации о расписании специалистов;
- хранение информации о услугах.

Для описания поведения и определения функциональных требований разрабатываемой системы существует UML диаграмма прецедентов, состоящая из актеров (действующих лиц), которые взаимодействуют с системой по средствам вариантов использования.

Диаграмма вариантов использования представлена на рисунке 1.

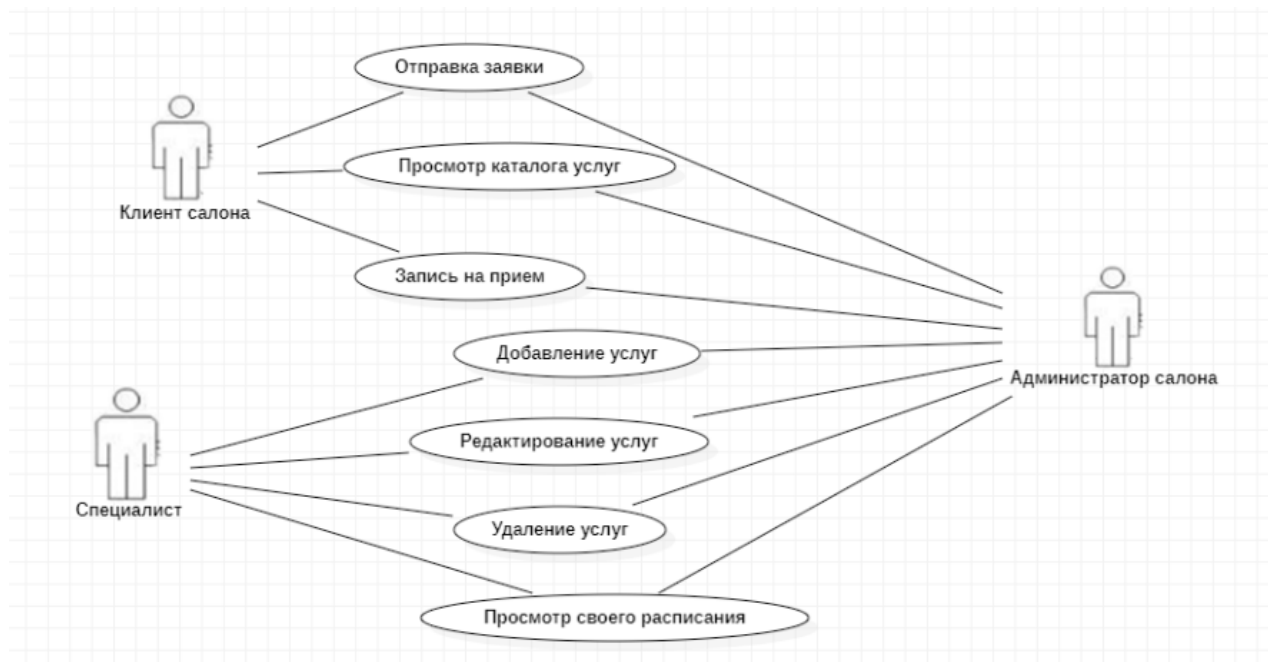


Рисунок 1 – Диаграмма вариантов использования

Описание действующих лиц – клиент салона, специалист и администратор, представлено на таблице 2, а краткое описание вариантов использования – добавление услуг, редактирование услуг, удаление услуг, просмотр своего расписания, на таблице 3. Также в приложении А приведена диаграмма с использованием включений, обобщений.

Таблица 2 – Описание действующих лиц

Название	Профиль, подготовка и навыки
1	2
Администратор салона	Лицо, занимающееся администрированием системы контроля работы салона красоты. Может вносить изменения в работу приложения

Продолжение таблицы 2

1	2
Клиент салона	Лицо, имеющее или не имеющее опыт работы с данным приложением, ознакомлен с узким функционалом программы. Вносить изменения и корректировать работу приложения не может.
Специалист	Лицо, занимающееся администрированием системы контроля работы салона красоты. Может вносить изменения в узкую часть работы приложения

Таблица 3 – Описание вариантов использования

Действующее лицо	Цель	Краткое описание
Администратор	Отправка заявки	Позволяет администратору формировать заявку
Администратор	Просмотр каталога услуг	Позволяет просматривать каталог услуг
Администратор	Запись на прием	Позволяет записываться на прием
Специалист	Добавление услуг	Позволяет добавлять услуги
Специалист	Редактирование услуг	Позволяет редактировать услуги
Специалист	Удаление услуг	Позволяет удалять услуги
Специалист	Просмотр своего расписания	Позволяет просматривать график записи

Более подробно некоторые из прецедентов описаны в диаграммах UML в приложении А.

1.3.2 Список нефункциональных требований

Требования к временным характеристикам:

- время открытия главной страницы не должно быть более 20 с;
- время перехода между страницами не должно быть более 10 с.

1.3.3 Требования к надежности

Администрирование системы.

Доступ к админ-панели может быть получен только администратором путем его перехода к админ-панели по специальной ссылке и авторизации с помощью личного логина и пароля;

Необходимое время для восстановительных мер после отключения.

При отключении сервера, необходимо обеспечивать восстановление информации утраченной в результате сбоя в работе.

1.3.4 Условия эксплуатации

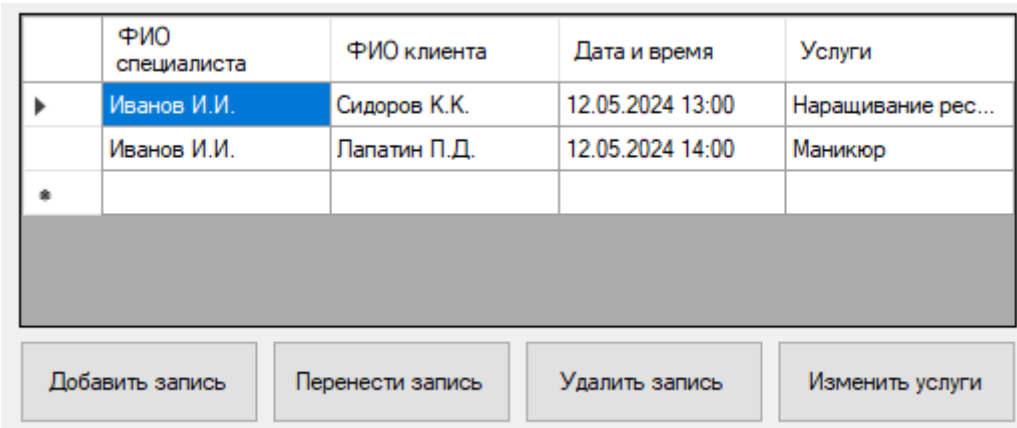
Для успешного функционирования данного веб приложения, необходима операционная система, современный web-браузер, СУБД MySQL, подключение к глобальной сети Интернет. Система должна быть ориентирована на простого пользователя с навыками работы в web-браузерах. Администрирование системы должно осуществляться лицом, ответственным за контроль работы салона красоты (например, администратор салона или менеджер), обладающими специальными знаниями в сфере администрирования информационных систем, с навыками работы в web-браузерах, знающим структуру и все функциональные возможности проектируемой и разрабатываемой информационной системы контроля работы салона красоты, а также знающего структуру работы салона красоты, предоставляемые в данном салоне.

2 Проектирование программного обеспечения

2.1 Проектирование пользовательского интерфейса

Пользовательский интерфейс – это совокупность программных средств, обеспечивающих взаимодействие пользователя с программным продуктом. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае можно понимать обмен информацией между человеком и программным средством, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи.

Пользовательский интерфейс информационной системы контроля работы салона красоты представлен ниже на рисунке 2.



	ФИО специалиста	ФИО клиента	Дата и время	Услуги
▶	Иванов И.И.	Сидоров К.К.	12.05.2024 13:00	Наращивание рес...
	Иванов И.И.	Лапатын П.Д.	12.05.2024 14:00	Маникюр
*				

Добавить запись
Перенести запись
Удалить запись
Изменить услуги

Рисунок 2 – Стартовая страница системы

Для администратора имеется собственная форма авторизации, позволяющая ему получить доступ к операциям создания/изменения/удаления расписания специалистов и услуг. Она включает в себя два поля, заполнив которые администратор может получить доступ к админ панели. Окно регистрации имеет аналогичные поля и служит для регистрации нового пользовательского аккаунта (административный аккаунт нельзя зарегистрировать). Вид окна регистрации представлен на рисунке 3.

2.2 Анализ бизнес процессов организации в нотации IDEF0

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

На контекстной диаграмме организации работы системы контроля работы салона красоты видно функционирование основной бизнес-функции:

- каталог услуг (вход);
- новая запись в БД (выход);
- список специалистов (управление);
- адрес администратора (управление);
- клиент (механизм);
- администратор (механизм).

Диаграмма бизнес-процессов в нотации IDEF0 [3] представлена на рисунке 5.

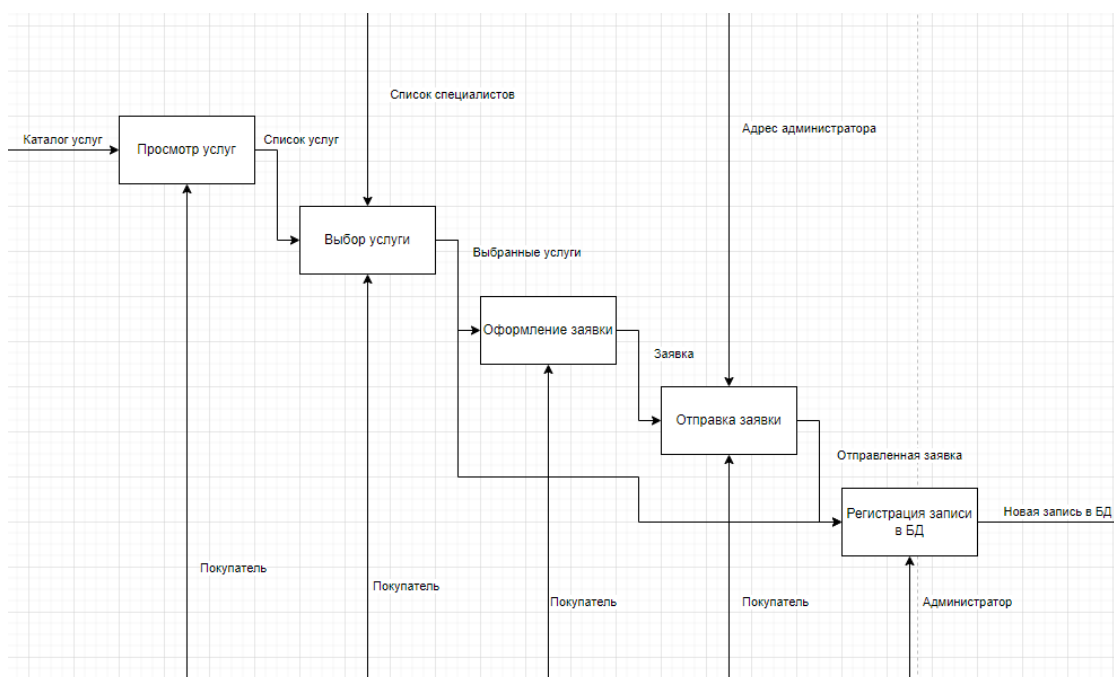


Рисунок 5 – Контекстная диаграмма деятельности по организации работы системы контроля работы салона красоты

При декомпозиции главной бизнес-функции выделяются следующие функциональные блоки: просмотр услуг, выбор услуги, оформление заявки, отправка заявки, регистрация записи в БД.

2.3 Проектирование модели базы данных

Для разработки моделей базы данных используется Case-средство ERwin и его методологию IDEF1X.

В ERwin существуют два уровня представления и моделирования – инфологический и физический.

2.3.1 Проектирование физической и инфологической моделей базы данных с помощью Erwin

Инфологический уровень – это общий взгляд на данные, на нем данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире. Объекты модели, представляемые на инфологическом уровне, называются сущностями и атрибутами. Инфологическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Создание модели данных, как правило, начинается с создания инфологической модели. После описания инфологической модели, человек, занимающийся проектированием может выбрать необходимую СУБД и ERwin [4] автоматически создаст соответствующую физическую модель. На основе физической модели ERwin может сгенерировать системный каталог СУБД или соответствующий SQL-скрипт [5], необходимый для создания базы данных.

В Case средстве ERwin [6] разработана инфологическая модель базы данных для хранения информации организации с учетом всех нюансов и специфики данной предметной области. Данная модель содержит семь таблиц, связанных между собой логическими связями. Инфологическая модель представлена на рисунке 6.

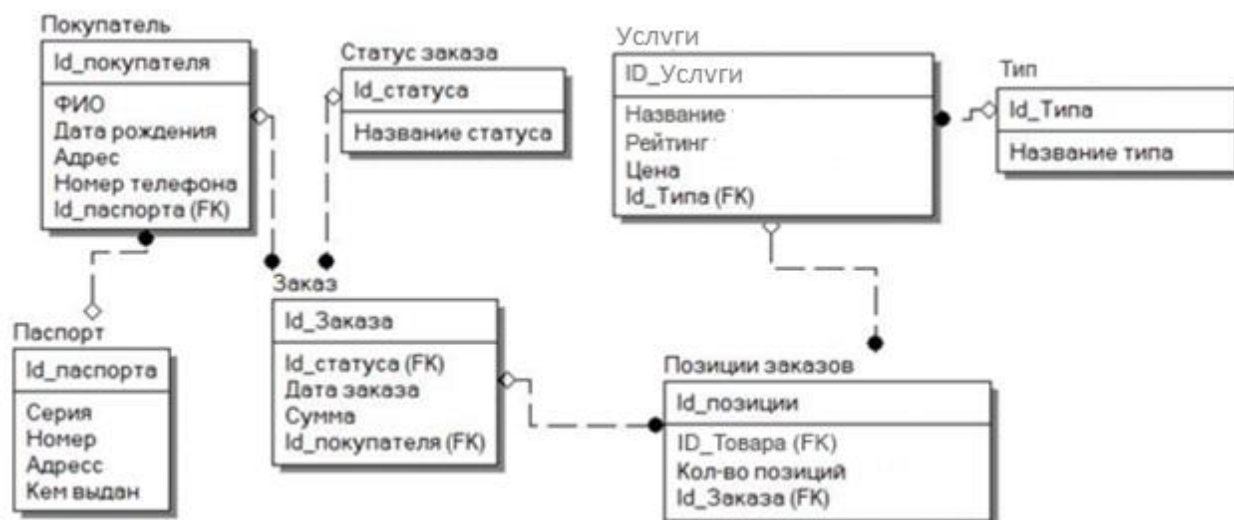


Рисунок 6 – Диаграмма инфологической модели

Физическая модель данных зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация о всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же инфологической модели могут соответствовать несколько разных физических моделей.

Физическая модель представлена на рисунке 7.

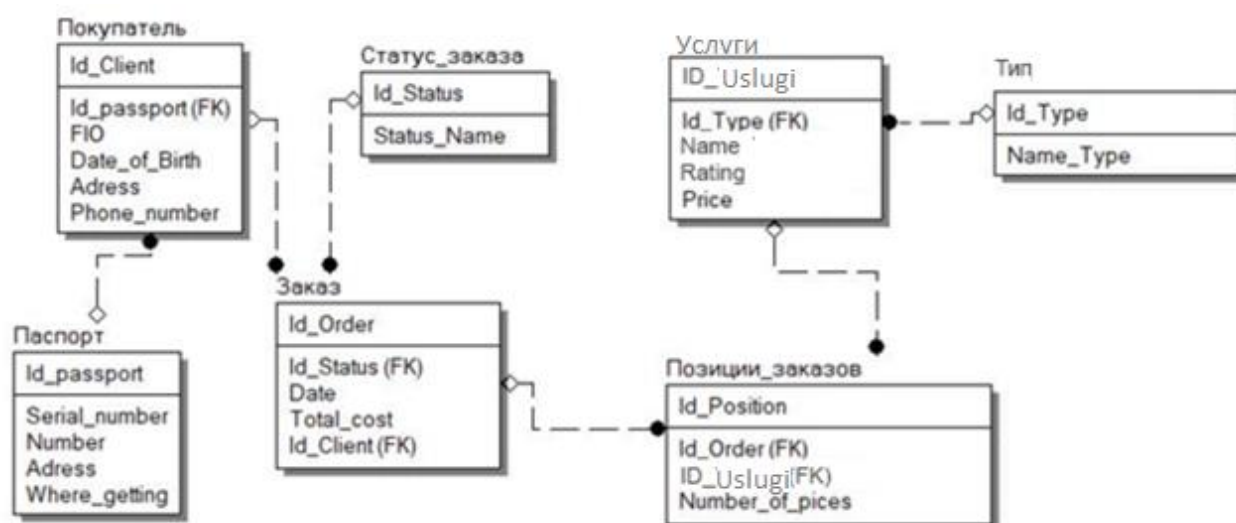


Рисунок 7 – Диаграмма физической модели

Если в инфологической модели не имеет значения, какой конкретно тип данных имеет атрибут, то в физической модели дело обстоит иначе – важно описать всю информацию о конкретных физических объектах, таких как таблицы, колонки, индексы, первичные и внешние ключи.

На диаграмме IDEF1X [7] физического уровня представлены таблицы и связи между ними, которые будут реализованы в разрабатываемой базе данных.

2.4 Проектирование поведения ИС

Для отражения взаимодействия информационной системы контроля работы салона красоты и ее компонентов построены поведенческие диаграммы: диаграмма состояния, диаграмма деятельности, диаграмма последовательности, диаграмма коммуникации, диаграмма компонентов и диаграмма развертывания.

2.4.1 Диаграмма состояний

Объекты характеризуются поведением и состоянием, в котором находятся. Диаграмма состояний показывает, как объект переходит из одного состояния в другое. В данном случае диаграмма состояний отражает состояние объекта класса «Список к заказу», и его возможные состояния.

Класс «Заказ», помимо начального и конечного состояния может, находиться в следующих состояниях:

- order is empty (заказ пуст) – состояние актуально в начале работы с заказом;
- order contains Uslugis (заказ содержит услуги) – в это состояние заказ переходит после добавления в нее услуги.

Переходы в указанные состояния на диаграмме имеют сторожевые условия (условия перехода).

2.4.2 Диаграмма деятельности

Диаграмма деятельности используется для графических изображений разбиения конкретного действия на составляющие элементы. В этом контексте деятельность представляет собой спецификацию поведения, выражена через совместное выполнение параллельных и последовательных элементов. В этих компонентах могут быть вложенные виды деятельности, а также отдельные мероприятия.

Взаимодействие элементов в диаграмме происходит через потоки, связывающие выходы одного узла с входами другого, определяя порядок исполнения.

Диаграмма деятельности построена для прецедента «Совершить заказ».

На диаграмме деятельности отражено выполнение действий:

Сначала необходимо просмотреть каталог услуг, затем добавить услуга в заказ. Если пользователь добавил все что хотел в заказ, пользователь переходит к оформлению заказа. Если нет, пользователь возвращается к выбору услуг. На диаграмме это отражено с помощью ветвления (ромб). Затем, пользователь вводит контактные данные в заявку. Если данные введены верно, оформление заявки завершается и деятельность заканчивается. Если ввел данные неверно – возвращается к этапу ввода данных.

Диаграмма представлена в приложении А.

2.4.3 Диаграмма последовательности

Диаграмма последовательности является средством для визуализации взаимодействий объектов системы в временных шкалах. В диаграмме представлен жизненный цикл определенного объекта, а также взаимодействие актеров системы в соответствии с определенным сценарием применения прецедента. Основным вниманием является порядок передачи и передачи сообщений между предметами.

Для разработанной информационной системы диаграммы последовательности демонстрируют сценарий «Добавление новых сотрудников». Диаграмма демонстрирует взаимодействие актера «Пользователь» с такими системными объектами, как «Форма заполнения», «Модуль управления данными» и «База данных», отправляет им запросы, получает ответ.

Объект «Клиент» выбирает интересующую его категорию услуги, затем «Система» возвращает услуги только запрошенной категории. Объект «Клиент» добавляет услуга в заказ. Объект «Система» Возвращает содержимое корзины на экран. «Пользователь» заполняет и отправляет заявку их «Системе», она, в свою очередь, отправляет заявку на почту администратора и возвращает объекту «Пользователь» сообщение, что заявка принята.

Диаграмма представлена в приложении А.

2.4.4 Диаграмма коммуникаций

Диаграмма коммуникаций является инструментом для визуализации взаимодействия элементов структурной структуры и ролей в сотрудничестве. По сравнению с диаграммой последовательности, акцентируемой на хронологическом аспекте, диаграмма связи сосредоточена на отношениях объектов. В диаграмме взаимодействия время не отдельное измерение, последовательность взаимодействия отображается в порядке номеров вызова.

Информационная система контроля закупки салона имеет диаграмму коммуникаций для иллюстрации процесса «администрирования каталога». В диаграмме представлены объекты, которые участвуют в процессе и их связи и порядок обмена сообщениями между ними.

На диаграмме коммуникаций представлены следующие объекты:

- администратор;
- система;
- админ панель;

- база данных.

Администратор переходит в админ панель, система возвращает ему форму авторизации, администратор вводит свои логин и пароль. После этого она попадает в админ-панель, где создает/изменяет/удаляет услугу, и отправляет запрос в БД. После успешного проведения операции БД возвращает измененную таблицу в админ панель.

Диаграмма представлена в приложении А.

2.4.5 Диаграмма компонентов

Диаграмма компонентов – статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты [8] и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты.

Автоматизированная информационная система контроля работы салона красоты содержит в себе следующие структурные компоненты:

- loginViewModel.cs, RaspIndexModel.cs, PagingInfo.cs, UslugisListViewModel.cs – модели;
- login.php, admin.Edit.php, rasp.checkUslugi.php, rasp.complUslugi.php, rasp.Index.php, rasp.unvUslugis.php – представления;
- accountController.cs, AdminController.cs, UslugiController.cs, NavController.cs, RaspController.cs – контролеры.

Компоненты связаны между собой отношениями зависимости.

Диаграмма представлена в приложении А.

2.4.6 Диаграмма развертывания

Диаграмма развертывания – это диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений-маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов компонентов программы, существующих только на этапе ее исполнения. Выделим основные компоненты web-приложения информационной системы контроля работы салона красоты. В компоненте SQL Server DB находится база данных нашей информационной системы. На компоненте ISS 11 web-сервер находится начальная страница web-приложения ИС <http://localhost/port>. Компонент браузер служит для подключения к стартовой странице приложения.

Компоненты связаны между собой отношениями зависимости и интерфейсами.

Диаграмма представлена в приложении А.

3 Разработка приложения

3.1 Программные средства

После ознакомления с основными существующими на данный момент типами баз данных, выбор пал на реляционную базу данных на основе SQL Server 2022. Microsoft SQL Server – система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основным используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями.

SQL (Structured Query Language) [9] – стандартный язык запросов по работе с реляционными базами данных. SQL в чистом (базовом) виде является информационно-логическим языком, а не языком программирования. Однако стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений, с учетом которых язык уже может рассматриваться в качестве языка программирования. SQL предоставляет пользователю достаточно простой и понятный механизм доступа к данным, не связанный с конструированием алгоритма и его описанием на языке программирования высокого уровня. Так, вместо указания того, как необходимо действовать, пользователь с помощью операторов SQL объясняет СУБД, что нужно сделать. Далее СУБД сама анализирует текст запроса и определяет, как именно его выполнять, и после этого выдает результат запроса.

Соединение с базой данных [10] и клиентского приложения осуществляется с помощью языковых средств C# [11,12], а именно его фреймворка ASP.NET в среде разработки Visual Studio 2022.

C# – это язык программирования, предназначенный для разработки самых разнообразных приложений, предназначенных для выполнения в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой

разработки приложений, но при этом сохраняет выразительность и элегантность, присущую языкам C. Так же весомым преимуществом языка C# является то, что он постоянно обновляется, благодаря чему пользователю не приходится прибегать к сторонним средствам для выполнения поставленных задач. Достаточно загрузить нужные библиотеки, добавляющие необходимый функционал.

Платформа ASP.NET MVC [13, 14] представляет технологию от компании Microsoft, предназначенную для создания различного рода Web-приложений: от небольших web-сайтов до крупных web-порталов и web-сервисов.

С одной стороны, ASP.NET MVC является продолжением развития платформы ASP.NET. Но с другой стороны, это не просто очередной релиз. Выход ASP.NET MVC фактически означает революцию всей платформы, ее качественное изменение.

С помощью ASP.NET MVC разработчик может создавать кроссплатформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь он уже не ограничен только этой операционной системой. То есть разработчик может запускать web-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания web-приложения можно использовать традиционный IIS, либо кросс-платформенный web-сервер

Microsoft Visual Studio – линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии как устаревшей технологии Windows Forms, так и более новой Windows Presentation Foundation. Так же эти продукты позволяют разрабатывать веб-сайты,

Web-приложения, web-службы как в родном, так и в управляемом кодах для всех платформ.

Преимущества ASP.NET MVC:

- архитектурный шаблон MVC;
- расширяемость;
- жесткий контроль над HTML и HTTP;
- тестируемость.

В результате, с использованием данных средств приложение было реализовано и запущено. Текст приложения приведен в приложении Б.

4 Отладка и тестирование

4.1 Тестирование программного обеспечения

Тестирование [15] – очень важный этап разработки как приложений. По способам, целям и глубине тестирования оно делится на следующие виды:

- функциональное тестирование (functional testing) [16];
- системное тестирование (system testing);
- тестирование производительности (performance testing);
- регрессионное тестирование (regression testing);
- модульное тестирование (unit testing);
- тестирование безопасности (security testing);
- тестирование локализации (localization testing);
- юзабилити тестирование (usability testing).

Модульное тестирование (Unit Testing) – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирования выполняется разработчиками на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект. Отсутствие модульного тестирования при написании кода значительно увеличивает уровень дефектов при дальнейшем (интеграционном, системном, и приемочном) тестировании. Качественное модульное тестирование на этапе разработки экономит время, а следовательно, в конечном итоге, и бюджет проекта.

Проведем модульное тестирование [17] работы отдельных функций приложения. Таблица с описанием тестов представлена ниже (таблица 4).

Результаты тестирования приведены в приложении В.

Таблица 4 – Модульное тестирование работы системы

№	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
1	2	3	4	5
1	Adding_Record_To_Rasp_Goes_To_Rasp_Screen	При добавлении записи в форму открывается модуль	Соответствует ожидаемым результатам	Тест пройден
2	Calculate_Rasp_Total	Подсчет итоговой загруженности дня	Соответствует ожидаемым результатам	Тест пройден
3	Can_Add_New_Lines	Добавление новых строк в базу данных из админ-панели работает верно	Соответствует ожидаемым результатам	Тест пройден
4	Can_Add_Quantity_For_Existing_Lines	Повторное добавление записи приводит к увеличению счетчика	Соответствует ожидаемым результатам	Тест пройден
5	Can_Add_To_Rasp	Добавление записи в базу данных работает верно	Соответствует ожидаемым результатам	Тест пройден
6	Can_Checkout_And_Submit_Order	Формирование и отправка запроса работает верно	Соответствует ожидаемым результатам	Тест пройден
7	Can_Clear_Contents	Функция очистки дня работает верно	Соответствует ожидаемым результатам	Тест пройден
8	Can_Create_Users	Добавление новых пользователей работает верно	Соответствует ожидаемым результатам	Тест пройден

Продолжение таблицы 4

1	2	3	4	5
9	Can_Delete_Valid_Records	Удаление записей из админ панели работает верно	Соответствует ожидаемым результатам	Тест пройден
10	Can_Edit_Record	Изменение записей из админ панели работает верно	Соответствует ожидаемым результатам	Тест пройден
11	Can_Generate_Page_Links	Генерация ссылок на страницы работает верно	Соответствует ожидаемым результатам	Тест пройден
12	Can_Login_With_Valid_Credits	Авторизация администратора для доступа к админ-панели проходит успешно	Соответствует ожидаемым результатам	Тест пройден
13	Can_Paginate	Разбиение на страницы работает верно	Соответствует ожидаемым результатам	Тест пройден
14	Can_Remove_Line	Удаление отдельных записей из дня работает верно	Соответствует ожидаемым результатам	Тест пройден
15	Can_Retrieve_Info_Data	Добавление информации пользователей работает верно	Соответствует ожидаемым результатам	Тест пройден
16	Can_Save_Valid_Changes	Изменения, соответствующие требованиям, успешно сохраняются	Соответствует ожидаемым результатам	Тест пройден

Окончание таблицы 4

1	2	3	4	5
17	Can_View_Rasp_Contents	Пользователь может просматривать расписание	Соответствует ожидаемым результатам	Тест пройден
18	Cannot_Checkout_Empty_Rasp	Невозможно отправить заявку при пустой записи	Соответствует ожидаемым результатам	Тест пройден
19	Cannot_Edit_Nonexist_Record	Невозможно редактировать несуществующую запись	Соответствует ожидаемым результатам	Тест пройден
20	Cannot_Login_Without_Valid	Невозможно войти по не верным данным аккаунта	Соответствует ожидаемым результатам	Тест пройден
21	Cannot_Retrive_Info_Data	Невозможно присвоить сведения несуществующей записи	Соответствует ожидаемым результатам	Тест пройден
22	Cannot_Save_Invalid_Changes	Невозможно сохранить изменения, если они не соответствуют требования	Соответствует ожидаемым результатам	Тест пройден
23	Generate_Day_Specific_Records	Количество записей подсчитывается верно	Соответствует ожидаемым результатам	Тест пройден
24	Index_Contains_All_Records	Расписание содержит все записи, содержащиеся в базе данных	Соответствует ожидаемым результатам	Тест пройден

Модульное тестирование [18] показало, что система функционирует корректно и не имеет явных ошибок и сбоев в работе, что является положительным результатом проведения тестирования.

4.1.1 Метрики программного средства

Метрика программного обеспечения – мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций.

Метрики используются в следующих целях:

- для контроля над процессом разработки в целом, а в тестировании в частности;
- для оценки прогресса выполнения работ по тестированию ПО;
- для последующего планирования процессов.

4.1.2 Расчет метрик кода с помощью встроенного средства Visual Studio

Среда разработки Microsoft Visual Studio 2022 имеет встроенное средство для выполнения автоматического расчета метрик кода. К таковым относятся:

- индекс удобства поддержки;
- сложность организации циклов;
- глубина наследования;
- объединение классов;
- строки кода.

Для расчета метрик кода средствами Microsoft Visual Studio 2022 необходимо выбрать пункт контекстного меню «Анализ», далее – «Рассчитать метрики кода».

На рисунке 8 приведен результат расчета метрик кода в среде Microsoft Visual Studio 2022.

Имя компонента	Индекс удобства поддержки	Сложность организации циклов	Глубина наследования	Взаимозависимость классов	Строки кода
nain (Debug)	84	71	2	32	108
Domain.Abstract	100	4	0	4	0
Domain.Concrete	72	21	2	20	54
Domain.Entities	89	46	1	16	54
ITests (Debug)	67	47	1	76	245
UnitTests	64	38	1	62	232
UnitTests.App_Start	81	9	1	17	13
bUI (Debug)	80	102	3	99	193
WebUI	80	4	2	16	11
WebUI.App_Start	80	9	1	19	14
WebUI.Controllers	71	47	3	43	104
WebUI.HtmlHelpers	59	3	1	7	12
WebUI.Infrastructure	80	5	1	19	11
WebUI.Infrastructure.	100	1	0	0	0
WebUI.Infrastructure.I	71	5	1	6	10
WebUI.Infrastructure.I	80	3	1	2	6
WebUI.Models	93	25	1	7	25

Рисунок 8 – Метрики кода в Visual Studio

Для наглядности среда Microsoft Visual Studio отображает цветовую индикацию уровня качества кода. Зеленая цвет свидетельствует о высоком качестве кода, желтая означает, что качество кода ниже оптимального. В данном случае видно, что качество кода всех компонентов имеет высокое качество.

4.1.3 Определение метрик компонента серверной части приложения

Далее рассмотрен исходный код метода Login класса AccountController.cs (рисунок 9), для которого будут рассчитаны метрики Холстеда и выполнен комплексный анализ кода. Анализ кода будет выполнен согласно формулам метрики Холстеда.

```
public ActionResult Login(LoginViewModel model, string returnUrl)
{
    if (ModelState.IsValid)
    {
        if (authProvider.Authenticate(model.UserName, model.Password))
        {
            return Redirect(returnUrl ?? Url.Action("Index", "Admin"));
        }
        else
        {
            ModelState.AddModelError("", "Неправильный логин или пароль");
            return View();
        }
    }
    else
    {
        return View();
    }
}
```

Рисунок 9 – Код метода Login(LoginViewModel, string)

Словарь для расчета метрик вышеуказанного блока исходного кода приведен в **таблице 2.**

Таблица 5 – Словарь

Переменная	№	Встречается	Оператор	№	Встречается
model	1	1	if	1	2
returnUrl	2	1	return	2	3
ModelState	3	2	else	3	2
authProvider	4	1	??	4	1
UserName	5	1			
Password	6	1			
Итого:	6	7		4	8
	n1	N1		n2	N2

Расчет метрик Холстеда выполнен с использованием математического пакета Mathcad. Результат расчета приведен ниже.

Исходные данные:

$$mn1 := 6, mN1 := 7, mn2 := 4, mN2 := 8.$$

Словарь, длина и объем программы:

$$n := mn1 + mn2 = 10;$$

$$N := mN1 + mN2 = 15;$$

$$V := N * \log(2, n) = 49,83.$$

Уровень качества программирования:

$$L := 2mn2 / (mn1 * mN2) = 0,17.$$

Сложность понимания программы.

$$EC := V/2L = 146,55.$$

Уровень языка:

$$Y := V/D2 = 1,38.$$

Оценка необходимых интеллектуальных усилий:

$$E := N * \log(2, n/L) = 87,87.$$

Трудоемкость кодирования:

$$D := 1/L = 6.$$

Информационное содержание программы:

$$I := V/D = 8,31.$$

Подход, приведенный в расчете метрик Холстеда, значительно усложняется при увеличении объема кода проекта. В таком случае целесообразно использовать встроенные в среду разработки или отдельно поставляемые инструменты для автоматического анализа качества кода.

Заключение

С помощью языка C#, базы данных SQL SERVER 2022 и среды разработки Visual Studio 2022 от компании Microsoft были реализованы база данных и web приложение информационной системы контроля работы салона красоты. По результатам функционального тестирования автоматизированная информационная система работает корректно и выполняет поставленные задачи.

Применение данной системы позволит организовать деятельность салонов красоты, где требуется онлайн запись к специалистам:

- быстро и удобно вести администрирование имеющихся услуг и специалистов;
- снизить случайные ошибки связанных с ручным оформлением записей к специалистам;
- автоматизировать обработку информации в системе;
- получить доступ к расписанию специалистов из любой точки, где есть компьютер, имеющий выход в Интернет;
- получить на выходе систему с низкой стоимостью разработки, но имеющую весь основной функционал, необходимый для комфортного и удобного работы с системой контроля работы салона красоты.

Список использованных источников

1. Информационная система – это //Что такое информационные системы и зачем они нужны – классификация и использование [Электронный ресурс] URL: <https://ktonanovenkogo.ru/voprosy-i-otvety/informacionnayasistema-chto-eh-to-takoe.html?ysclid=l4hevni6mn440545180> (дата обращения 07.05.2024).
2. Вендров А.М. Структурный подход к проектированию ИС // Портал научно-практических публикаций [Электронный ресурс]. URL: http://citforum.ru/database/case/glava2_1.shtml (дата обращения: 07.05.2024).
3. Приложение с открытым исходным кодом для персональных компьютеров // ArgoUML [Электронный ресурс]. URL: <https://argouml.softonic.ru> (Дата обращения: 15.05.2024).
4. Построение UML-модели в среде Visual Paradigm 16.1. Система обработки заказов [Электронный ресурс]. URL: <http://sp.cs.msu.ru/ooap/exerb2020.html> (Дата обращения: 18.05.2024).
5. Маклаков, С. В. BPwin и ERwin: CASE-средства для разработки информационных систем // Портал научно-практических публикаций [Электронный ресурс]. URL: <https://elib.pnzgu.ru/files/eb/doc/DSA5gziAMfNS.pdf> (дата обращения: 18.05.2024).
6. Проектирование информационных систем [Электронный ресурс]. URL: <https://moodle.kstu.ru/course/view.php?id=4638&> (Дата обращения: 26.05.2024).
7. Варианты использования // Диаграмма вариантов использования [Электронный ресурс]. URL: http://www.telenir.net/uchebniki/samouchitel_uml/p4.php?ysclid=l4hg07qpf139464158 (Дата обращения: 28.05.2024).
8. Варианты использования // Диаграмма вариантов использования [Электронный ресурс]. URL:

http://www.telenir.net/uchebniki/samouchitel_uml/p4.php?ysclid=l4hg07qpf139464158 (Дата обращения: 28.05.2024).

9. Что такое интерфейс пользователя? [Электронный ресурс]. URL: <https://www.internet-technologies.ru/articles/newbie/polzovatelskiy-interfeys.html> (Дата обращения: 01.06.2024).

10. Простое руководство по диаграмме компонентов: [Электронный ресурс] // So creately URL: <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/>. (Дата обращения: 04.06.2024).

11. Об одном подходе к разбиению на классы эквивалентности: [Электронный ресурс] // cyberleninka.ru. Теория. URL: <https://cyberleninka.ru/article/n/ob-odnom-podhode-k-razbieniyu-na-klassy-ekvivalentnosti> (дата обращения: 26.03.2024).

12. Граничные условия в неизолированных системах: [Электронный ресурс] // cyberleninka.ru. Теория. URL: <https://cyberleninka.ru/article/n/granichnye-usloviya-v-neizolirovannyh-sistemah>. (Дата обращения: 26.03.2024).

13. Простое сложное программирование: [Электронный ресурс] // <http://habr.com>. Теория. URL: <https://habr.com/ru/companies/timeweb/articles/551754/>. (Дата обращения: 26.03.2024).

14. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ: [Электронный ресурс] // cyberleninka.ru. Теория. URL: <https://cyberleninka.ru/article/n/primenenie-metrik-holsteda-dlya-kolichestvennogo-otsenivaniya-harakteristik-programm-evm>. (Дата обращения: 26.03.2024).

Приложение А
(рекомендуемое)
Диаграммы UML

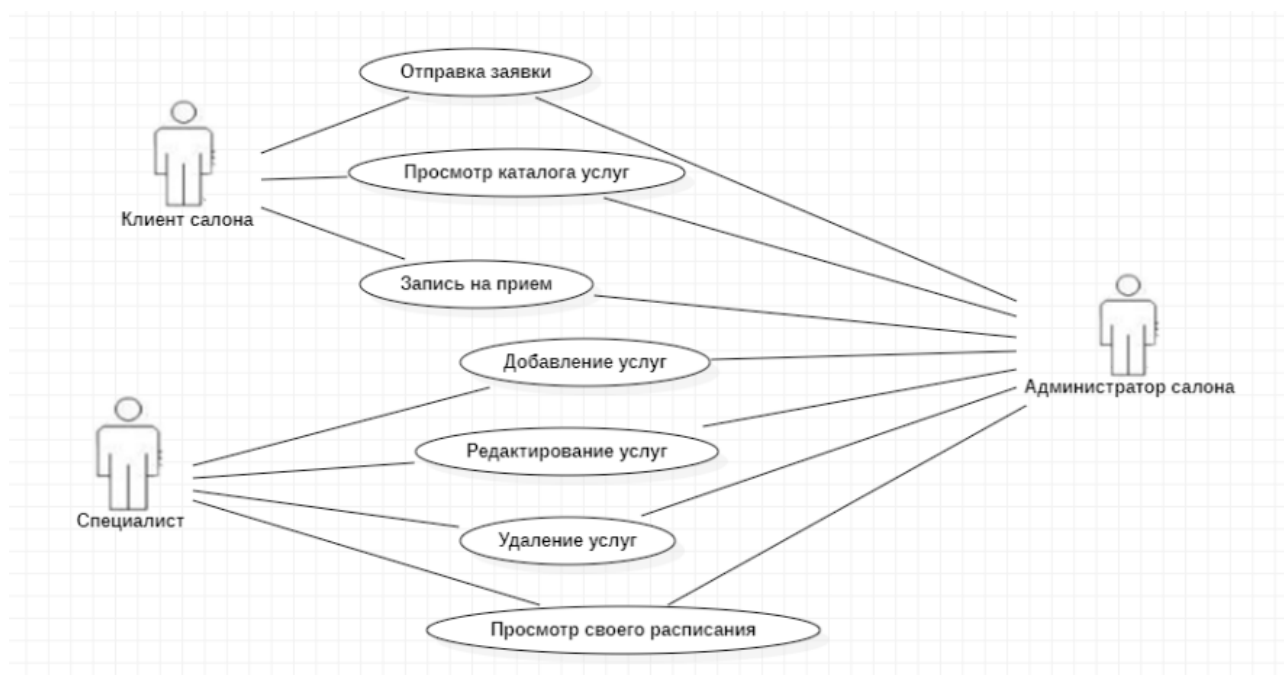


Рисунок А.1 – Диаграмма вариантов использования

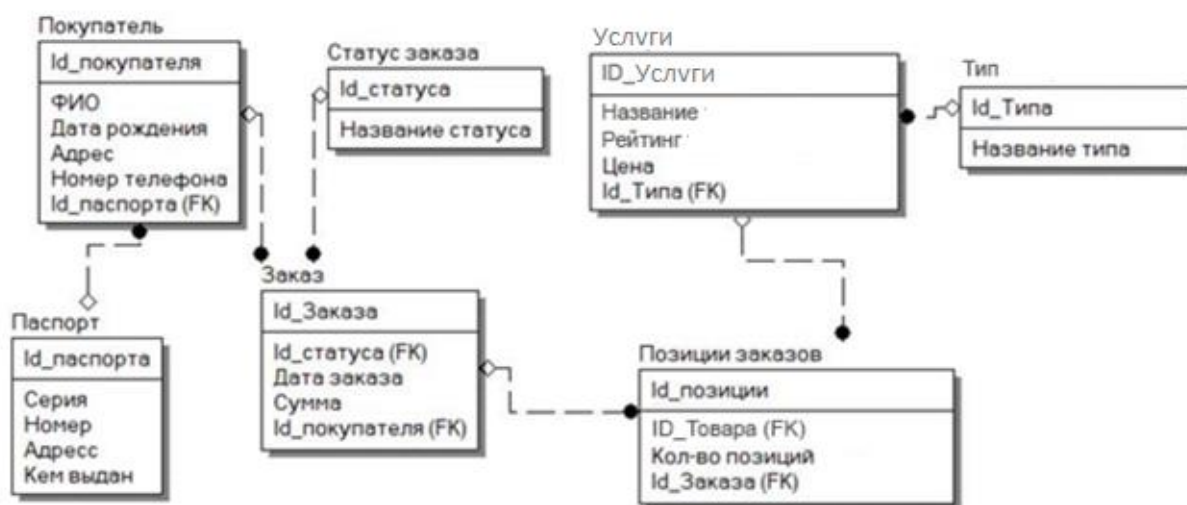


Рисунок А.2 – Диаграмма классов



Рисунок А.3 – Диаграмма коммуникаций

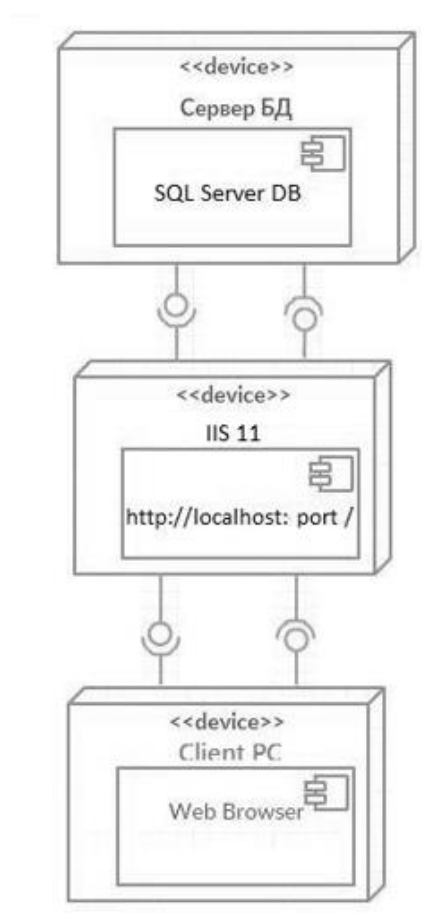


Рисунок А.4 – Диаграмма развертывания



Рисунок А.5 – Диаграмма состояний

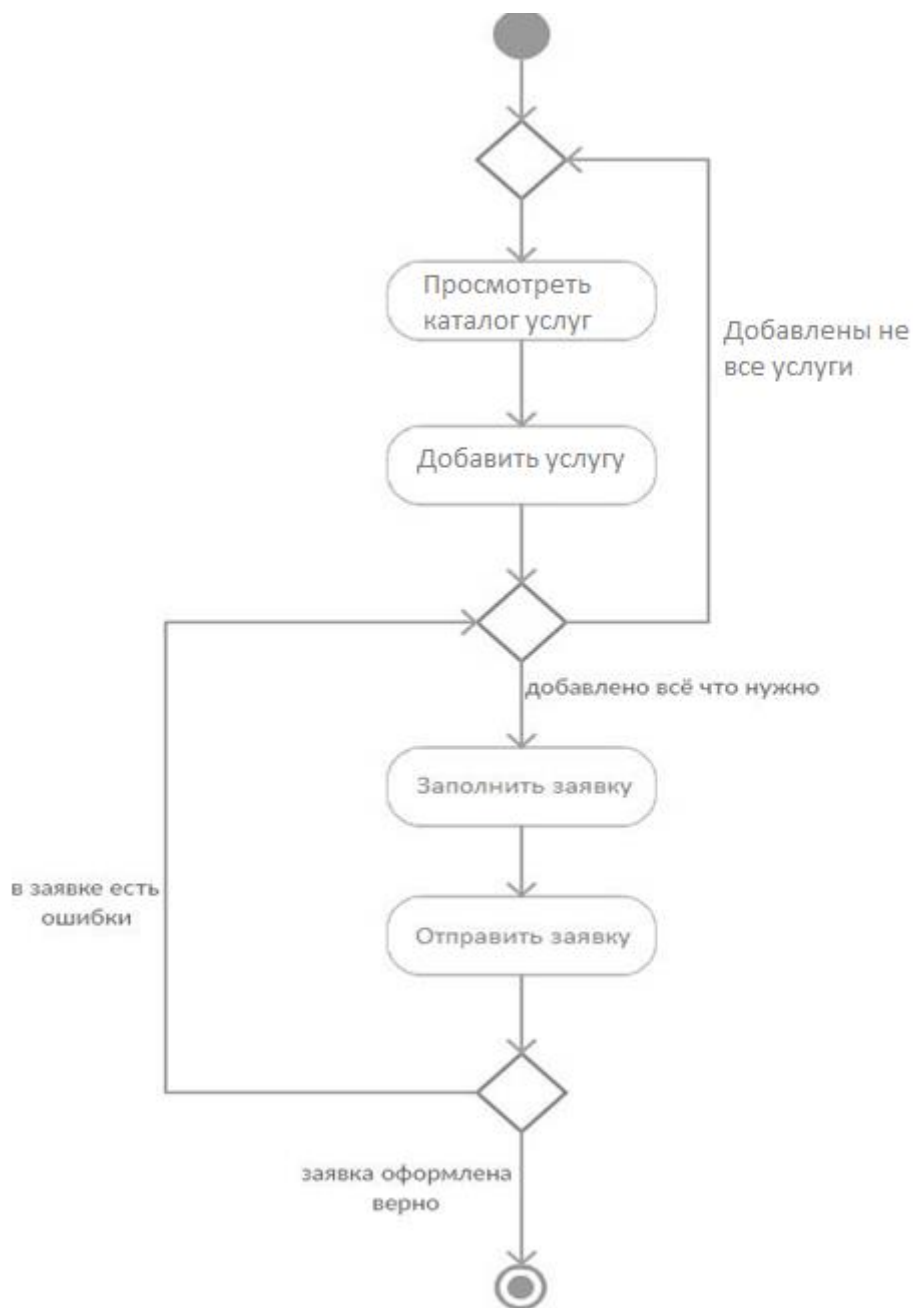


Рисунок А.6 – Диаграмма деятельности

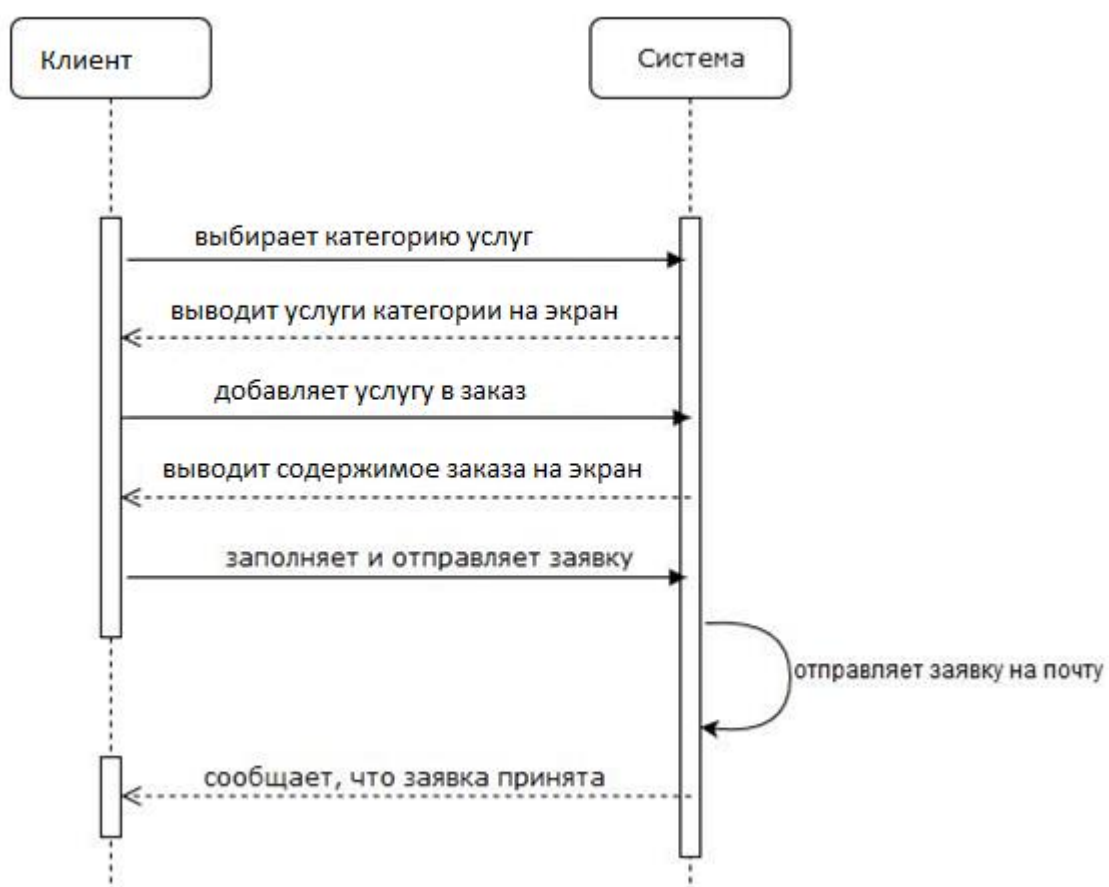


Рисунок А.7 – Диаграмма последовательности

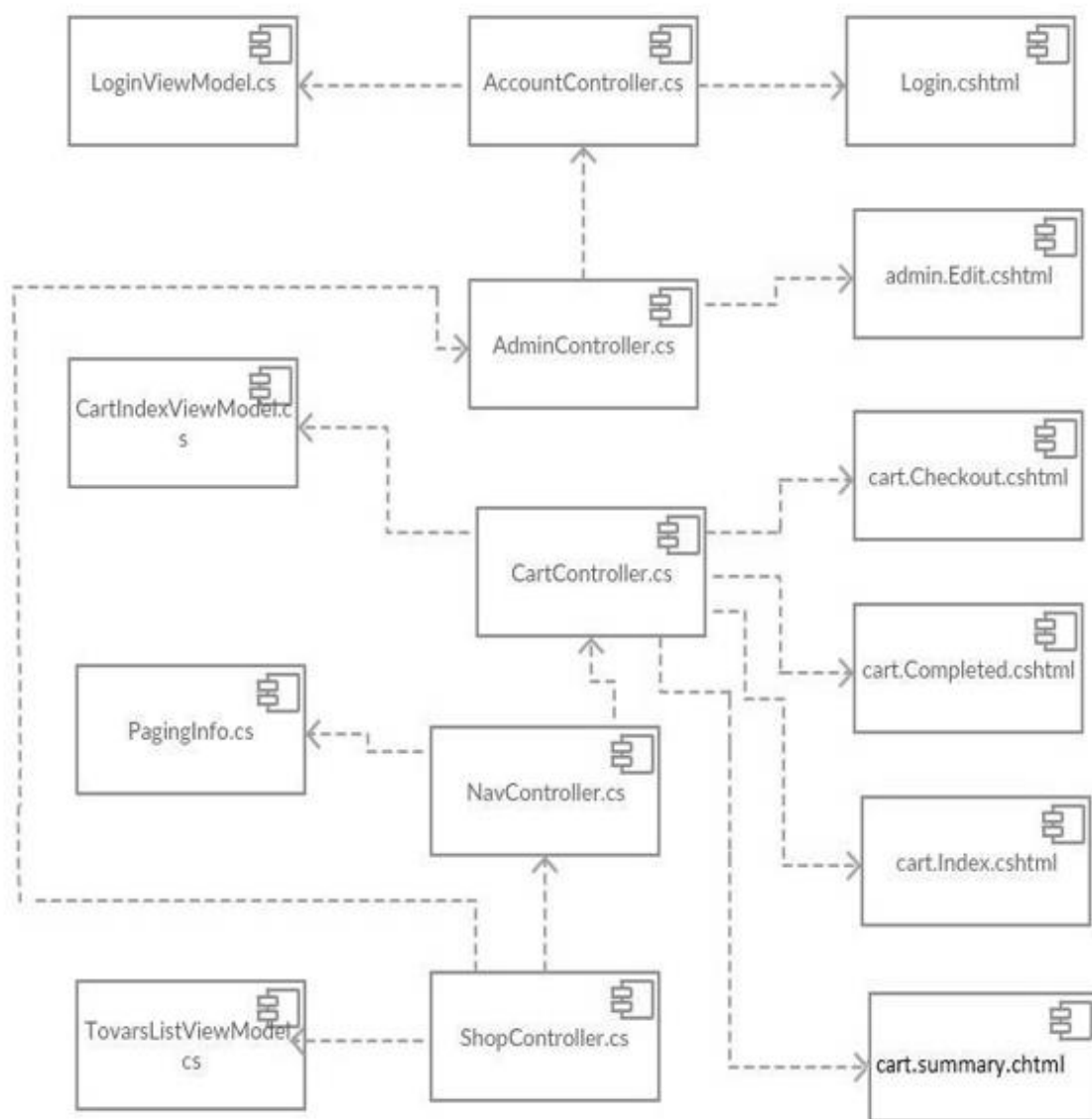


Рисунок А.8 – Диаграмма компонентов

Приложение Б
(рекомендуемое)
Листинг программы

Листинг Б.1 – Код модуля «Rasp.Domain.Concrete.EFDbContext.cs»

```

using System;
using System.Collections.Generic;
using System.Linq;

using System.Text;
using System.Threading.Tasks;
using Rasp.Domain.Entities;

namespace Rasp.Domain.Concrete
{

    public class EFDbContext : DbContext
    {
        public DbSet<Task> Tasks { get; set; }

    }

}

```

Листинг Б.2 – Код модуля «Rasp.Domain.Concrete EFRaspRepository.cs»

```

using System.Collections.Generic;

using Rasp.Domain.Entities;
using Rasp.Domain.Abstract;
using System.Linq;
using System.Text;

namespace Rasp.Domain.Concrete
{

    public class EFRaspRepository : IRaspRepository
    {
        EFDbContext ccontext = new EFDbContext();

        public IEnumerable<Task> Task
        {

            get { return ccontext.Tasks; }

        }

        public void SaveTask(Task task)

        {
            if (task.TaskId == 0)
                ccontext.Tasks.Add(task);

            else
            {
                Task dbEntry = ccontext.Tasks.Find(task.TaskId);

```

```

        if (dbEntry != null)
        {
            dbEntry.Name = task.Name; dbEntry.Description =
task.Description; dbEntry.Price = task.Price; dbEntry.Category =
task.Category; dbEntry.ImageData = task.ImageData;

            dbEntry.ImageMimeType = task.ImageMimeType;
        }
    }

    ccontext.SaveChanges();
}

public Task DeleteTask(int taskId)
{
    Task dbEntry = ccontext.Tasks.Find(taskId);
    if (dbEntry != null)
    {
        ccontext.Tasks.Remove(dbEntry);
        ccontext.SaveChanges();
    }

    return dbEntry;
}
}
}

```

Листинг Б.3 –Код модуля «Rasp.UnitTests. AdminSecurityTests.cs»

```

using System.Web.Mvc;

using Microsoft.VisualStudio.TestTools.UnitTesting; using Moq;
using Rasp.WebUI.Controllers;

using Rasp.WebUI.Infrastructure.Abstract; using
Rasp.WebUI.Models; using System.Linq;
using System.Text;

namespace Rasp.UnitTests
{
    [TestClass]
    public class AdminSecurityTests
    {
        [TestMethod]

        public void Can_Login_With_Valid_Credentials()

```

```

    {
//      Организация - создание имитации поставщика аутентификации

        Mock<IAuthProvider>newmock = new Mock<IAuthProvider>();
newmock.Setup(m => m.Authenticate("admin",
    "12345")).Returns(true);

//      Организация - создание модели представления
//      с правильными учетными данными
LoginViewModel model = new LoginViewModel
{

    UserName = "admin", Password = "12345" };

//      Организация - создание контроллера
AccountController target = new
AccountController(newmock.Object);

//      Действие - аутентификация
ActionResult result = target.Login(model, "/MyURL");

//      Утверждение

    Assert.IsInstanceOfType(result, typeof(RedirectResult));
Assert.AreEqual("/MyURL", ((RedirectResult)result).Url); }

[TestMethod]
public void Cannot_Login_With_Invalid_Credentials()

    {
//      Организация - создание имитации поставщика аутентификации

        Mock<IAuthProvider>newmock = new Mock<IAuthProvider>();
newmock.Setup(m => m.Authenticate("badUser",
    "badPass")).Returns(false);

//      Организация - создание модели представления

//      с некорректными учетными данными
LoginViewModel model = new LoginViewModel
{

    UserName = "badUser", Password = "badPass" };

//      Организация - создание контроллера
AccountController target = new
AccountController(newmock.Object);

//      Действие - аутентификация
ActionResult result = target.Login(model, "/MyURL");

//      Утверждение

```

```

        Assert.IsInstanceOfType(result, typeof(ViewResult));
Assert.IsFalse(((ViewResult)result).ViewData.ModelState.IsValid);

    }
}
}

```

Листинг Б.4 – Код модуля Rasp.UnitTests. AdminTests.cs

```

using Rasp.WebUI.Controllers;

using Microsoft.VisualStudio.TestTools.UnitTesting; using Moq;

using Rasp.Domain.Abstract;
using Rasp.Domain.Entities;
using System;

using System.Collections.Generic;

using System.Linq;
using System.Web.Mvc;
using System.Text;

namespace Rasp.UnitTests
{
    [TestClass]

    public class AdminTests
    {
        [TestMethod]

        public void Index_Contains_All_Taskss()
        {
            // Организация - создание имитированного хранилища данных

            Mock<IRaspRepository> newmock = new Mock<IRaspRepository>();
            newmock.Setup(m => m.Task).Returns(new List<Task>
            {

                new Task { TaskId = 1, Name = "Запись1"}, new Task { TaskId =
                2, Name = "Запись2"}, new Task { TaskId = 3, Name = "Запись3"},
                new Task { TaskId = 4, Name = "Запись4"}, new Task { TaskId = 5,
                Name = "Запись5"} });

            // Организация - создание контроллера

            AdminController controller = new
            AdminController(newmock.Object);

            // Действие

```

```

        List<Task> result =
            ((IEnumerable<Task>)controller.Index().
            ViewData.Model).ToList();

//          Утверждение

        Assert.AreEqual(result.Count(), 5); Assert.AreEqual("Запись1",
result[0].Name); Assert.AreEqual("Запись2", result[1].Name);
Assert.AreEqual("Запись3", result[2].Name);
    }
    [TestMethod]

    public void Can_Edit_Task()
    {
//          Организация - созданиеимитированногохранилищаданных

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
newmock.Setup(m => m.Task).Returns(new List<Task>
    {

        new Task { TaskId = 1, Name = "Запись1"}, new Task { TaskId =
2, Name = "Запись2"}, new Task { TaskId = 3, Name = "Запись3"},
new Task { TaskId = 4, Name = "Запись4"}, new Task { TaskId = 5,
Name = "Запись5"} });

//          Организация - созданиеконтроллера
        AdminController controller = new
        AdminController(newmock.Object);

//          Действие

        Task task1 = controller.Edit(1).ViewData.Model as Task; Task
task2 = controller.Edit(2).ViewData.Model as Task; Task task3 =
controller.Edit(3).ViewData.Model as Task;

        // Assert

        Assert.AreEqual(1, task1.TaskId);
        Assert.AreEqual(2, task2.TaskId);
        Assert.AreEqual(3, task3.TaskId);
    }

    [TestMethod]
    public void Cannot_Edit_Nonexistent_Task()
    {
//          Организация - созданиеимитированногохранилищаданных

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
newmock.Setup(m => m.Task).Returns(new List<Task>
    {

```

```

        new Task { TaskId = 1, Name = "Запись1"}, new Task { TaskId =
2, Name = "Запись2"}, new Task { TaskId = 3, Name = "Запись3"},
new Task { TaskId = 4, Name = "Запись4"}, new Task { TaskId = 5,
Name = "Запись5"} });

//          Организация - создание контроллера

        AdminController controller = new
AdminController(newmock.Object);

//          Действие
        Task result = controller.Edit(6).ViewData.Model as Task;

//          Assert
    }

    [TestMethod]
    public void Can_Save_Valid_Changes()
    {

//          Организация - создание имитированного хранилища данных
        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();

//          Организация - создание контроллера

        AdminController controller = new
AdminController(newmock.Object);

//          Организация - создание объекта Task Task task = new Task {
Name = "Test" };

//          Действие - попытка сохранения Записи
        ActionResult result = controller.Edit(task);

//          Утверждение - проверка того, что к хранилищу производится
обращение
        newmock.Verify(m => m.SaveTask(task));

//          Утверждение - проверка типа результата метода
        Assert.IsNotInstanceOfType(result, typeof(ViewResult));
    }

    [TestMethod]
    public void Cannot_Save_Invalid_Changes()
    {

//          Организация - создание имитированного хранилища данных
        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();

//          Организация - создание контроллера

        AdminController controller = new
AdminController(newmock.Object);

```

```

//      Организация - создание объекта Task Task task = new Task {
Name = "Test" };

//      Организация - добавление ошибки в состояние модели
controller.ModelState.AddModelError("error", "error");

//      Действие - попытка сохранения Записи

        ActionResult result = controller.Edit(task);

//      Утверждение - проверка того, что обращение к хранилищу НЕ
производится

        newmock.Verify(m => m.SaveTask(It.IsAny<Task>()),
Times.Never());

//      Утверждение - проверка типа результата метода
Assert.IsInstanceOfType(result, typeof(ViewResult));
    }

    [TestMethod]
    public void Can_Delete_Valid_Tasks()
    {

//      Организация - создание объекта Task
Task task = new Task { TaskId = 2, Name = "Запись2" };

//      Организация - создание имитированного хранилища данных

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
newmock.Setup(m => m.Task).Returns(new List<Task>
    {

        new Task { TaskId = 1, Name = "Запись1"}, new Task { TaskId = 2,
Name = "Запись2"}, new Task { TaskId = 3, Name = "Запись3"},

        new Task { TaskId = 4, Name = "Запись4"}, new Task { TaskId =
5, Name = "Запись5" } });

//      Организация - создание контроллера
AdminController controller = new
AdminController(newmock.Object);

//      Действие - удаление Записи controller.Delete(task.TaskId);

//      Утверждение - проверка того, что метод удаления в
хранилище

//      вызывается для корректного объекта Task newmock.Verify(m
=>m.DeleteTask(task.TaskId));
    }
}
}

```

Листинг Б.5 –Код модуля «Rasp.UnitTests.RaspTests.cs»

```

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;      using
System.Linq;
using System.Collections.Generic;
using Rasp.Domain.Entities;
using Rasp.Domain.Abstract;

using Moq;
using Rasp.WebUI.Controllers;
using System.Web.Mvc;

using Rasp.WebUI.Models;

namespace Rasp.UnitTests
{
    [TestClass]
    public class RaspTests
    {
        [TestMethod]
        public void Can_Add_New_Lines()
        {
            //      Организация - создание нескольких тестовых Записей

            Task task1 = new Task { TaskId = 1, Name = "Запись1" }; Task task2
            = new Task { TaskId = 2, Name = "Запись2" };

            //      Организация - создание списка

            Rasp rasp = new Rasp();

            //      Действие rasp.AddItem(task1, 1);

            rasp.AddItem(task2, 1);
            List<RaspLine> results = rasp.Lines.ToList();

            //      Утверждение

            Assert.AreEqual(results.Count(),                2);
            Assert.AreEqual(results[0].Task,                 task1);
            Assert.AreEqual(results[1].Task, task2);
        }

        [TestMethod]

        public void Can_Add_Quantity_For_Existing_Lines()
        {

```

```

//          Организация - создание нескольких тестовых Записей

    Tasktask1 = new Task { TaskId = 1, Name = "Запись1" }; Task task2
= new Task { TaskId = 2, Name = "Запись2" };

//          Организация - создание списка

    Rasp rasp = new Rasp();

//          Действие  rasp.AddItem(task1, 1);  rasp.AddItem(task2, 1);
    rasp.AddItem(task1, 5);
    List<RaspLine> results = rasp.Lines.OrderBy(c =>

        c.Task.TaskId).ToList();

// Утверждение
    Assert.AreEqual(results.Count(), 2);

    Assert.AreEqual(results[0].Quantity, 6);
    экземпляров добавлено в корзину

    Assert.AreEqual(results[1].Quantity, 1);

}
// 6

[TestMethod]
public void Can_Remove_Line()

{
//          Организация - создание нескольких тестовых Записей

    Tasktask1 = new Task { TaskId = 1, Name = "Запись1" }; Task task2
= new Task { TaskId = 2, Name = "Запись2" }; Task task3 = new Task {
TaskId = 3, Name = "Запись3" };

//          Организация - создание списка

    Rasp rasp = new Rasp();

//          Организация - добавление нескольких Записей в корзину
    rasp.AddItem(task1, 1);

        rasp.AddItem(task2, 4);        rasp.AddItem(task3, 2);
    rasp.AddItem(task2, 1);

//          Действие
    rasp.RemoveLine(task2);

//          Утверждение
    Assert.AreEqual(rasp.Lines.Where(c => c.Task ==

        task2).Count(), 0);

```

```

Assert.AreEqual(rasp.Lines.Count(), 2);
}

[TestMethod]
public void Calculate_Rasp_Total()
{
//      Организация - создание нескольких тестовых Записей

    Tasktask1 = newTask { TaskId = 1, Name = "Запись1", Price
    100 };

    Task task2 = new Task { TaskId = 2, Name = "Запись2", Price =
55 };

//      Организация - создание списка

    Rasp rasp = new Rasp();

//      Действие
    rasp.AddItem(task1, 1);

    rasp.AddItem(task2, 1);
    rasp.AddItem(task1, 5);
    decimal result = rasp.ComputeTotalValue();

//      Утверждение
    Assert.AreEqual(result, 655);
}

[TestMethod]
public void Can_Clear_Contents()
{
//      Организация - создание нескольких тестовых Записей

    Tasktask1 = newTask { TaskId = 1, Name = "Запись1", Price
    100 };

    Task task2 = new Task { TaskId = 2, Name = "Запись2", Price =
55 };

//      Организация - создание списка

    Rasp rasp = new Rasp();

//      Действие
    rasp.AddItem(task1, 1);
    rasp.AddItem(task2, 1);
    rasp.AddItem(task1, 5);
    rasp.Clear();

//      Утверждение

```

```

        Assert.AreEqual(rasp.Lines.Count(), 0);

    }

    [TestMethod]

    public void Can_Add_To_Rasp()
    {
        //      Организация - созданиеимитированногохранилища

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
        newmock.Setup(m => m.Task).Returns(new List<Task> {

            new Task {TaskId = 1, Name = "Записы1", Category = "Кат1"},
        }.AsQueryable());

        //      Организация - созданиесписка
        Rasprasp = newRasp();

        //      Организация - созданиеконтроллера
        RaspController controller = new
            RaspController(newmock.Object, null);

        //      Действие - добавить Запись в корзину
        controller.AddToRasp(rasp, 1, null);

        //      Утверждение

        Assert.AreEqual(rasp.Lines.Count(), 1);
        Assert.AreEqual(rasp.Lines.ToList()[0].Task.TaskId, 1);

    }

    ///      <summary>

    ///      После добавления Запися в корзину, должно быть
    ///      перенаправление на страницу списка
    ///      </summary>
    [TestMethod]

    public void Adding_Task_To_Rasp_Goes_To_Rasp_Screen()
    {
        //      Организация - созданиеимитированногохранилища

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
        newmock.Setup(m => m.Task).Returns(new List<Task> {

            new Task {TaskId = 1, Name = "Записы1", Category = "Кат1"},
        }.AsQueryable());

        //      Организация - созданиесписка
        Rasprasp = newRasp();

        //      Организация - созданиеконтроллера

```

```

        RaspController controller = new
        RaspController(newmock.Object, null);

//        Действие - добавитьЗаписьвкорзину

        RedirectToRouteResult result = controller.AddToRasp(rasp, 2,
        "myUrl");

//        Утверждение

        Assert.AreEqual(result.RouteValues["action"], "Index");
        Assert.AreEqual(result.RouteValues["returnUrl"], "myUrl");
    }

//        Проверяем URL [TestMethod]
    public void Can_View_Rasp_Contents()

    {
//        Организация - создание списка
        Rasp rasp = new Rasp();

//        Организация - созданиеконтроллера
        RaspController target = new RaspController(null, null);

//        Действие - вызов метода действия Index()
        RaspIndexViewModel result
        = (RaspIndexViewModel)target.Index(rasp,
        "myUrl").ViewData.Model;

//        Утверждение

        Assert.AreSame(result.Rasp, rasp);
        Assert.AreEqual(result.ReturnUrl, "myUrl");
    }

    [TestMethod]

    public void Cannot_Checkout_Empty_Rasp()
    {
//        Организация - созданиеимитированногообработчиказаказов

        Mock<IOrderProcessor>newmock = new Mock<IOrderProcessor>();

//        Организация - создание пустой списка
        Rasp rasp = new Rasp();

//        Организация - созданиедеталейодоставке
        ShippingDetailsshippingDetails = newShippingDetails();

//        Организация - созданиеконтроллера

        RaspController controller = new RaspController(null,
        newmock.Object);

```

```

//          Действие

        ViewResult      result      =      controller.Checkout(rasp,
shippingDetails);

//          Утверждение - проверка, что заказ не был передан
обработчику

        newmock.Verify(m      =>      m.ProcessOrder(It.IsAny<Rasp>()),
It.IsAny<ShippingDetails>()),
        Times.Never());

//          Утверждение - проверка, что метод вернул стандартное
представление
        Assert.AreEqual("", result.ViewName);

//          Утверждение - проверка, что-представлению передана
неверная модель

        Assert.AreEqual(false, result.ViewData.ModelState.IsValid); }

[TestMethod]
public void Can_Checkout_And_Submit_Order()
{
//          Организация - созданиеимитированногообработчиказаказов

        Mock<IOrderProcessor>newmock = new Mock<IOrderProcessor>();

//          Организация - создание списка с элементом

        Rasp rasp = new Rasp();
        rasp.AddItem(new Task(), 1);

//          Организация - создание контроллера

        RaspController controller = new RaspController(null,
newmock.Object);

//          Действие - попыткапереходакоплате

        ViewResult      result      =      controller.Checkout(rasp,      new
ShippingDetails());

//          Утверждение - проверка, что заказ передан обработчику
newmock.Verify(m => m.ProcessOrder(It.IsAny<Rasp>()),
        It.IsAny<ShippingDetails>()),

        Times.Once());

//          Утверждение - проверка, что метод возвращает
представление

        Assert.AreEqual("Completed", result.ViewName);

```

```
//      Утверждение - проверка, что представлению передается
допустимая модель
    Assert.AreEqual(true, result.ViewData.ModelState.IsValid);
}
}

}
```

Листинг Б.6 –Код модуля «Rasp.UnitTests.ImageTests.cs»

```
using Microsoft.VisualStudio.TestTools.UnitTesting; using Moq;

using Rasp.Domain.Abstract;
using Rasp.Domain.Entities;
using Rasp.WebUI.Controllers;

using System.Linq;
using System.Web.Mvc;
using System.Collections.Generic;

namespace Rasp.UnitTests
{
    [TestClass]

    public class ImageTests

    {
        [TestMethod]
        public void Can_Retrieve_Image_Data()

        {

//      Организация - создание объекта Task с данными изображения
        Task task = new Task

        {

            TaskId = 2, Name = "Запись2",

            ImageData = new byte[] { }, ImageMimeType = "image/png" };

//      Организация - создание имитированного хранилища

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
        newmock.Setup(m => m.Task).Returns(new List<Task> {

            new Task {TaskId = 1, Name = "Запись1"}, task,

            new Task {TaskId = 3, Name = "Запись3"} }.AsQueryable());
```

```

//          Организация - создание контроллера

        RaspController          controller          =          new
RaspController(newmock.Object);

//          Действие - вызов метода действия GetImage()
        ActionResult result = controller.GetImage(2);

//          Утверждение
        Assert.IsNotNull(result);

        Assert.IsInstanceOfType(result, typeof(FileResult));
        Assert.AreEqual(task.ImageMimeType,
            ((FileResult)result).ContentType);
    }

    [TestMethod]
    public void Cannot_Retrieve_Image_Data_For_Invalid_ID()
    {

//          Организация - создание имитированного хранилища

        Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
        newmock.Setup(m => m.Task).Returns(new List<Task> {

            new Task {TaskId = 1, Name = "Запись1"}, new Task {TaskId =
2, Name = "Запись2"} }.AsQueryable());

//          Организация - создание контроллера

        RaspController          controller          =          new
RaspController(newmock.Object);

//          Действие - вызов метода действия GetImage()
        ActionResult result = controller.GetImage(10);

//          Утверждение
        Assert.IsNull(result);

    }
}
}

```

Листинг Б.7 – Код модуля «Rasp.UnitTests.UnitTest1.cs»

```

using System;
using System.Collections.Generic;
using System.Linq;

using System.Web.Mvc;

```

```

using Microsoft.VisualStudio.TestTools.UnitTesting; using Moq;
using Rasp.Domain.Abstract;

using Rasp.Domain.Entities;
using Rasp.WebUI.Controllers;
using Rasp.WebUI.Models;

using Rasp.WebUI.HtmlHelpers;

namespace Rasp.UnitTests
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void Can_Paginate()
        {
            //      Организация (arrange)

            Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
            newmock.Setup(m => m.Task).Returns(new List<Task> {

                new Task { TaskId = 1, Name = "Запись1"}, new Task { TaskId =
2, Name = "Запись2"}, new Task { TaskId = 3, Name = "Запись3"},
new Task { TaskId = 4, Name = "Запись4"}, new Task { TaskId = 5,
Name = "Запись5"} });

            RaspController controller = new
RaspController(newmock.Object);

            controller.pageSize = 3;

            //      Действие (act) TasksListViewModel result =
            (TasksListViewModel)controller.List(null, 2).Model;

            //      Утверждение

            List<Task> tasks = result.Tasks.ToList();
            Assert.IsTrue(tasks.Count == 2); Assert.AreEqual(tasks[0].Name,
"Запись4"); Assert.AreEqual(tasks[1].Name, "Запись5");

        }
        [TestMethod]
        public void Can_Generate_Page_Links()
        {

            //      Организация - определение вспомогательного метода HTML -
            это необходимо

            //      для применения расширяющего метода
            HtmlHelper myHelper = null;

```

```

//      Организация - создание объекта PagingInfo
PagingInfo pagingInfo = new PagingInfo
{

    CurrentPage = 2, TotalItems = 28, ItemsPerPage = 10 };

//      Организация - настройка делегата с помощью лямбда-
выражения
Func<int, string> pageUrlDelegate = i => "Page" + i;

//      Действие

MvphpString      result      =      myHelper.PageLinks(pagingInfo,
pageUrlDelegate);

//      Утверждение
Assert.AreEqual(@"<a class=""btn btn-default""

href=""Page1"">1</a>"

    @"<a      class=""btn      btn-default      btn-primary      selected""
href=""Page2"">2</a>"

    @"<a      class=""btn      btn-default""      href=""Page3"">3</a>",
result.ToString());
}
[TestMethod]

public void Can_Send_Pagination_View_Model()
{
    // Организация (arrange)

    Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
    newmock.Setup(m => m.Task).Returns(new List<Task>
    {

        new Task { TaskId = 1, Name = "Запись1"}, new Task { TaskId =
2, Name = "Запись2"}, new Task { TaskId = 3, Name = "Запись3"},
new Task { TaskId = 4, Name = "Запись4"}, new Task { TaskId = 5,
Name = "Запись5"} });
    RaspController controller = new
    RaspController(newmock.Object);

    controller.pageSize = 3;

    // Act

    TasksListViewModel result
    = (TasksListViewModel)controller.List(null, 2).Model;

    // Assert

```

```

        PagingInfo pageInfo = result.PagingInfo;
Assert.AreEqual(pageInfo.CurrentPage, 2);
Assert.AreEqual(pageInfo.ItemsPerPage, 3);
Assert.AreEqual(pageInfo.TotalItems, 5);
Assert.AreEqual(pageInfo.TotalPages, 2); }

[TestMethod]
public void Can_Create_Categories()
{
//      Организация - созданиеимитированногохранилища

    Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
newmock.Setup(m => m.Task).Returns(new List<Task> {
    new Task { TaskId = 1, Name = "Запись1",
    Category="Мат.платы"},

    new Task { TaskId = 2, Name = "Запись2", Category="Задачи"},

    new Task { TaskId = 3, Name = "Запись3", Category=" Запись
"}, new Task { TaskId = 4, Name = "Запись4", Category=" Запись "},
});

//      Организация - созданиеконтроллера
NavController target = new NavController(newmock.Object);

//      Действие - получениинаборакатегорий
List<string> results =

    ((IEnumerable<string>)target.Menu().Model).ToList();

//      Утверждение

Assert.AreEqual(results.Count(), 3); Assert.AreEqual(results[1],
"Задачи"); Assert.AreEqual(results[0], " Задачи ");
Assert.AreEqual(results[2], " Задачи ");
}

[TestMethod]
public void Indicates_Selected_Category()
{
//      Организация - созданиеимитированногохранилища

    Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
newmock.Setup(m => m.Task).Returns(new Task[] {

    new Task { TaskId = 1, Name = "Запись1",
    Category=" Задачи "},

    new Task { TaskId = 2, Name = "Запись2", Category=" Задачи "}
});

```

```

// Организация - создание контроллера

NavController target = new NavController(newmock.Object);

// Организация - определение выбранной категории string
categoryToSelect = "Задачи";

// Действие
string result =

target.Menu(categoryToSelect).ViewBag.SelectedCategory;

// Утверждение

Assert.AreEqual(categoryToSelect, result);
}

[TestMethod]

public void Generate_Category_Specific_Task_Count()
{
/// Организация (arrange)

Mock<IRaspRepository>newmock = new Mock<IRaspRepository>();
newmock.Setup(m => m.Task).Returns(new List<Task> {

    new Task { TaskId = 1, Name = "Запись1", Category="Cat2"},
    new Task { TaskId = 2, Name = "Запись2", Category="Cat3"}, new
Task { TaskId = 3, Name = "Запись3", Category="Cat2"}, new Task {
TaskId = 4, Name = "Запись4", Category="Cat1"}, new Task { TaskId
= 5, Name = "Запись5", Category="Cat1"} });

RaspController controller = new RaspController(mock.Object);
controller.pageSize = 3;

// Действие - тестирование счетчиков Записей для различных
категорий

int res1 =
((TasksListViewModel)controller.List("Cat1").Model).PagingInfo.
TotalItems;

int res2 =
((TasksListViewModel)controller.List("Cat2").Model).PagingInfo.
TotalItems;

int res3 =
((TasksListViewModel)controller.List("Cat3").Model).PagingInfo.
TotalItems;

int resAll =
((TasksListViewModel)controller.List(null).Model).PagingInfo.To
talItems;

```

```
//      Утверждение
Assert.AreEqual(res1, 2);

Assert.AreEqual(res2, 2);
Assert.AreEqual(res3, 1);
Assert.AreEqual(resAll, 5);

}
```

Приложение В
(рекомендуемое)
Результаты тестирования

