

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 АНАЛИЗ ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ	8
1.1 Роль электронных систем очередей	8
1.2 Оборудование и программное обеспечение электронных очередей	10
1.3 Обзор аналогов существующих систем	14
2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	19
2.1 Алгоритм работы системы управления электронными очередями	19
2.2 Разработка модели вариантов использования	22
2.3 Разработка информационной модели предметной области	24
3 РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ	27
3.1 Описание использованных инструментов	27
3.2 Описание структуры программного модуля	35
3.3 Описание модуля Client	36
3.4 Описание модуля Data	37
3.5 Описание модуля Services	38
3.6 Описание модуля Infrastructure	39
4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ	40
4.1 Модульное тестирование	40
4.2 Функциональное тестирование	41
5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	42
6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ	45
6.1 Характеристика программного модуля и исходные данные	45
6.2 Расчет затрат и отпускной цены программного средства	46
6.3 Расчет сметы затрат и цены заказного ПО	48
6.4 Оценка экономической эффективности применения программного средства у пользователя	52
ЗАКЛЮЧЕНИЕ	57
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	59
ПРИЛОЖЕНИЕ А	60

ВВЕДЕНИЕ

Каждая управленческая функция сопровождается определенным комплексом документов, состав которых зависит от круга решаемых задач, объема и характера компетенции предприятия, порядка принятия решения, формами взаимосвязей с другими предприятиями и т.п.

В качестве оправдавшего себя механизма повышения эффективности деятельности предприятий проявила себя оптимизация документооборота, осуществляемая с участием консультантов при активном использовании внутренних возможностей самих предприятий.

Электронные системы управления потоками клиентов помогают изменить и повысить качество обслуживания. В случае необходимости позволяют организовать запись посетителей на прием по времени и дате. Системы электронной очереди позволяют на основе полученных в процессе работы данных оптимизировать обслуживание или разрабатывать новые методики, а также оперативно вносить коррективы.

Следствием применения систем электронных очередей является улучшение общего климата обслуживания и более высокий коэффициент работы персонала учреждения.

Система электронной очереди отличается от систем «вызова клиента» тем, что позволяет ввести гибко настраиваемый алгоритм управления потоком клиентов, вести учёт и статистику работы операторов и интенсивности потока, что позволяет эффективно планировать нагрузку. Кроме этого, в системе предусмотрены функции управления настройками системы и её исполнительными модулями. Данный тип систем можно отнести к on-line системам, работающим и управляемым в реальном времени.

Целью дипломного проекта является разработка и создание автоматизированной системы управления формированием электронной очереди.

Программный модуль формирования электронной очереди должен обеспечивать выполнение следующих основных функций:

- а) запись в очередь на прием к начальникам и заместителям начальников организации;
- б) управление очередью секретарем, заместителями и начальниками, а именно удаление, изменение порядка;
- в) выставление приоритета очереди, а именно заместители руководителя имеют право попадания на прием перед рядовыми сотрудниками.

Потребность в разработке программного модуля формирования электронной очереди, обусловлена жесткой конкуренцией на рынке информационных технологий и необходимостью обеспечить более высокую производительность труда, большую надежность и достоверность информации, лучшую ее сохранность.

Таким образом, применение разрабатываемого программного модуля формирования электронной очереди является целесообразным и необходимым современных в условиях.

Объектом для анализа экономической эффективности деятельности выбран разрабатываемый программный модуль формирования электронной очереди.

1 АНАЛИЗ ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1.1 Роль электронных систем очередей

Системы управления очередью помогают избежать скопления людей в местах приёма посетителей и организовать «цивилизованный» порядок обслуживания клиентов. В основном применяется для распределения, оптимизации и учета клиентов в очереди и вызова их к кассам с помощью звукового сигнала и визуального отображения индивидуального номера очереди клиента. Наиболее типичные применения подобных систем: кассы по продаже ж/д и авиабилетов, сервис центры по оказанию услуг на вокзалах, кассы приема платежей, государственные учреждения, офисы крупных фирм, банки, пункты регистрации автотранспорта и другие.

Конфигурация и логика работы подобных систем разрабатывается исходя из конкретных потребностей заказчика.

В состав систем могут входить различные компоненты: принтеры для печати порядкового номера клиента, пульта управления для операторов и администратора, групповые информационные табло, табло оператора, интерфейс с компьютером и т.д. Дополнительно возможно сохранение информации о работе системы с функциями контроля и подсчета статистики, что во многих случаях позволяет иметь полную информацию по загруженности пункта обслуживания клиентов, работе отдельных операторов и др.

Над каждым рабочим местом оператора и над каждой кассой установлено световое табло, отображающее текущий номер очереди клиента, обслуживаемого в данный момент. Звуковой сигнал сопровождает вызов нового клиента.

Система, позволяет не только эффективно управлять очередью, но и одновременно заботится о том, чтобы каждый клиент получил индивидуальное обслуживание в комфортной обстановке. В любой момент сотрудники могут получить практически любую статистическую информацию об обслуживании клиентов и работе операторов. Использование системы управления очередью позволяет ускорить время обслуживания посетителей более чем вдвое, что выгодно для организации и клиентов.

Сенсорный экран (от англ. “touchscreen” – «экран для прикосновений») – экран, позволяющий выбрать нужную услугу, нажав пальцем непосредственно на экран, а точнее, на электронную клавишу с названием желаемой услуги.

Ждать не любит никто – это всегда негативно влияет на качество услуги. Тем не менее, это явление не такое уж редкое в повседневной жизни. Фактически в деятельности любой организации возникают очереди.

Эмоции, которые выносят клиенты из очередей, во многом определяют их отношение к уровню полученного сервиса в целом и перспективы повторного обслуживания организацией.

Лимитированная производительность всегда является слабым местом для сферы услуг, поскольку услуги не могут производиться заранее и сохраняться до момента, когда будут востребованы. Современная клиент-ориентированная организация должна пытаться разработать стратегию, приносящую порядок, предсказуемость и справедливость в очередь.

Применение системы электронной очереди для обслуживания клиентов обеспечивает:

- внедрение современной технологии обслуживания клиентов, распределение и оптимизацию потоков клиентов;
- среднее время ожидания на каждый тип обслуживания по продолжительности рабочего дня, недели (в виде графиков и диаграмм по различным отрезкам времени);
- время обслуживания одного клиента отдельным оператором для определения относительной эффективности работы каждого оператора (для оценки производительности труда и планирования работы);
- статистика по типам обслуживания, позволяющая оперативно определять загрузку оператора и распределять их по типу обслуживания;
- получение оперативной информации в реальном масштабе времени о текущей работе каждого оператора, количестве работающих операторов, количестве обслуженных клиентов, количестве клиентов, ждущих в очереди и др.

Существуют различные модификации системы от простой, обеспечивающей только продвижение очереди до системы с расширенными сервисными функциями.

Находясь в «зоне ожидания», посетители могут подготовить необходимые документы, ознакомиться с образцами правильного заполнения бланков и заявлений или просто просмотреть имеющиеся здесь справочники и почитать газеты. При этом отсутствует «живая» очередь и связанные с ней отрицательные моменты.

Звуковой сигнал, привлекающий внимание при изменении информации на табло, не даст возможности клиентам пропустить свою очередь. Имеется даже возможность печатать на талонах ориентировочное время обслуживания или среднее время ожидания.

Благодаря этому клиент может при желании отлучиться из зала ожидания по своим делам и вернуться к моменту обслуживания. Это создает дополнительное удобство при большом потоке посетителей.

Еще одним достоинством системы является наличие программы статистического учета. Она фиксирует работу каждого оператора, позволяет анализировать собираемые данные и соответствующим образом планировать и изменять работу отдела в зависимости от дня недели, сезона и потребностей предприятия.

Эта программа собирает данные о количестве клиентов, обслуженных каждым работником и отделом в целом в определенный момент времени – час, день, неделю, месяц и т.д., а также время обслуживания и ожидания.

Необходимость автоматизации подхода к решению проблем, связанных с обслуживанием клиентов, обуславливает необходимость систем управления очередями.

Электронная очередь имеет два главных преимущества:

- комфорт клиента: клиент имеет возможность записаться на прием, получать актуальную информацию о состоянии очереди, а также дать оценку качества работы сотрудников компании.
- комфорт персонала: система управления очередью дает четкую картину распределения нагрузки на сотрудников и позволяет грамотно построить график работы без стрессовых нагрузок.

В практике разработки и реализации систем электронного управления очередью существует большой диапазон их функциональных и инженерных решений.

1.2 Оборудование и программное обеспечение электронных очередей

В состав оборудования для традиционной системы управления очередью могут входить: пункт регистрации, центральное информационное табло, мини-табло для каждого окна оператора, сервер (в качестве которого может выступать сенсорный киоск), система веб-регистрации, система администрирования.

Пульт регистрации – устройство, позволяющее клиенту выбрать услугу и получить номер очереди (талон с номером). Пульты регистрации бывают сенсорные и кнопочные. Сенсорные пульта регистрации помимо функции выдачи талонов, могут иметь расширенную функциональность, например, встроенную справочную систему или систему оценки качества обслуживания (отзывы). Самый простой вариант реализации данного компонента системы – лента с заранее напечатанными номерами очереди.



Рисунок 1.1 – Пульт выбора услуг

На экране пульта выбора услуг, пример которого представлен на рисунке 1.1, возможен вывод любой графической информации, в частности, с использованием элементов фирменного стиля организации/учреждения, а также вывод различных видов рекламного и информационного контента (бегающая строка, видеоролик).

С помощью конфигурирования графического интерфейса можно в любое время изменить вид и комбинацию отображаемых на экране терминала элементов [1].

Пульт оператора – устройство, применяемое для вызова клиентов из очереди, также оно позволяет перенаправить клиента к другому оператору или завершить обслуживание. Возможны программная либо аппаратная реализации пульта оператора.

Физический пульт оператора, пример которого представлен на рисунке 1.2, применяется в случае, если политика безопасности учреждения не допускает использования существующей локальной вычислительной сети и строго регламентирует состав программ, которые могут быть установлены на персональные компьютеры операторов.



Рисунок 1.2 – Пульт оператора

Физический пульт оператора устанавливается на рабочем месте оператора, и позволяет выполнять основные функции по вызову посетителей:

- вызов посетителя;
- повтор вызова;
- отложенный вызов посетителя;
- вызов посетителя по номеру;
- переназначение списка услуг посетителю.

Если политика безопасности учреждения допускает использование существующих на объекте локальной вычислительной сети и персональных компьютеров операторов, то возможна программная реализация пульта оператора, возможности которого гораздо шире, чем у физического пульта [1].

Главное информационное табло расположено в зоне ожидания в удобном для обзора месте и позволяет посетителю контролировать свое положение в очереди [2].

При вызове очередного посетителя на главное табло выводится номер этого посетителя и номер окна, к которому ему необходимо подойти.

Главное информационное табло расположено в зоне ожидания в удобном для обзора месте и позволяет посетителю контролировать свое положение в очереди.

При вызове очередного посетителя на главное табло выводится номер этого посетителя и номер окна, к которому ему необходимо подойти.

Как правило, главное информационное отображает список последних вызванных клиентов в порядке FIFO. FIFO (акроним First In, First Out - «первым пришёл - первым ушёл») – способ организации и манипулирования данными относительно времени и приоритетов. Это выражение описывает принцип технической обработки очереди или обслуживания конфликтных требований путём упорядочения процесса по принципу: «первым пришёл - первым обслужен». Тот, кто приходит первым, тот и обслуживается первым, пришедший следующим ждёт, пока обслуживание первого не будет закончено, и так далее.

На светодиодное информационное табло, пример которого приведен на рисунке 1.3, выводится информация об истории вызовов в формате «номер посетителя – номер окна». Нижняя строка может служить для вывода информации в виде бегущей строки. Если главное табло реализовано на основе телевизионной панели, то на него возможен вывод любой текстовой информации (последний вызов посетителя, история вызовов, рекламная и информационная информация организации либо учреждения), а также вывод любой графической информации и любого мультимедиа-контента (бегущая строка, видеоролик) [1].



Рисунок 1.3 – Светодиодное табло

Табло оператора – устройство, позволяющее продублировать информацию главного табло по конкретному вызванному клиенту [2]. В качестве табло оператора применяются мониторы, телевизоры и светодиодные табло. Во многих случаях от применения данного устройства можно отказаться, поскольку

оно дублирует информацию главного табло. В таком случае рабочее место оператора помечают табличкой с номером рабочего места [1].

Табло оператора может быть выполнено на основе матричных светодиодных табло.

Система голосового оповещения – это программно-аппаратное решение, позволяющее дублировать функциональность главных табло и табло оператора. При вызове нового клиента система произносит номер его очереди и номер окна (пульта оператора), где его будут обслуживать. Электронная очередь может функционировать только в режиме голосовых оповещений, то есть без главных табло и табло оператора [1].

Система оценки качества – это программно-аппаратное решение, дополняющее функциональность электронной очереди. Оно позволяет оценить результаты оказания услуг по заранее заданной шкале. Данная функциональность может быть реализована как с помощью отдельных аппаратных устройств, таких как аппаратные пульта системы оценки качества, так и с помощью интеграции с существующими компонентами электронной очереди, такими как виртуальный пульт оператора или пульт регистрации.



Рисунок 1.4 – Аппаратный пульт системы оценки качества

Аппаратный пульт системы оценки качества – это устройство с несколькими кнопками, позволяющее выбрать один из вариантов оценки качества оказания услуги. Как правило, интерфейс устройства очень интуитивен – кнопки маркируются смайликами. Пример интерфейса аппаратного пульта системы оценки качества приведен на рисунке 1.4. Нажимая ту или иную кнопку, посетитель оценивает качество услуг, предоставленных сотрудником организации. Так же может быть реализована система оценки качества услуг при выходе посетителя. Например, для разблокировки турникета, находящегося при выходе, от посетителя требуется поднести к считывающему устройству свой номерок со штрих-кодом и оценить качество предоставленных услуг, нажав соответствующую сенсорную кнопку. Либо, нажав соответствующую клавишу, вызвать книгу отзывов и написать развернутый отзыв, жалобу или рекомендацию [1].

1.3 Обзор аналогов существующих систем

1.3.1 Система управления очередями AKIS Micro

AKIS Micro является эффективным решением для организаций, в которых все виды услуг объединены в одну очередь [3]. Аппаратное обеспечение системы AKIS Micro (рисунок 1.5) состоит из:

- пульта регистрации;
- светодиодных экранов;
- рабочих дисплеев операторов.



Рисунок 1.5 – Аппаратное обеспечение системы AKIS Micro

Регистрация посетителей осуществляется контроллером, встроенным в пульт регистрации. Контроллер также управляет печатью талонов, которая осуществляется в автоматическом режиме на компактном диспенсере. Светодиодные экраны указывают направление движения очереди, а также позволяют пользователям отслеживать движение очереди [4].

Система поддерживает до шести рабочих мест операторов. Каждый оператор может быть оборудован рабочим светодиодным дисплеем для отображения номера вызываемого клиента.

1.3.2 Система управления очередями Meta-Q

Система управления потоками посетителей Meta-Q позволяет оптимизировать процесс клиентского обслуживания в самых разных сферах деятельности.

Система управления очередями Meta-Q – инструмент организации клиентского потока и повышения качества клиентского сервиса для организаций,

деятельность которых связана с предоставлением услуг. Meta-Q – высоко конфигурируемая система с модульной архитектурой [5]. Схема Meta-Q приведена на рисунке 1.6.

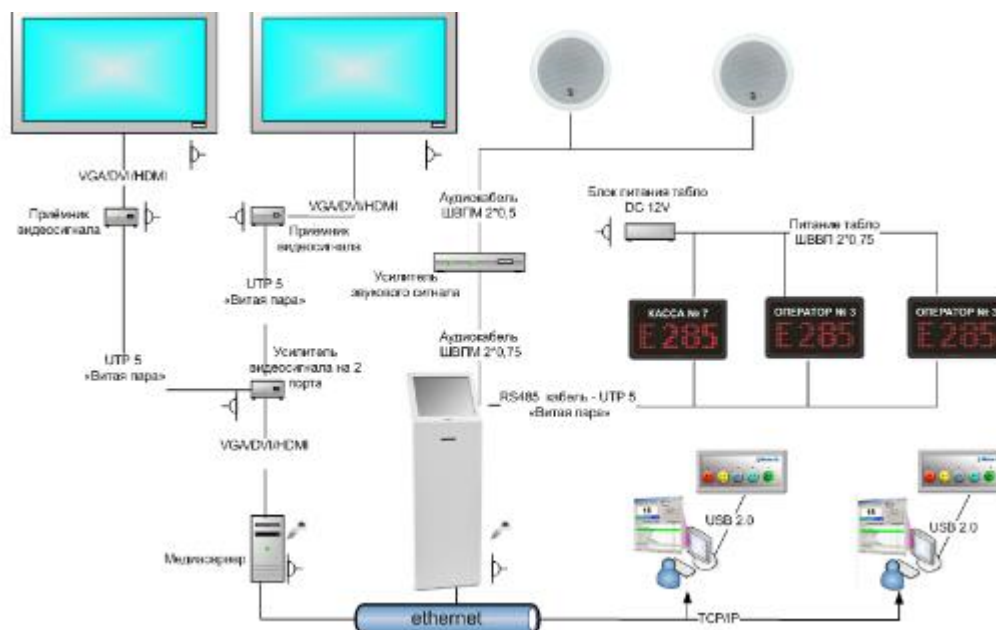


Рисунок 1.6 – Архитектура системы Meta-Q

Регистрация посетителей осуществляется программным модулем регистрации в очередь [6]. Модуль регистрации может быть представлен следующими аппаратными компонентами:

- сенсорный терминал;
- кнопочный терминал;
- персональный компьютер администратора с термопринтером;
- сенсорная панель с термопринтером.

Навигация и информирование пользователей осуществляется главным информационным табло. Рабочие места операторов также могут оснащаться информационными табло. Система поддерживает следующие виды информационных табло:

- матричное светодиодное табло;
- сегментное светодиодное табло;
- жидкокристаллическая панель.

Для управления потоком посетителей системой предусмотрены пульта операторов очереди, устанавливаемые на рабочих местах операторов. Пульт оператора существует в следующих видах:

- программный модуль «Виртуальный пульт», устанавливаемый на персональный компьютер;
- упрощенный аппаратный пульт;
- полнофункциональный аппаратный пульт с функциями перенаправления и вызова по номеру.

Также, система Meta-Q может оснащаться следующими дополнительными функциями:

- звуковое сопровождение вызова;
- голосовое сопровождение вызова;
- модуль предварительной регистрации с помощью телефона и сети Интернет;
- модуль централизованной статистики по нескольким подразделениям;
- расширенная авторизация клиента с помощью сканера штрих-кодов, считывателя смарт-карт, считывателя магнитных карт, ввода данных с помощью клавиатуры;
- поддержка удалённого администрирования системы;
- поддержка вывода медиаконтента на одном экране с информацией о состоянии очереди;
- модуль оценки качества обслуживания.

1.3.3 Система управления очередью IBA QueueMASTER

Система управления очередью IBA QueueMASTER предназначена для управления потоками пользователей, обеспечения удобства и комфорта клиентам, позволяет оптимизировать работу по обслуживанию клиентов [7]. Схема системы приведена на рисунке 1.7.

Функции системы управления очередью IBA QueueMASTER включают:

- распределение потока клиентов по зонам обслуживания;
- индивидуальное обслуживание каждого клиента;
- возможность приоритетного обслуживания клиентов;
- возможность удаленной регистрации клиента на обслуживание посредством сети Интернет;
- предоставление рекламы на информационном табло, талоне.

Система позволяет осуществлять приоритетное обслуживание клиентов, таких как ветераны войны и труда, постоянные клиенты, VIP-клиенты.

1.4 Требования к разрабатываемому программному модулю

Разрабатываемый в ходе дипломного проекта программный модуль предназначен для работы с электронными очередями. В отличие от рассмотренных выше аналогов систем управления очередями, разрабатываемый модуль предназначен для использования внутри организаций для управления потоками посещений руководителей и заместителей руководителей.

Предполагаемых пользователей модуля можно разделить на следующие типы:

- рядовой сотрудник;
- руководитель;

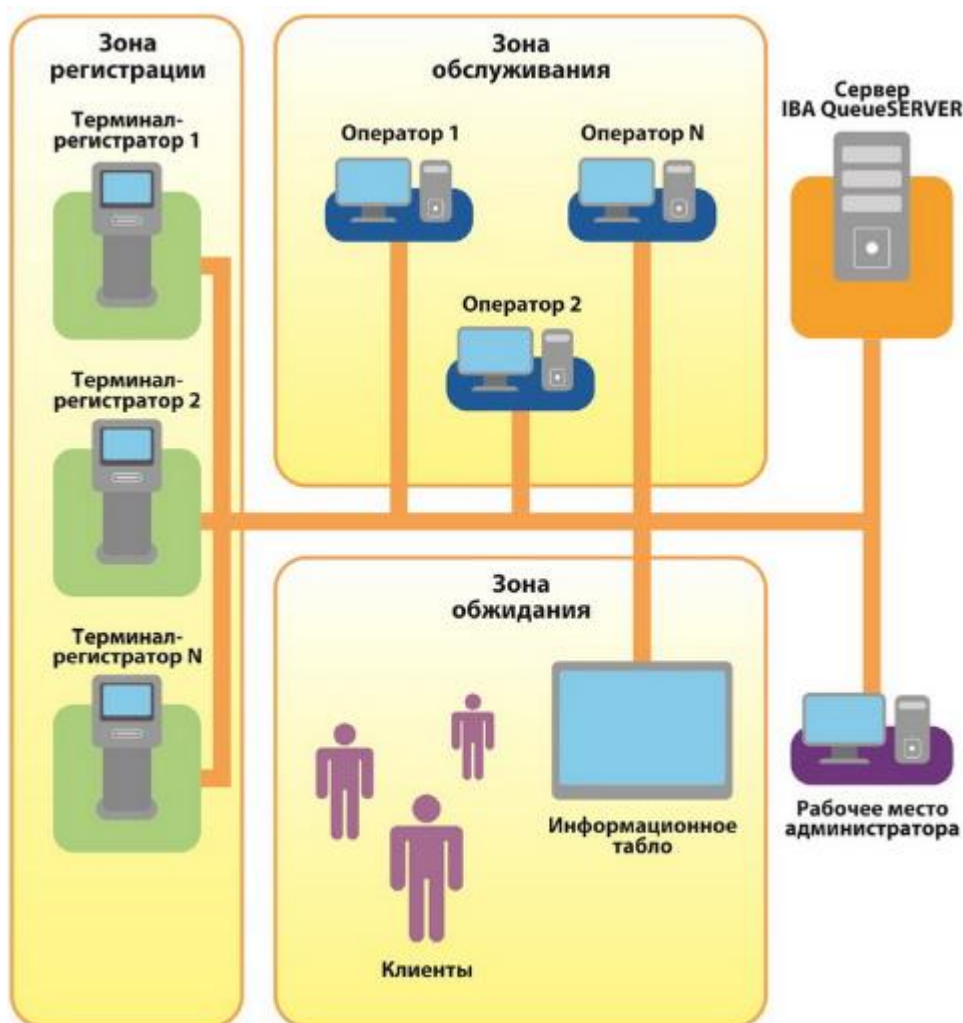


Рисунок 1.7 – Структурная схема системы IBA QueueMASTER

- заместитель руководителя;
- делопроизводитель или секретарь;
- администратор.

Базовые функции модуля, доступ к которым имеют все пользователи, включают:

- просмотр доступных очередей;
- постановка в очередь;
- просмотр вхождений в очереди и текущих позиций в этих очередях;
- просмотр и изменение собственной учетной записи.

В дополнение к базовым функциям, руководители и заместители руководителей имеют доступ к следующим функциям:

- просмотр собственной очереди;
- удаление вхождений в собственную очередь;
- изменение порядка следования сотрудников в очереди;
- предоставление доступа к очереди делопроизводителям;
- отзыв доступа к очереди делопроизводителям.

При постановке в очередь, позиция в очереди определяется с учетом приоритетов. При этом, заместители руководителей имеют приоритет высший, чем рядовые сотрудники; руководители имеют наивысший приоритет.

Владельцы очередей (руководители и заместители руководителей) имеют возможность предоставлять доступ к своей очереди делопроизводителям. Делопроизводитель, обладающий доступом к очереди, может просматривать и управлять очередью аналогично владельцу очереди.

Администраторы имеют доступ ко всем функциям, доступным сотрудникам, а также:

- изменение очередей;
- просмотр учетных записей;
- изменение свойств учетной записи;
- изменение типов учетных записей.

Доступ в систему осуществляется с помощью логина и пароля. Логин и пароль указывается пользователем при регистрации. Пароль может в последствии изменяться пользователем. Также, администратор может управлять паролем. Клиентское приложение обладает функцией сохранения логина и пароля после успешного входа в систему, и последующей вставки в соответствующие поля.

Программный модуль предназначен для эксплуатации под управлением операционных систем семейства Microsoft Windows с установленной программной платформой .NET Framework. Серверная часть программного модуля предназначена для эксплуатации под управлением веб-сервера Microsoft Internet Information Services. Программный модуль поддерживает работу с данными под управлением СУБД Microsoft SQL Server.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Алгоритм работы системы управления электронными очередями

2.1.1 Алгоритм работы традиционных систем управления электронными очередями

Схема алгоритма работы традиционных систем управления электронными очередями представлена на рисунке 2.1.

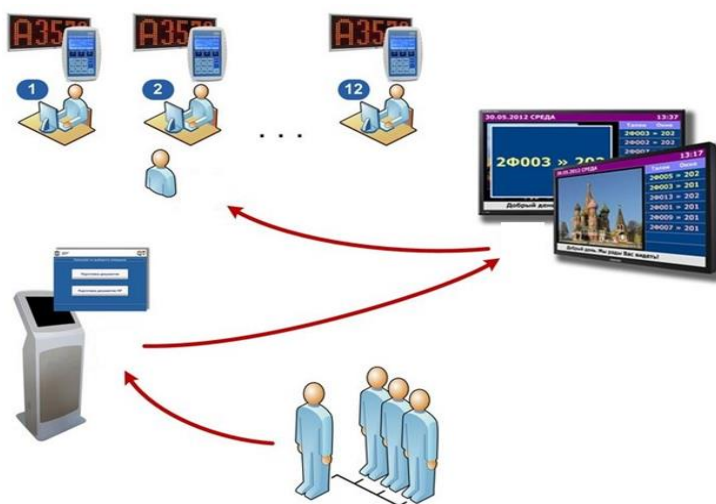


Рисунок 2.1– Алгоритм работы системы со стороны клиента

Посетитель подходит к пульту выбора вопросов и выдачи талонов и формирует группу вопросов, нажимая соответствующие клавиши. Выбираемые клавиши подсвечиваются другим цветом. Повторное нажатие на подсвеченную клавишу отменяет выбор, и клавиша гаснет.

Вопросы сформированы в группы, объединяющие несколько операций под одним логическим названием. Это сделано для того, чтобы сократить время выбора посетителем перечня вопросов. Перечень вопросов, а также состав групп операций, соответствующих вопросам может быть изменен в любое время с рабочего места администратора системы.

Сформировав список вопросов, посетитель нажимает клавишу получения талона. Пульт регистрации выдает ему талон, на котором напечатан список вопросов, индивидуальный номер очереди посетителя и штрих-код, соответствующий этому номеру. Также, на талоне может размещаться другая информация, такая как информация рекламного характера или расписание работы офиса. Так же на талоне напечатаны инструкции посетителю, что он может сделать в период ожидания очереди, для ускорения процесса обслуживания, например, заполнить бланки или подготовить документы.

Посетитель проходит в зону ожидания и следит за информационным табло и слушает сообщения звукового информатора.

На рабочем месте оператора установлен пульт оператора со сканером штрих-кода. Пульт оператора позволяет:

- вызывать следующего посетителя;
- повторять вызов еще раз, если посетитель не появился;
- отложить обслуживание посетителя;
- перенаправить посетителя в другое окно;
- открыть или закрыть рабочий день;
- приостановить работу.

Список вопросов, по которым может работать тот или иной оператор задается с рабочего места администратора системы и может быть изменен в любой момент времени.

При наступлении очереди посетителя, он приглашается сообщением звукового информатора. Сообщение содержит номер очереди посетителя и номер оператора. Сообщение звукового информатора дублируется на информационных табло сообщениями аналогичного содержания.

Приглашенный посетитель подходит к рабочему месту оператора и передает ему талон для регистрации. Номер приглашенного посетителя и номер подошедшего посетителя отображаются в рабочем окне программы. Если номера не совпадают, то оператор должен отказать посетителю в приеме. Если номера совпадают, то в рабочем окне программы отображается информация о подошедшем посетителе, указанная им при регистрации. Используя эту информацию, оператор может правильно проинформировать посетителя о его месте в очереди.

Если подошел не тот посетитель или не подошел никто, оператор имеет возможность повторить вызов.

Если к оператору не подошел вызываемый посетитель и после повторного вызова, то оператор имеет возможность отложить вызов. В этом случае номер вызываемого посетителя переносится в отложенную очередь. После этого система ожидает заданный промежуток времени и повторяет вызов посетителя из отложенной очереди, используя сообщение звукового информатора и информационное табло.

Если к оператору и после этого сообщения не подошел вызываемый посетитель, оператор снова выбирает опцию отложенного вызова. После нескольких неудачных вызовов посетителя, система удаляет номер посетителя из очереди.

Схема взаимодействия сотрудников с клиентами при помощи электронной очереди дает целый ряд преимуществ:

- комфортное обслуживание для клиентов: получение необходимых справок о предоставляемых услугах, запись на прием к сотруднику на время, которое будет заранее известно и четко определено, возможность оценки качества обслуживания;

– комфортная работа персонала: благодаря статистическим данным, поступающим на административный пульт, о количестве и плотности посетителей, руководство имеет возможность распределить рабочее время своих сотрудников максимально эффективно и уменьшить стрессовые нагрузки.

У администратора есть возможность в реальном режиме времени отслеживать как общее, так и детализированное состояние очереди – по операторам, по услугам, по времени обслуживания, по времени ожидания в очереди и многим другим параметрам. Это позволяет принимать оперативные решения по изменению необходимого и достаточного в данное время дня (или другого периода) количества сотрудников.

Помимо статистики, имеющей непосредственно отношение к обслуживанию очереди, параллельно производится накопление статистики по рабочему времени каждого оператора (обеда, технические перерывы и т.п.) Система управления электронной очередью генерирует множество отчетов, на основании которых можно предпринимать те или иные управленческие решения.

Система контролирует конец рабочего дня и, в зависимости от настроек, контролирует процесс занятия очереди последним посетителем, основываясь на статистических данных, либо переносит очередность на следующий день.

2.1.2 Алгоритм работы разрабатываемого модуля управления электронными очередями

Как описано в спецификации требований, разрабатываемый программный модуль отличается от традиционных систем управления электронными очередями. Все стадии взаимодействия с клиентом происходят в рамках единственного клиентского приложения, запущенного на персональном компьютере. При этом, различные виды взаимодействия с клиентом представлены в виде форм и окон графического пользовательского интерфейса клиентского приложения.

Аналогом информационного табло в разрабатываемом программном модуле выступает форма вхождений в очереди. На форме вхождений в очереди сотрудник имеет возможность просмотреть очереди, в которые он входит, и возможность выхода из очередей. Также, из формы вхождений в очереди возможно перейти к окну вхождения в очередь.

Окно вхождения в очередь является аналогом пульта регистрации в традиционной системе управления электронными очередями. С помощью окна вхождения в очередь пользователь может выбрать очередь для вхождения и войти в нее. После этого, очередь появится на форме вхождений в очереди.

В качестве аналога пульта оператора разрабатываемый программный модуль содержит форму управления очередью. С помощью этой формы владельцы очередей (руководители и заместители руководителей), делопроизво-

дители, имеющие доступ к очереди, и администраторы имеют доступ к удалению сотрудников из очередей, переопределению порядка следования вхождений в очередях.

Администраторы имеют доступ к формам администрирования очередей и учетных записей. С помощью этих форм, администратор может выбрать очередь или учетную запись и получить доступ к форме управления выбранным элементом.

2.2 Разработка модели вариантов использования

Диаграмма вариантов использования представлена на рисунке 2.2.

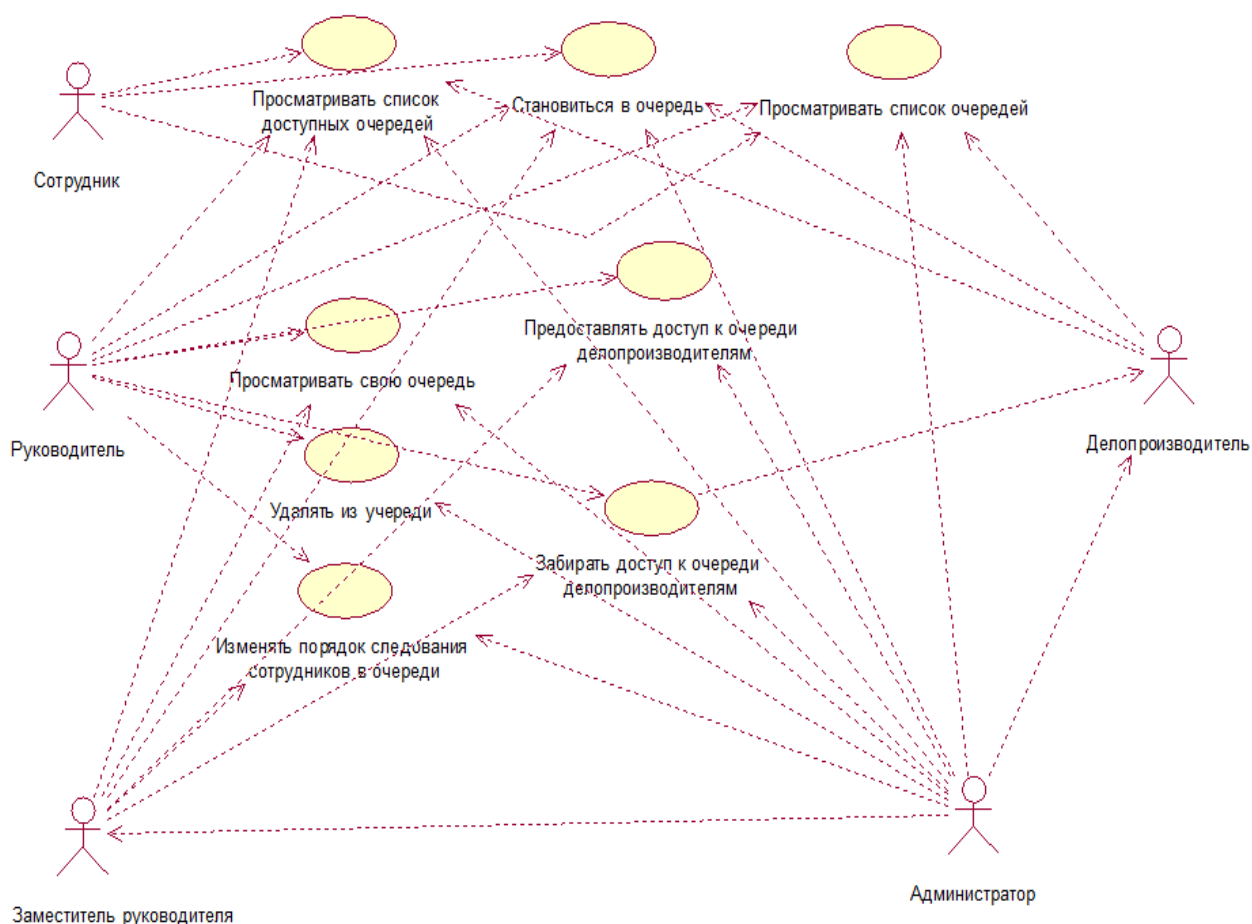


Рисунок 2.2 – Диаграмма вариантов использования

Модель предметной области предусматривает следующие типы пользователей:

- сотрудник;
- делопроизводитель;
- заместитель руководителя;
- руководитель;
- администратор.

Сотрудник – базовый тип пользователя. Все пользователи имеют доступ к функциям сотрудника, т.е. другие типы пользователей наследуют функции сотрудника.

Сотрудник имеет доступ к следующим функциям:

- просматривать список очередей, в которые входит сотрудник;
- просматривать список всех доступных очередей;
- становиться в очередь;
- выходить из очереди;
- просматривать свою учетную запись;
- редактировать свою учетную запись;
- изменять пароль.

Руководитель наследует все функции сотрудника, а также имеет доступ к следующим функциям:

- просматривать собственную очередь;
- удалять вхождения из собственной очереди;
- переопределять порядок вхождений в собственную очередь;
- предоставлять доступ к собственной очереди делопроизводителю;
- отзываться доступ делопроизводителя к собственной очереди.

Заместитель руководителя идентичен в функциях руководителю. Заместитель отличается от руководителя при обработке вхождений в очередь системой приоритетов.

Делопроизводитель наследует все функции сотрудника. Если владелец очереди предоставляет делопроизводителю доступ к очереди, делопроизводитель также получает доступ к следующим функциям:

- просматривать доступную очередь;
- удалять вхождения из доступной очереди;
- переопределять порядок вхождений в доступную очередь.

Администратор наследует все функции сотрудника, а также имеет доступ к следующим функциям:

- просматривать список доступных очередей;
- просматривать очереди;
- удалять вхождения из очередей;
- переопределять порядок вхождений в очередь;
- предоставлять доступ к очередям делопроизводителям;
- отзываться доступ делопроизводителей к очередям;
- просматривать список доступных учетных записей;
- просматривать учетные записи;
- редактировать свойства учетных записей;
- редактировать типы учетных записей;
- редактировать должности сотрудников;
- изменять пароли учетных записей.

2.3 Разработка информационной модели предметной области

Информационная модель предметной области представлена на рисунке 2.3.

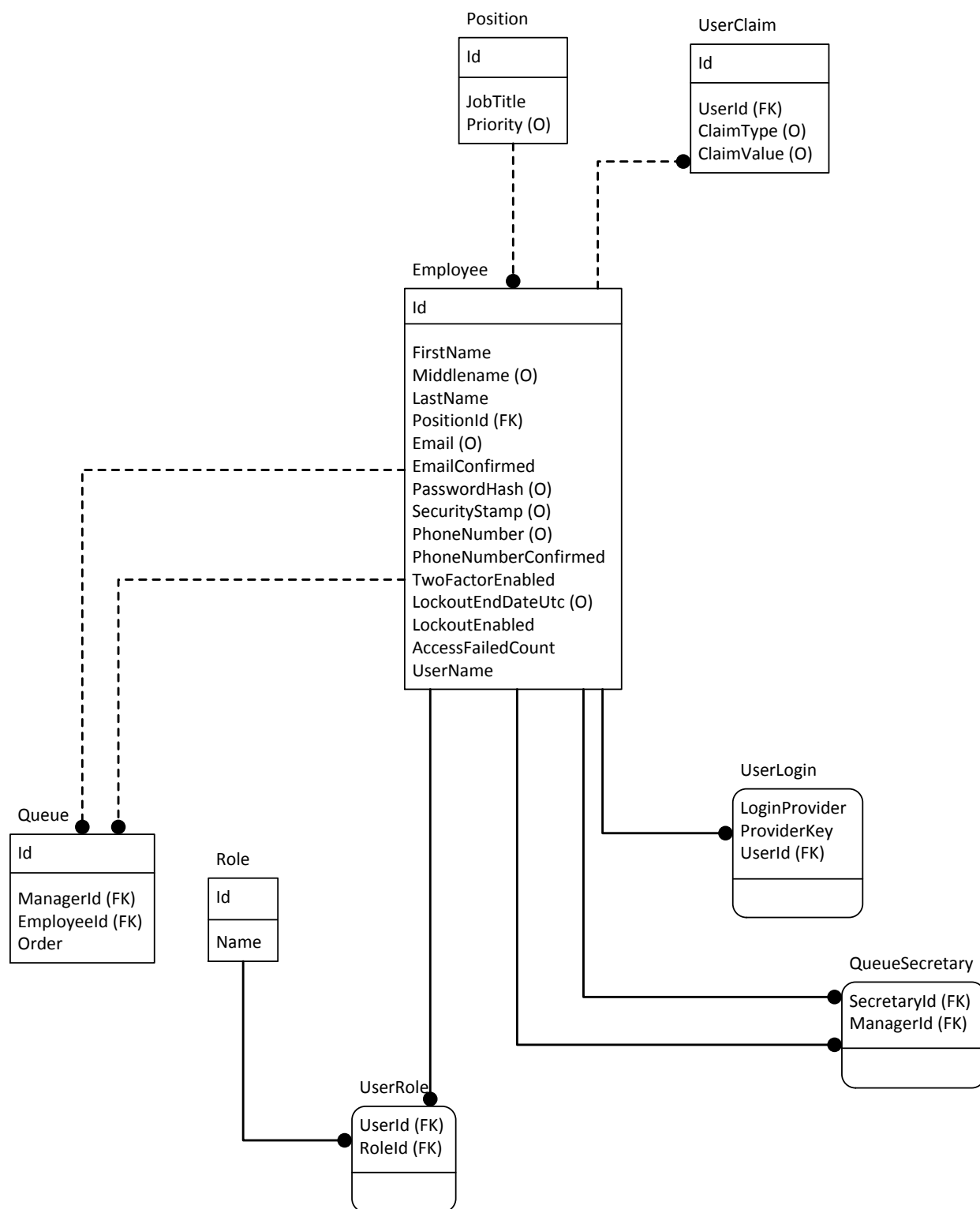


Рисунок 2.3 – Информационная модель системы

Информационная модель предметной области содержит следующие сущности:

- Employee;
- Position;
- Queue;
- QueueSecretary;
- Role;
- UserRole;
- UserLogin;
- UserClaim.

Сущность Employee содержит информацию о пользователях и их учетных записях. Сущность Employee обладает следующими атрибутами:

- Id – уникальный идентификатор сущности
- FirstName – имя;
- MiddleName – отчество;
- LastName – фамилия;
- PositionId – ссылка на сущность Position, представляющую должность пользователя;
- Email – адрес электронной почты;
- EmailConfirmed – флаг подтверждения адреса электронной почты;
- PasswordHash – пароль учетной записи в захешированном виде;
- SecurityStamp – номер версии сущности, изменяющаяся с каждым изменением сущности;
- PhoneNumber – номер телефона;
- PhoneNumberConfirmed – флаг подтверждения номера телефона;
- TwoFactorEnabled – флаг включенной опции двухкомпонентной авторизации;
- LockoutEndDateUtc – время истечения блокировки учетной записи;
- LockoutEnabled – флаг блокировки учетной записи;
- AccessFailedCount – количество неудачных попыток авторизации;
- UserName – логин.

Сущность Position представляет собой должность и обладает следующими атрибутами:

- Id – уникальный идентификатор сущности
- JobTitle – название;
- Priority – приоритет должности.

Сущность Queue представляет собой элемент очереди и обладает следующими атрибутами:

- Id – уникальный идентификатор сущности
- ManagerId – ссылка на сущность Employee, представляющую владельца очереди;
- EmployeeId – ссылка на сущность Employee, представляющую сотрудника, стоящего в очереди;
- Order – порядковый номер вхождения в очереди.

Сущность QueueSecretary – связующая сущность для поддержки связи многие-ко-многим между Employee и Employee. Используется для установления доступа делопроизводителей к очередям. QueueSecretary содержит следующие атрибуты:

- SecretaryId – ссылка на сущность Employee, представляющую делопроизводителя, обладающего доступом к очереди;
- ManagerId – ссылка на сущность Employee, представляющую владельца очереди.

Сущность Role представляет собой роль и обладает следующими атрибутами:

- Id – уникальный идентификатор сущности
- Name – название.

Сущность UserRole – связующая сущность для поддержки связи многие-ко-многим между Employee и Role. UserRole обладает следующими атрибутами:

- UserId – ссылка на сущность Employee;
- RoleId – ссылка на сущность Role.

Сущность UserLogin содержит информацию для авторизации пользователя с помощью сторонних систем и обладает следующими атрибутами:

- LoginProvider – идентификатор сторонней системы;
- RoleId – ключ, уникально идентифицирующий пользователя в сторонней системе;
- UserId – ссылка на сущность Employee, представляющую пользователя.

3 РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ

3.1 Описание использованных инструментов

3.1.1 Программная платформа Microsoft .NET Framework

Разрабатываемый в ходе данного дипломного проекта программный модуль базируется на технологии Microsoft .NET Framework. .NET Framework – это программная платформа для построения приложений на базе семейства операционных систем Windows, а также многочисленных операционных систем производства не Microsoft, таких как Mac OS X и различные дистрибутивы Unix и Linux [8].

Основные преимущества .NET включают в себя:

- обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения;
- обеспечение среды выполнения кода, минимизирующей конфликты при развертывании программного обеспечения и управлении версиями;
- обеспечение среды выполнения кода, гарантирующей безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем;
- обеспечение среды выполнения кода, исключающей проблемы с производительностью сред выполнения сценариев или интерпретируемого кода;
- обеспечение единых принципов работы разработчиков для разных типов приложений, таких как приложения Windows и веб-приложения;
- разработка взаимодействия на основе промышленных стандартов, которое обеспечит интеграцию кода платформы .NET Framework с любым другим кодом.

.NET состоит из нескольких ключевых компонентов:

- среда выполнения CLR;
- система типов CTS;
- спецификация языков программирования CLS.

Основой платформы .NET Framework является среда CLR [9]. Среду выполнения можно считать агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надежность. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом [10]. Библиотека классов является комплексной объектно-ориентированной коллекцией допускающих

повторное использование типов, которые применяются для разработки приложений.

На рисунке 3.1 демонстрируется взаимосвязь среды CLR и библиотеки классов с пользовательскими приложениями и всей системой.

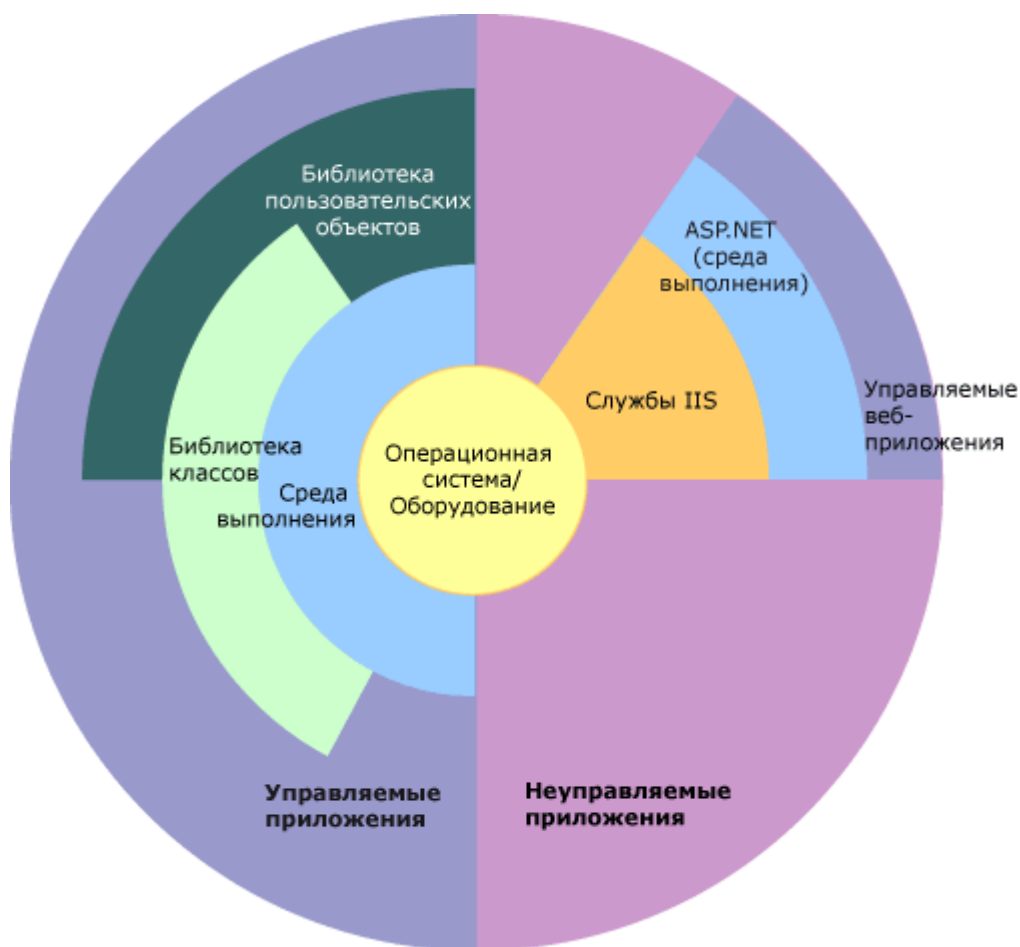


Рисунок 3.1 - .NET Framework в контексте

Другим строительным блоком платформы .NET является общая система типов Common Type System или CTS [8]. В спецификации CTS полностью описаны все возможные типы данных и все программные конструкции, поддерживаемые исполняющей средой. Кроме того, в CTS показано, как эти сущности могут взаимодействовать друг с другом, и указано, как они представлены в формате метаданных .NET.

Также существует общезыковая спецификация Common Language Specification или CLS, в которой описано подмножество общих типов и программных конструкций, которые должны поддерживать все языки программирования для .NET.

В дополнение к среде CLR и спецификациям CTS и CLS, платформа .NET предоставляет библиотеку базовых классов, которая доступна всем языкам программирования .NET. Библиотека классов платформы .NET Framework представляет собой коллекцию типов, которые тесно интегрируются со средой

CLR. Библиотека классов является объектно-ориентированной; предоставляя типы, из которых управляемый код пользователя может наследовать функции. Это не только упрощает работу с типами .NET Framework, но также уменьшает время, затрачиваемое на изучение новых средств платформы .NET Framework. Кроме того, компоненты независимых производителей можно легко объединять с классами платформы .NET Framework.

Типы .NET Framework позволяют решать типовые задачи программирования, включая работу со строками, сбор данных, подключения к базам данных и доступ к файлам. В дополнение к этим обычным задачам библиотека классов содержит типы, поддерживающие многие специализированные сценарии разработки. Некоторые технологии .NET, такие как Windows Presentation Foundation, ASP.NET Web API, Entity Framework были использованы при разработке описываемого программного модуля, и будут описаны ниже.

3.1.2 Язык программирования C#

C# – простой, современный, объектно-ориентированный и типобезопасный язык программирования. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java.

C# отвечает требованиям спецификации Common Language Specification, рассмотренной выше, что обеспечивает его совместимость с .NET Framework и компонентами, написанными на других CLS-совместимых языках [11].

C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и не требующих отдельной документации пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов – это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

Ключевые особенности и возможности C# включают:

- строгую статическую типизацию;
- полиморфизм;
- перегрузку операторов;
- делегаты;
- атрибуты;
- события;
- свойства;
- обобщённые типы и методы;
- итераторы;
- анонимные функции с поддержкой замыканий;

- LINQ;
- обработку исключений.

Полиморфизм – принцип объектно-ориентированного программирования, поддерживаемый C#. Во время выполнения объекты производного класса могут обрабатываться как объекты базового класса в таких местах, как параметры метода и коллекции или массивы. Когда это происходит, объявленный тип объекта перестает соответствовать своему типу во время выполнения. Базовые классы могут определять и реализовывать виртуальные методы, а производные классы – переопределять их, т.е. предоставлять свое собственное определение и реализацию. Во время выполнения, когда клиент вызывает метод, CLR выполняет поиск типа объекта во время выполнения и вызывает перезапись виртуального метода. Таким образом, в исходном коде можно вызывать метод на базовом классе и привести версию производного класса метода, который необходимо выполнить.

C# позволяет выполнять перегрузку операторов для их использования в собственных классах [8]. Это позволяет добиться естественного вида определяемого пользователем типа данных и использовать его в качестве основного типа данных. Функцию большинства встроенных операторов можно переопределить глобально или для отдельных классов. Перегруженные операторы реализуются в виде функции.

Делегат – это тип, представляющий ссылки на методы с конкретным списком параметров и возвращаемым типом [11]. При создании экземпляра делегата этот экземпляр можно связать с любым методом с совместимой сигнатурой и возвращаемым типом. Метод можно вызвать (активировать) с помощью экземпляра делегата. Делегаты используются для передачи методов в качестве аргументов к другим методам. Делегату можно назначить любой метод из любого доступного класса или структуры, соответствующей типу делегата. Этот метод должен быть статическим методом или методом экземпляра. Это позволяет изменять вызовы метода, а также включать новый код в существующие классы. Благодаря возможности ссылаться на метод как на параметр делегаты идеально подходят для определения методов обратного вызова. Например, ссылка на метод, сравнивающий два объекта, может быть передана в качестве аргумента алгоритму сортировки. Поскольку код сравнения находится в отдельной процедуре, алгоритм сортировки может быть написан более общим способом.

Атрибуты обеспечивают эффективный способ связывания метаданных или декларативной информации с кодом (сборками, типами, методами, свойствами и т. д.). Метаданные представляют собой сведения о типах, определенных в программе. Все сборки .NET содержат заданный набор метаданных, описывающих типы и члены типов, определенных в сборке. Один или несколько атрибутов могут применяться к сборкам, модулям или более мелким программным элементам, таким как классы и свойства. Атрибуты могут при-

нимать аргументы точно так же, как методы и свойства. После того как атрибут будет связан с программной сущностью, он может быть запрошен во время выполнения с помощью техники, называемой рефлексией.

Свойство – это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного поля [11]. Свойства можно использовать, как если бы они были членами общих данных, но фактически они представляют собой специальные методы, называемые методами доступа. Это позволяет легко получать доступ к данным и помогает повысить безопасность и гибкость методов. Свойства позволяют классу предоставлять общий способ получения и задания значений, скрывая при этом код реализации или проверки. Свойства могут быть доступны для чтения и записи, только для чтения или только для записи. Простые свойства, не требующие пользовательского кода метода доступа, можно реализовать как определения текста выражений или как автоматически реализуемые свойства.

События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций. Класс, отправляющий (или порождающий) событие, называется издателем, а классы, принимающие (или обрабатывающие) событие, называются подписчиками. События имеют следующие свойства:

- издатель определяет, когда возникает событие;
- подписчики определяют, какое действие выполняется в ответ на событие;
- у события может быть несколько подписчиков;
- подписчик может обрабатывать несколько событий от нескольких издателей;
- события, не имеющие подписчиков, никогда не возникают;
- если событие имеет несколько подписчиков, при возникновении события обработчики событий вызываются синхронно.

Аббревиатура LINQ обозначает целый набор технологий, создающих и использующих возможности интеграции запросов непосредственно в язык C#. Технологии LINQ превращают запросы в удобную языковую конструкцию, которая применяется аналогично классам, методам и событиям.

Для разработчика, который создает запросы, наиболее очевидной частью LINQ является интегрированное выражение запроса. Выражения запроса используют декларативный синтаксис запроса. С помощью синтаксиса запроса можно выполнять фильтрацию, упорядочение и группирование данных из источника данных, обходясь минимальным объемом программного кода. Одни и те же базовые выражения запроса позволяют одинаково легко получать и преобразовывать данные из баз данных SQL, наборов данных ADO.NET, XML-документов, XML-поток и коллекций .NET [9].

Особенности выражений запроса LINQ:

- выражение запроса можно использовать для получения и преобразования данных из любого источника данных, поддерживающего LINQ;

- выражение запроса очень легко освоить, поскольку в нем используется много знакомых конструкций языка C#;
- все переменные в выражениях запросов строго типизированы, хотя во многих случаях вам не нужно указывать тип явным образом, поскольку компилятор определит его автоматически;
- запрос не будет выполняться, пока вы не начнете обращаться к переменной запроса;
- во время компиляции выражения запроса преобразуются в вызовы метода стандартного оператора запроса.

Универсальные шаблоны в платформе .NET Framework представляют концепцию параметров типов, которые позволяют разрабатывать классы и методы, не придерживающиеся спецификации одного или нескольких типов до тех пор, пока класс или метод не будет объявлен клиентским кодом и пока не будет создан его экземпляр. Можно создавать собственные универсальные интерфейсы, классы, методы, события и делегаты. Доступ универсальных классов к методам можно ограничить определенными типами данных. Сведения о типах, используемых в универсальном типе данных, можно получить во время выполнения путем рефлексии. Шаблоны используются для достижения максимального уровня повторного использования кода, безопасности типа и производительности.

Итератор представляет собой метод, оператор или аксессор, возвращающий по очереди члены совокупности объектов от ее начала и до конца. Итератор можно использовать для прохода по коллекции, такой как список или массив.

Механизм отражения (или рефлексии) позволяет получать объекты, которые описывают сборки, модули и типы. Отражение можно использовать для динамического создания экземпляра типа, привязки типа к существующему объекту, а также получения типа из существующего объекта и вызова его методов или доступа к его полям и свойствам. Если в коде используются атрибуты, отражение обеспечивает доступ к ним.

3.1.3 Windows Presentation Foundation

Клиентское приложение, входящее в состав описываемого программного модуля, основано на технологии WPF.

Технология WPF (Windows Presentation Foundation) является частью экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов [12].

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. Графической технологией, лежащей в основе WPF, является DirectX. При использовании WPF значительная часть работы по отрисовке графики, как простейших элементов, так и сложных 3D-

моделей, ложится на графический процессор, что позволяет воспользоваться аппаратным ускорением графики.

Основные преимущества WPF включают:

- использование традиционных языков .NET-платформы – C# и VB.NET для создания логики приложения;
- возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML;
- независимость от разрешения экрана;
- создание трехмерных моделей;
- привязка данных;
- богатые возможности по созданию различных приложений: мультимедиа, двухмерная и трехмерная графика, богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимации, привязка данных, стили, шаблоны, темы;
- аппаратное ускорение графики.

XAML (eXtensible Application Markup Language) - язык разметки, используемый для инициализации объектов в технологиях на платформе .NET. В контексте WPF данный язык используется прежде всего для создания пользовательского интерфейса декларативным путем.

При работе с WPF широко используется шаблон MVVM или Model-View-ViewModel. MVVM позволяет отделить логику приложения от визуальной части (представления). Схема шаблона представлена на рисунке 3.2.

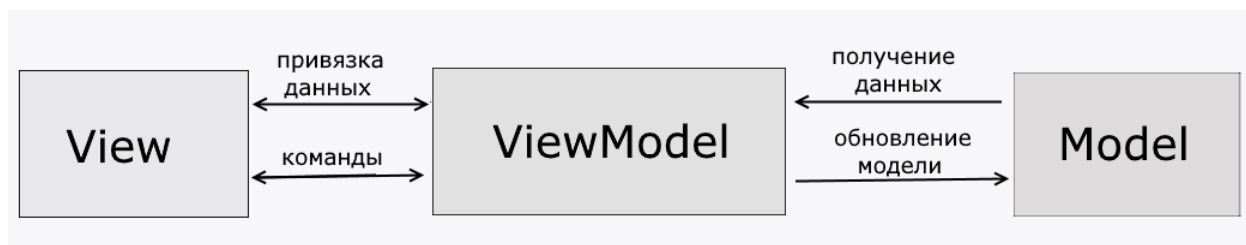


Рисунок 3.2 – Схема шаблона MVVM

Модель описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление – это XAML-код, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов.

ViewModel или модель представления связывает модель и представление через механизм привязки данных. ViewModel также содержит логику по

получению данных из модели, которые потом передаются в представление. И также ViewModel определяет логику по обновлению данных в модели.

Итогом применения шаблона MVVM является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать.

3.1.4 ASP.NET Web API

Серверное приложение, входящее в состав описываемого программного модуля, основано на технологии ASP.NET Web API.

ASP.NET Web API – технология для создания служб HTTP для широкого диапазона клиентов, включая браузеры и мобильные устройства. Веб-API ASP.NET идеально подходит для разработки RESTful приложений на платформе .NET Framework. Позволяет создавать сетевые API-интерфейсы, которые поддерживают большое количество различных типов содержимого, в том числе XML, JSON.

Имеются различные способы описания Web API – от автоматически формируемых HTML-страниц справки с описанием фрагментов до структурированных метаданных для интеграции API в OData.

3.1.5 Entity Framework

Работа с базами данных в описываемом программном модуле осуществляется с помощью технологии Entity Framework.

Платформа Entity Framework представляет собой набор технологий ADO.NET, обеспечивающих разработку приложений, связанных с обработкой данных [13]. Архитекторам и разработчикам приложений, ориентированных на обработку данных, приходится учитывать необходимость достижения двух совершенно различных целей. Они должны моделировать сущности, связи и логику решаемых бизнес-задач, а также работать с ядрами СУБД, используемыми для сохранения и получения данных. Данные могут распределяться по нескольким системам хранения данных, в каждой из которых применяются свои протоколы, но даже в приложениях, работающих с одной системой хранения данных, необходимо поддерживать баланс между требованиями системы хранения данных и требованиями написания эффективного и удобного для обслуживания кода приложения.

Entity Framework позволяет работать с данными в форме специфических для домена объектов и свойств, таких как клиенты и их адреса, без необходимости обращаться к базовым таблицам и столбцам базы данных, где хранятся эти данные. Entity Framework дает разработчикам возможность работать с данными на более высоком уровне абстракции, создавать и сопровождать приложения, ориентированные на данные, используя меньше кода, чем в традиционных приложениях.

Entity Framework поддерживает три подхода по проектированию базы данных:

- database first – на основе имеющейся базы данных;
- model first – на основе модели, построенной в графическом редакторе;
- code first – на основе классов .NET, представляющих сущности.

3.2 Описание структуры программного модуля

В основе программного модуля лежит модульная архитектура. При модульной архитектуре, программа разбивается на относительно независимые составные части - программные модули. При этом каждый модуль может разрабатываться, программироваться, транслироваться и тестироваться независимо от других. Внутреннее строение модуля для функционирования всей программы, как правило, значения не имеет. При модификации алгоритма, реализуемого модулем, структура программы меняется.

Разрабатываемый программный модуль разбит на подмодули, представленные в виде сборок .NET Framework. Сборки в .NET составляют основную единицу развертывания, управления версиями, повторного использования, областей действия активации и разрешений безопасности. Сборка представляет собой коллекцию типов и ресурсов, собранных для совместной работы и образующих логическую функциональную единицу. Сборка предоставляет общезыкоковой исполняющей среде сведения, необходимые для распознавания реализаций типов.

Структурная схема программного модуля на уровне сборок представлена на рисунке 3.3.

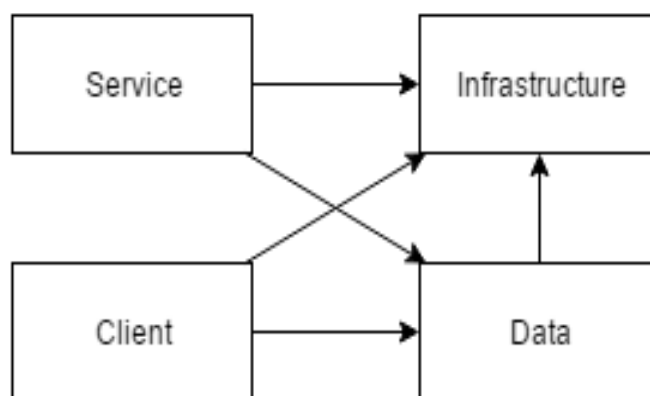


Рисунок 3.3 – Структурная схема программного модуля на уровне сборок

Схема содержит следующие компоненты:

- Service – серверная часть приложения. Представляет собой RESTful сервис на основе ASP.NET Web API, обрабатывает запросы от клиентов;

- Client – клиентское настольное приложение на основе WPF, предоставляет пользовательский интерфейс для данных, общается с сервером по протоколу HTTP;
- Data – содержит компоненты работы с данными (БД) на основе EntityFramework, классы для представления сущностей модели данных (пользователь, должность и т.п., соответствуют таблицам БД);
- Infrastructure – компоненты, используемые в обеих частях приложения, а именно модели данных, передающихся между клиентом и сервером.

3.3 Описание модуля Client

Диаграмма классов модуля Client представлена на рисунке 3.4.

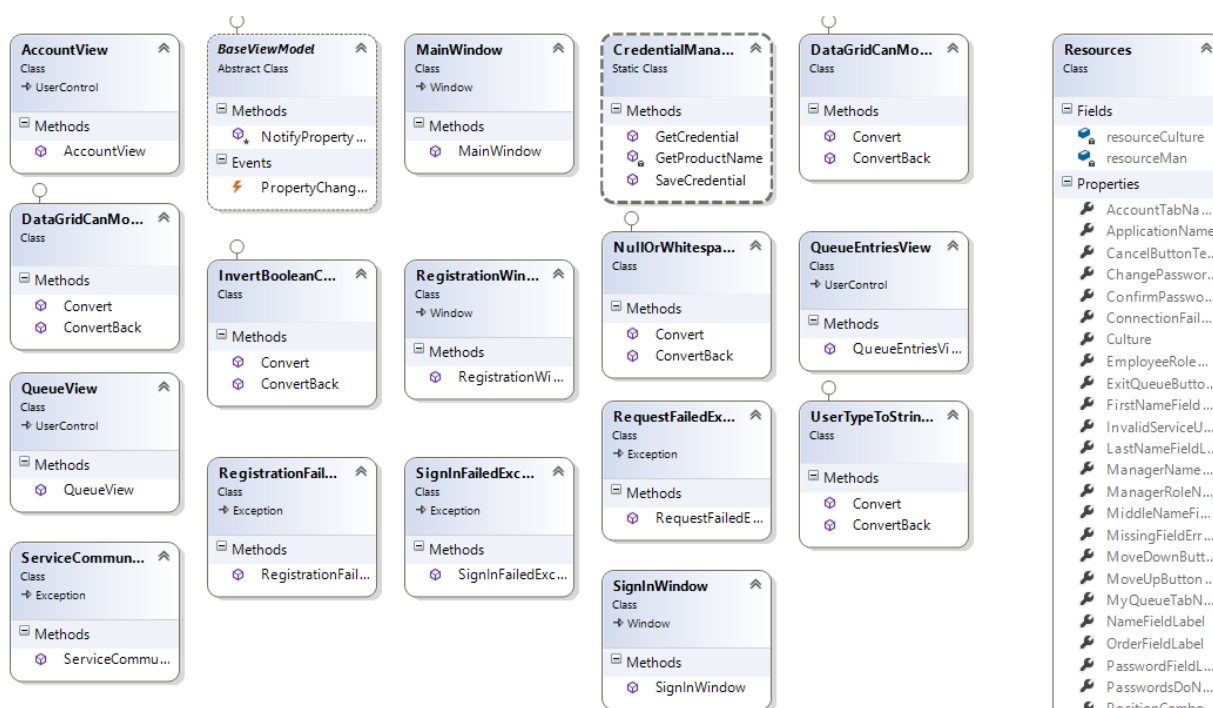


Рисунок 3.4 – Диаграмма классов модуля Client

Данный модуль содержит следующие классы и интерфейсы:

- класс AccountView – аккаунты клиентов;
- класс QueueView – просмотр очереди;
- класс ServiceCommunicationException – исключения, относящиеся к взаимодействию с сервером;
- класс RegistrationFailedException – исключения, происходящие в процессе регистрации;
- класс MainWindow – окно приложения;
- класс RegistrationWindow – окно регистрации;
- класс SignInFailedException – ошибки входа в кабинет пользователя;

- класс `CredentialManager` – класс, содержащий логику обращения с диспетчером учетных данных `Windows`;
- класс `RequestFailedException` – исключения, содержащие ошибки, полученные от сервера;
- класс `SignInWindow` – окно входа в систему;
- класс `QueueEntriesView` – представление вхождений пользователя в очереди;
- класс `Resources` – строковые ресурсы, подлежащие локализации;
- классы `DataGridCanMoveUpConverter`, `DataGridCanMoveDownConverter` – конвертер состояния таблицы сотрудников в возможность изменения порядка;
- класс `BaseViewModel` – базовый класс моделей представлений, содержащий общую логику для них логику;
- класс `InvertBooleanConverter` – конвертер булевых значений в обратные;
- класс `NullOrWhitespaceToVisibilityConverter` – конвертер нулевых значений в негативные значения видимости компонентов пользовательского интерфейса.

3.4 Описание модуля Data

Диаграмма классов модуля `Data` представлена на рисунке 3.5.

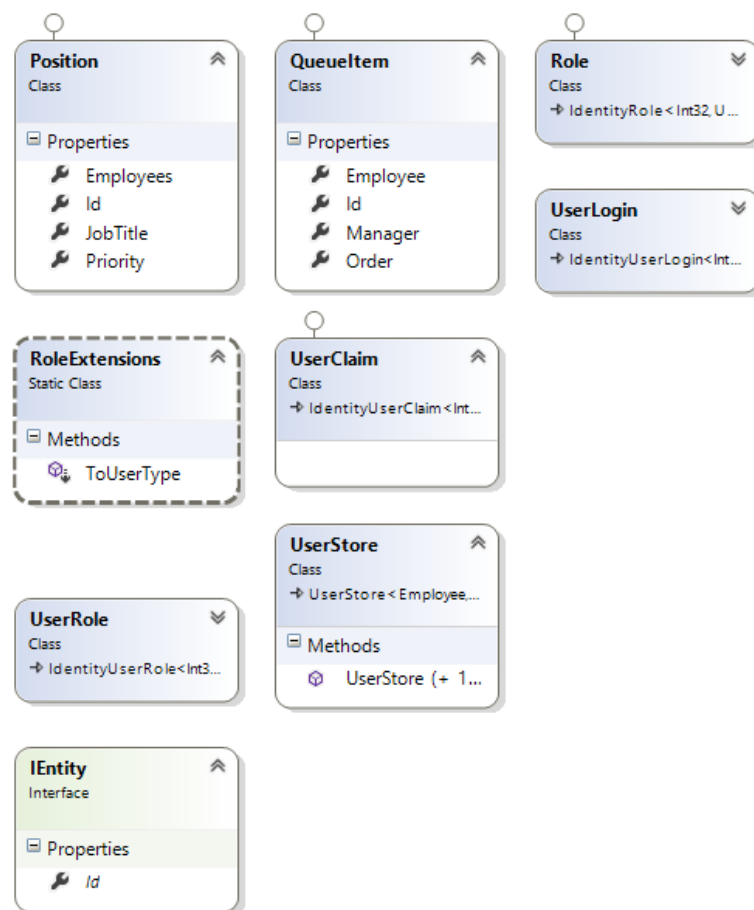


Рисунок 3.5 – Диаграмма классов Data

Данный модуль содержит классы и интерфейсы компонентов работы с данными (БД) на основе Entity Framework, классы для представления сущностей модели данных (пользователь, должность и т.п.). Подробно расписывать их не целесообразно.

3.5 Описание модуля Services

Диаграмма классов модуля Services представлена на рисунке 3.6.

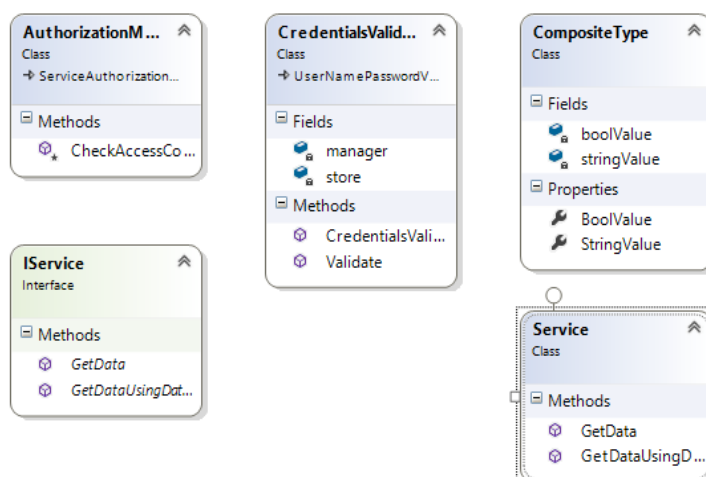


Рисунок 3.6 – Диаграмма классов Services

Серверная часть приложения, содержащаяся в модуле Services, представляет собой RESTful сервис на основе ASP.NET Web API. Данный модуль содержит следующие классы и интерфейсы:

- класс AuthorizationManager – авторизация менеджеров системы;
- класс CredentialsValidator – валидатор данных для аутентификации и авторизации пользователей;
- интерфейс IService – интерфейс сервиса для взаимодействия с ним клиентов;
- класс Service – реализация сервиса.

3.6 Описание модуля Infrastructure

Модуль Infrastructure содержит классы и интерфейсы, общие для всех частей приложения и используемые для передачи данных между частями. Диаграмма классов модуля Infrastructure представлена на рисунке 3.7.

Модуль Infrastructure содержит следующие компоненты:

- класс ChangePasswordBindingMode – данные для смены пароля пользователем;
- класс RoleNames – роли пользователей системы;
- класс RegisterBindingMode – данные для регистрации пользователей;
- класс TokenGrantModel – данные для входа в систему с помощью логина и пароля.

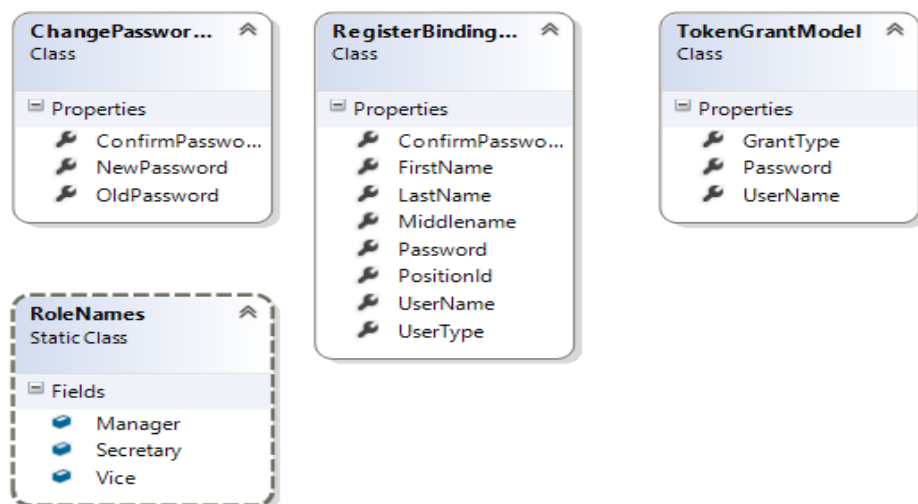


Рисунок 3.7 – Диаграмма классов модуля Infrastructure

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ

4.1 Модульное тестирование

При тестировании разработанного программного модуля, основное внимание уделялось модульному тестированию.

Модульное тестирование позволяет проверить на корректность отдельные модули исходного кода программы. Модульное тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно модульное тестирование проводится, вызывая код, который необходимо проверить, и при поддержке сред разработки, таких как фреймворки для модульного тестирования или инструменты для отладки.

Цель модульного тестирования – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Такое тестирование позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Это поощряет разработчиков к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Модульное тестирование основывается на проверке работоспособности модуля в отдельности от других, что побуждает избавляться от сильного связывания и писать более структурированный и модульный код.

Для удовлетворения зависимостей тестируемого объекта, а также для имитации и проверки его взаимодействия с другими объектами, используются заглушки, также называемые mock-объектами. Mock-объект представляет собой конкретную фиктивную реализацию интерфейса, предназначенную исключительно для тестирования взаимодействия и относительно которого высказывается утверждение.

Для осуществления модульного тестирования использовалась технология Microsoft Visual Studio Unit Testing Framework. Эта технология используется для создания модульных тестов, их запуска и создания отчетов о результатах таких тестов. Она позволяет писать модульные тесты в виде .NET классов и методов, помеченных специальными атрибутами. Также, Visual Studio Unit Testing Framework предоставляет тесную интеграцию со средой разработки Visual Studio, что облегчает написание и запуск тестов, а также позволяет автоматизировать запуск тестов.

В ходе разработки компонентов программного модуля, перед реализацией поведения компонента для него создавался набор модульных тестов в соответствии с ожидаемым поведением компонента. Затем, реализация произво-

дидась с учетом ожидаемых результатов модульных тестов. Компонент считался реализованным, если 100% тестирующих его модульных тестов проходили успешно. Благодаря такому подходу, компоненты работали как ожидалось, и после сборки компонентов не возникло ошибок, связанных с реализацией компонентов. Также, сформированный набор тестов служит гарантией работы имеющегося функционала в случае внесения изменений.

4.2 Функциональное тестирование

После сборки компонентов, программный модуль был подвергнут ручному функциональному тестированию в соответствии с описанными выше требованиями.

В результате тестирования были выявлены дефекты, которые в последствии были устранены. Функциональное тестирование подтвердило соответствие разработанного программного модуля требованиям.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для входа в приложение пользователю необходимо прохождение авторизации. Для этого необходимо ввести логин и пароль, как показано на рисунке 5.1.

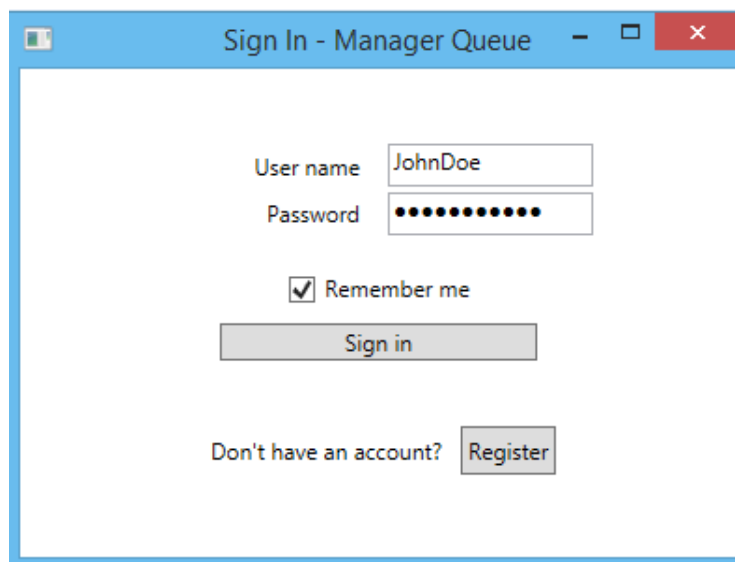


Рисунок 5.1 – Авторизация в системе

Если пользователь работает в системе первый раз, ему необходимо пройти процедуру регистрации. Для этого необходимо заполнить все поля формы и нажать кнопку “Register”, как показано на рисунке 5.2.

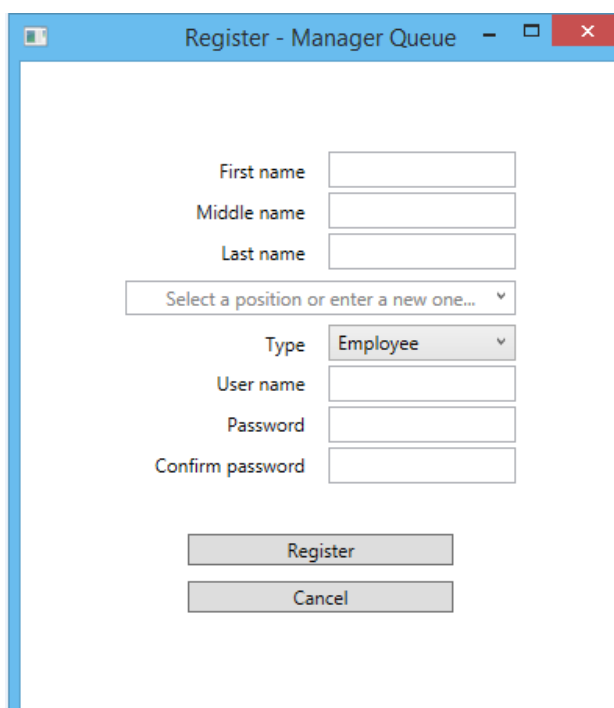


Рисунок 5.2 – Регистрация в системе

Управления собственной очередью осуществляется как показано на рисунке 5.3. По приоритету руководителей, очередь можно поднимать или опускать вниз/вверх списка.

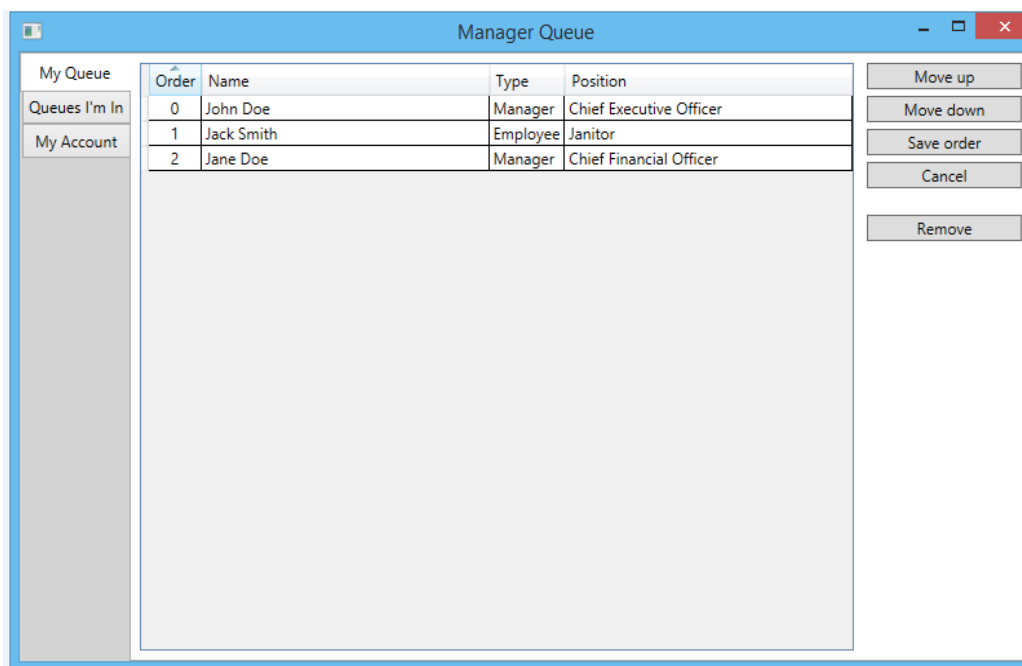


Рисунок 5.3 – Управление очередью

Вкладка “My Account” осуществляет управление учетной записью, как показано на рисунке 5.4. Можно изменить и редактировать данные, указанные при регистрации.

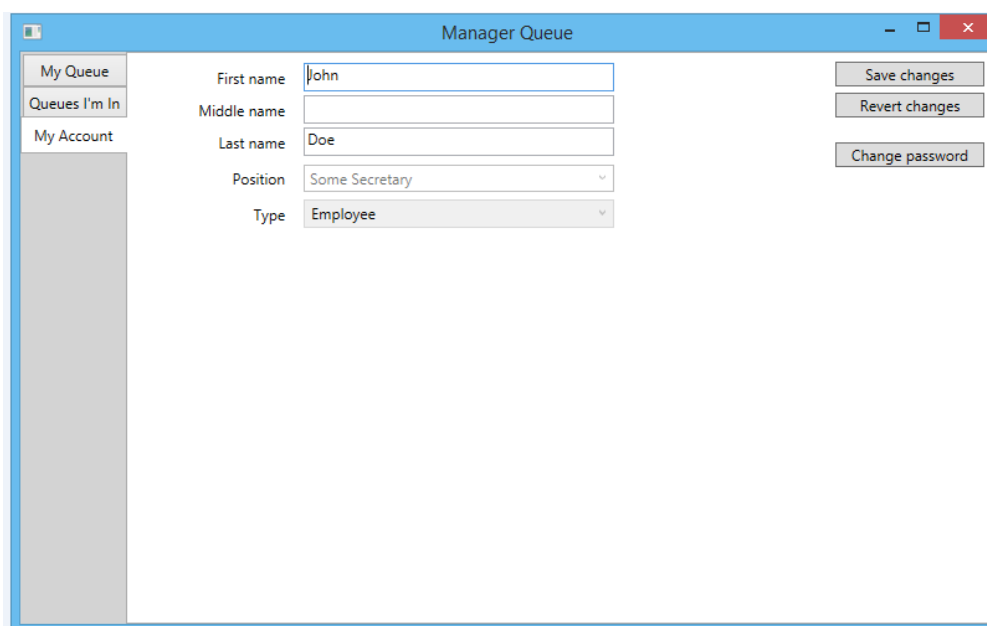


Рисунок 5.4 – Управление учетной записью

Вкладка “Queues I’m In”, показанная на рисунке 5.5, осуществляет просмотр очередей, в которых стоит пользователь, выход из очередей и постановку в новую очередь.

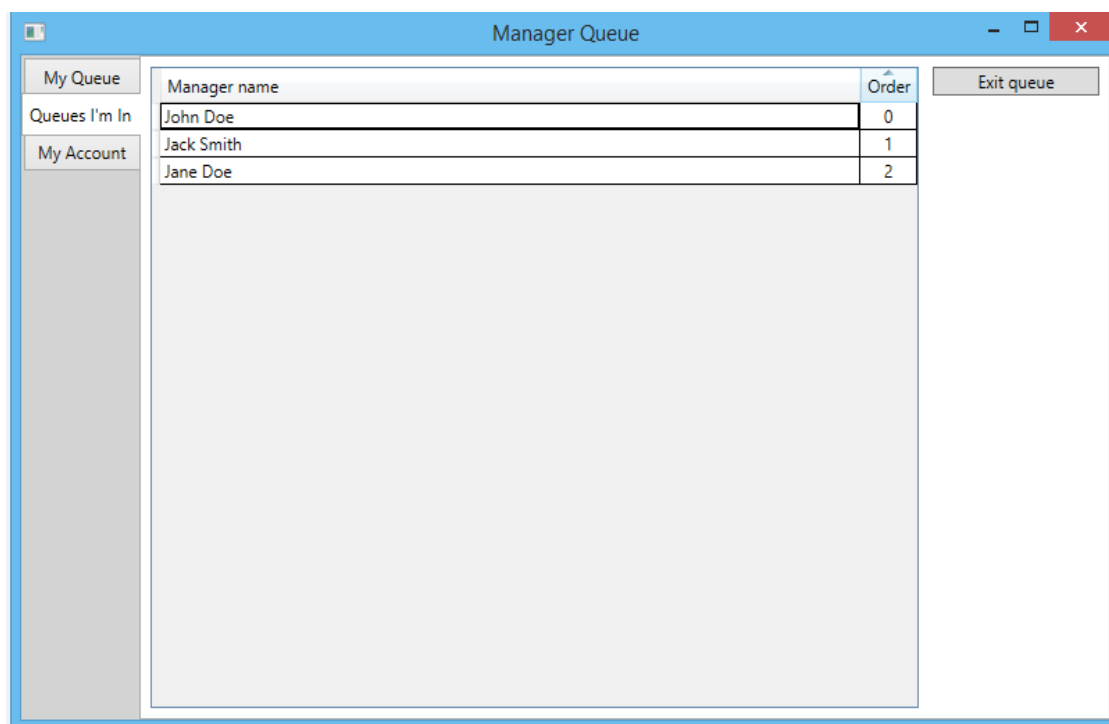


Рисунок 5.5 – Просмотр очередей

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

6.1 Характеристика программного модуля и исходные данные

В ходе выполнения дипломного проекта была проведена следующая работа:

- раскрыто понятие и назначение формирования электронной очереди;
- проанализированы существующие системы формирования электронной очереди;
- произведена оценка существующей системы формирования электронной очереди;
- выявлены резервы повышения эффективности функционирования модуля формирования электронной очереди;
- разработать предложения по повышению эффективности работы модуля формирования электронной очереди.

Программный модуль формирования электронной очереди обеспечивает выполнение следующих основных функций:

- запись в очередь на прием к начальникам и заместителям начальников организации;
- управление очередью секретарем, заместителями и начальниками, а именно удаление, изменение порядка;
- выставление приоритета очереди, а именно заместители руководителя имеют право попадания на прием перед рядовыми сотрудниками.

Потребность в разработке программного модуля формирования электронной очереди, обусловлена жесткой конкуренцией на рынке информационных технологий и необходимостью обеспечить более высокую производительность труда, большую надежность и достоверность информации, лучшую ее сохранность.

За счет доступного и простого интерфейса обеспечивается снижение вероятности возникновения каких-либо ошибок и практически полное отсутствие необходимости обучения пользования приложением. Следовательно, данное приложение может иметь широкий спектр применения.

Таким образом, применение разрабатываемого программного модуля формирования электронной очереди, является целесообразным и необходимым современных в условиях.

Разрабатываемое ПС относится к 3-й группе сложности. Так как данное ПС является развитием определенного параметрического ряда ПО и не является уникальным в своем роде, то данное ПС по степени новизны можно отнести к группе «В» с коэффициентом новизны 0,7. Данное ПС обеспечивает интерактивный доступ к электронной очереди, поэтому дополнительный коэффициент сложности будет иметь значение 0,06. Степень охвата реализуемых функций разрабатываемого ПС стандартными модулями типовых программ составляет около 20%, следовательно, $K_T=0,8$.

Таблица 6.1 – Исходные данные

Наименование показателей	Обозначения	Единицы измерения	Количество
Коэффициент новизны	K_n	единиц	0,7
Группа сложности	-	единиц	3
Дополнительный коэффициент сложности	K_c	единиц	0,06
Поправочный коэффициент, учитывающий использование типовых программ	K_t	единиц	0,8
Установленная плановая продолжительность разработки	T_p	лет	0,33
Продолжительность рабочего дня	$T_{\text{ч}}$	ч	8
Тарифная ставка 1-го разряда	T_{m1}	руб.	31
Коэффициент премирования	K_p	единиц	1,2
Норматив дополнительной заработной платы исполнителей	H_d	%	20
Отчисления в фонд социальной защиты населения	$З_{сз}$	%	34
Отчисления в Белгосстрах	$H_{не}$	%	0,6
Налог на прибыль	H_p	%	18
Норматив расходов на сопровождение и адаптацию	H_c	%	20
Налог на добавленную стоимость	$H_{дс}$	%	20
Уровень рентабельности ПС	$У_{рпi}$	%	15

6.2 Расчет затрат и отпускной цены программного средства

6.2.1. Уточнённый объем ПО (строки исходного кода, LOC)

$$V_y = \sum_{i=1}^n V_{yi}, \quad (6.1)$$

где V_y – уточненный объем ПО;

V_{yi} – уточненный объем отдельной функции ПО (LOC);

n – общее число функций.

Таблица 6.2 – Перечень и объем функций программного модуля

№ функции	Наименование (содержание) функции	Объем функции (LOC)	
		V _i	V _{yi}
101	Организация ввода информации	150	150
102	Контроль, предварительная обработка и ввод информации	450	450
109	Организация ввода/вывода информации в интерактивном режиме	320	320
111	Управление вводом/выводом	2400	2100
301	Формирование последовательного файла	290	290
305	Обработка файлов	720	1250
306	Обработка файлов в диалоговом режиме	3050	2500
309	Формирование файла	1020	1000
501	Монитор ПО (управление работой компонентов)	740	740
506	Обработка ошибочных и сбойных ситуаций	410	300
507	Обеспечение интерфейса между компонентами	970	900
Итого		10523	10000

Среда разработки ПО – Visual Studio 2015 (C#), программное обеспечение функционального назначения.

V_y = 10000 LOC.

6.2.2. Трудоемкость разработки ПО

Общая трудоемкость небольшого проекта высчитывается по формуле 6.2:

$$T_o = T_n \cdot K_c \cdot K_T \cdot K_H = 228 \cdot 1,06 \cdot 0,8 \cdot 0,7 = 136 \text{ чел./дн.} \quad (6.2)$$

где T_o – общая трудоемкость;

T_n – нормативная трудоемкость ПО, равная 228 чел./дн.;

K_c – дополнительный коэффициент сложности;

K_T – поправочный коэффициент, учитывающий использование типовых программ;

K_n – коэффициент новизны.

4.2.3. Численность исполнителей проекта

Численность исполнителей проекта:

$$Ч_p = \frac{T_o}{T_p \cdot \Phi_{эф}}, \quad (6.3)$$

где $Ч_p$ – численность исполнителей проекта;

T_p – срок разработки проекта (лет);

$\Phi_{эф}$ – эффективный фонд времени работы одного работника.

Эффективный фонд времени работы одного работника:

$$\Phi_{эф} = D_r - D_n - D_v - D_o = 236 \text{ дн.}, \quad (6.4)$$

где $\Phi_{эф}$ – эффективный фонд времени работы одного работника;

D_r – количество дней в году;

D_n – количество праздничных дней в году;

D_v – количество выходных дней в году;

D_o – количество дней отпуска.

$$Ч_p = \frac{136}{0,33 \cdot 236} \approx 2 \text{ чел.}$$

6.3 Расчет сметы затрат и цены заказного ПО

6.3.1 Основная заработная плата ($З_{oi}$) исполнителей на конкретное ПО рассчитывается по формуле:

$$З_{oi} = \sum_i^n З_{ci} \cdot \Phi_{pi} \cdot K, \quad (6.5)$$

где n – количество исполнителей, занятых разработкой конкретного ПО;

$З_{ci}$ –среднедневная заработная плата i -го исполнителя (д.е.);

Φ_{pi} – плановый фонд рабочего времени i -го исполнителя (дн.);

K – коэффициент премирования.

Расчет основной заработной платы представлен в табл. 6.3.

Таблица 6.3 – расчет основной заработной платы

Категория исполнителя	Средне-дневная заработная плата, руб.	Плановый фонд рабочего времени, дн.	Коэф-коэффициент премирования	Основная заработная плата, руб.
Руководитель проекта	22,5	114	1,2	3078
Инженер-программист	17,5	114	1,2	2394
Итого с премией (20%), Z_o	-	-	-	5472

6.3.2. Дополнительная заработная плата исполнителей проекта определяется по формуле:

$$Z_d = \frac{Z_o \cdot H_d}{100}, \quad (6.6)$$

где Z_d – дополнительная заработная плата;

H_d – норматив дополнительной заработной платы (20%).

Дополнительная заработная плата составит:

$$Z_d = 5472 \cdot 20 / 100 = 1094,4 \text{ руб.}$$

6.3.3. Отчисления в фонд социальной защиты населения и на обязательное страхование ($Z_{сз}$) определяются в соответствии с действующими законодательными актами по формуле 1.7.

$$Z_{сз} = \frac{(Z_o + Z_d) \cdot H_{сз}}{100}, \quad (6.7)$$

где $H_{сз}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 + 0,6%).

Рассчитаем величину отчислений в фонд социальной защиты населения и на обязательное страхование:

$$Z_{сз} = (5472 + 1094,4) \cdot 34,6 / 100 = 2271,98 \text{ руб.}$$

6.3.4. Расходы по статье «Машинное время» (P_M) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле:

$$P_m = C_m \cdot \frac{V_y}{100} T_{\text{ч}} \cdot H_{\text{мв}}, \quad (6.8)$$

где P_m – расходы по статье «Машинное время»;

C_m – цена одного машино-часа;

V_y – объём кода;

$H_{\text{мв}}$ – норматив времени на отладку 100 строк кода (12 ч.).

Стоимость машино-часа на предприятии составляет 1,4 руб. Разработка проекта займет 228 дней. Определим затраты по статье “Машинное время” для одной машины:

$$P_m = 1,4 \cdot 10000 / 100 \cdot 12 = 1680 \text{ руб.}$$

6.3.5. Затраты по статье «Накладные расходы» (P_n), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды (P_n), определяются по формуле

$$P_n = \frac{3_o \cdot H_{\text{рн}}}{100}, \quad (6.9)$$

где $H_{\text{рн}}$ – норматив накладных расходов (50%).

Накладные расходы составят:

$$P_n = 5472 \cdot 0.5 = 2736 \text{ руб.}$$

Общая сумма расходов по всем статьям сметы (C_n) на ПО рассчитывается по формуле:

$$C_n = 3_o + 3_d + 3_{\text{сз}} + P_m + P_n, \quad (6.10)$$

Рассчитаем сумму расходов по всем статьям сметы:

$$C_n = 5472 + 1094,4 + 2271,98 + 1680 + 2736 = 13254,38 \text{ руб.}$$

Прибыль по создаваемому ПС рассчитывается по формуле:

$$P_o = \frac{C_n \cdot y_{\text{рп}}}{100}, \quad (6.11)$$

где P_o – прибыль от реализации ПС заказчику (руб.);
 U_{pp} – уровень рентабельности ПС (15%);
 C_n – себестоимость ПС (руб.).

Рассчитаем прибыль по создаваемому ПС:

$$P_o = 13254,38 \cdot 15 / 100 = 1988,16 \text{ руб.}$$

Прогнозируемая цена ПС (C_n) без налогов вычисляется по формуле:

$$C_n = C_n + P_o, \quad (6.12)$$

Рассчитаем прогнозируемую цену ПС без налогов:

$$C_n = 13254,38 + 1988,16 = 15242,54 \text{ руб.}$$

Налог на добавленную стоимость (НДС_i):

$$\text{НДС} = \frac{C_n \cdot H_{дс}}{100}, \quad (6.13)$$

где $H_{дс}$ – норматив НДС (20%).

Рассчитаем НДС:

$$\text{НДС} = 15242,54 \cdot 20 / 100 = 3048,51 \text{ руб.}$$

Прогнозируемая отпускная цена (C_o):

$$C_o = C_n + \text{НДС}, \quad (6.14)$$

Рассчитаем прогнозируемую отпускную цену

$$C_o = 15242,54 + 3048,51 = 18291,05 \text{ руб.}$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение ПС (P_c), которые определяются по нормативу расходов на сопровождение и адаптацию (20%):

$$P_c = \frac{H_c \cdot C_p}{100}, \quad (6.15)$$

где H_c – норматив расходов на сопровождение и адаптацию ПС;

C_p – смета расходов в целом по организации без расходов на сопровождение и адаптацию (руб.).

Рассчитаем затраты на сопровождение ПС:

$$P_c = 13254,38 \cdot 20 / 100 = 2650,88 \text{ руб.}$$

6.4 Оценка экономической эффективности применения программного средства у пользователя

Таблица 6.4 – исходные данные для расчета экономии ресурсов в связи с применением нового программного средства

Наименование показателей	Обозначения	Единицы измерения	Значение показателя		Наименование источника информации
			в базовом варианте	в новом варианте	
1. Капитальные вложения, включая затраты пользователя на приобретение	$K_{пр}$	руб.		18291,05	Договор заказчика с разработчиком
2. Затраты на сопровождение ПО	K_c	руб.		2650,88	Договор заказчика с разработчиком
3. Время простоя сервиса, обусловленное ПО, в день	$П1, П2$	мин	50	10	Расчетные данные пользователя и паспорт ПО
4. Стоимость одного часа простоя	$Сп$	руб.	31	31	Расчетные данные пользователя и паспорт ПО
5. Среднемесячная ЗП одного программиста	$Зсм$	руб.	430,0	430,0	Расчетные данные пользователя

Продолжение таблицы 6.4

6. Коэффициент начислений на зарплату	Кнз		1,5	1,5	Рассчитывается по данным пользователя
7. Среднемесячное количество рабочих дней	Др	день		21.5	Принято для расчета
8. Количество типовых задач, решаемых за год	Зт1 ,Зт2	задача	1806	1806	План пользователя
9. Объем выполняемых работ	А1, А2	задача	1806	1806	План пользователя
10. Средняя трудоемкость работ на задачу	Тс1,Тс2	Человеко-часов	2,0	1,0	Рассчитывается по данным пользователя
11. Количество часов работы в день	Тч	ч	8	8	Принято для расчета
12. Ставка налога на прибыль	Нп	%		18	

Экономия затрат на заработную плату в расчете на 1 задачу ($C_{зе}$):

$$C_{зе} = \frac{З_{см} \cdot (Т_{с1} - Т_{с2})}{Др \cdot Тч}, \quad (6.16)$$

где $З_{см}$ – среднемесячная заработная плата одного программиста (руб.);

$Т_{с1}$, $Т_{с2}$ – снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);

$Тч$ – количество часов работы в день (ч);

$Др$ – среднемесячное количество рабочих дней.

Рассчитаем экономию затрат на заработную плату в расчете на 1 задачу:

$$C_{зе} = 430 \cdot (2-1) / (8 \cdot 21.5) = 2,5 \text{ руб}$$

Экономия заработной платы при использовании нового ПО (руб.):

$$C_3 = C_{зе} \cdot A_2, \quad (6.17)$$

где C_3 – экономия заработной платы;

A_2 – количество типовых задач, решаемых за год (задач).

Рассчитаем экономию заработной платы при использовании нового ПО:

$$C_3 = 2,5 \cdot 1806 = 4515 \text{ руб.}$$

Экономия с учетом начисления на зарплату (C_H):

$$C_H = C_3 \cdot K_{HЗ}, \quad (6.18)$$

Рассчитаем экономию с учетом начисления на зарплату:

$$C_H = 4515 \cdot 1,5 = 6772,5 \text{ руб.}$$

Экономия за счет сокращения простоев сервиса (C_c) рассчитывается по формуле:

$$C_c = \frac{D_{рг} \cdot C_{п} \cdot (П_1 - П_2)}{60}, \quad (6.19)$$

где $D_{рг}$ – плановый фонд работы сервиса (дней).

$$C_c = 365 \cdot 31 \cdot (50-10) / 60 = 7543,33 \text{ руб.}$$

Общая готовая экономия текущих затрат, связанных с использованием нового ПО (C_o), рассчитывается по формуле:

$$C_o = C_H + C_c, \quad (6.20)$$

$$C_o = 6772,5 + 7543,33 = 14315,83 \text{ руб.}$$

Расчет экономического эффекта.

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, т.е. практически получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении ($\Delta\Pч$), которая определяется по формуле:

$$\Delta\Pч = C_o - \frac{C_o \cdot H_{\pi}}{100}, \quad (6.21)$$

где H_{π} – ставка налога на прибыль (18%).

$$\Delta\Pч = 14315,83 - (14315,83 \cdot 18) / 100 = 11738,98 \text{ руб.}$$

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капитальных вложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования α . В данном примере используются коэффициенты: 2017 г. – 1, 2018-й – 0,8696, 2019-й – 0,7561, 2020 г. – 0,6575. Все рассчитанные данные экономического эффекта сводятся в таблицу 1.5.

Таблица 6.5 – расчет экономического эффекта от использования нового программного средства

Показатели	Единицы измерения	Годы			
		2017	2018	2019	2020
<i>Результаты</i>					
Прирост прибыли за счет экономии затрат ($\Pi_ч$)	руб.		11738,98	11738,98	11738,98
То же с учетом фактора времени	руб.		10208,22	8875,85	7718,38
<i>Затраты</i>					
Приобретение ПО (Кпр)	руб.	18291,05			
Сопровождение (Кс)	руб.	2650,88			
Всего затрат	руб.	20941,93			
То же с учетом фактора времени	руб.	20941,93			

Продолжение таблицы 6.5

<i>Экономический эффект</i>					
Превышение результата над затратами	руб.	-20941,93	10208,22	8875,85	7718,38
То же с нарастающим итогом	руб.	-20941,93	-10733,7	-1857,86	5854,52
Коэффициент приведения	единицы	1	0,8696	0,7561	0,6575

Реализация проекта ПО позволит заказчику снизить трудоемкость решения задач, сократить простои сервиса и минимизировать возникновение ошибок из-за человеческого фактора.

Все затраты заказчика окупятся на 3-м году эксплуатации ПО.

Проект представляется эффективным и полезным для заказчика.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была проведена следующая работа:

- раскрыто понятие и назначение формирования электронной очереди;
- проанализированы существующие системы формирования электронной очереди;
- произведена оценка существующей системы формирования электронной очереди;
- выявлены резервы повышения эффективности функционирования модуля формирования электронной очереди;
- разработаны предложения по повышению эффективности работы модуля формирования электронной очереди.

Потребность в разработке программного модуля формирования электронной очереди, обусловлена жесткой конкуренцией на рынке информационных технологий и необходимостью обеспечить более высокую производительность труда, большую надежность и достоверность информации, лучшую ее сохранность.

Важным этапом является проектирование такого интерфейса системы, чтобы пользователю было легко пользоваться приложением.

Большое внимание следует уделить разработке базы данных, на которой основывается проект. Были продуманы все необходимые сущности, атрибуты и связи. После всего этого можно и приступить к разработке самого проекта.

Разработанный модуль формирования электронной очереди обеспечивает выполнение следующих основных функций:

- запись в очередь на прием к начальникам и заместителям начальников организации;
- управление очередью секретарем, заместителями и начальниками, а именно удаление, изменение порядка;
- выставление приоритета очереди, а именно заместители руководителя имеют право попадания на прием перед рядовыми сотрудниками.

Также произведена оценка экономической эффективности разработки модуля формирования электронной очереди.

Важным преимуществом данного приложения является возможность не только осуществлять работу с базой данных, но и анализировать информацию, представить в структурированном и читабельном виде.

За счет доступного и простого интерфейса обеспечивается снижение вероятности возникновения каких-либо ошибок и практически полное отсутствие необходимости обучения пользования приложением. Следовательно, данное приложение может иметь широкий спектр применения.

В дальнейшем может быть расширен набор функций и задач, решаемый использованием данной системы. Например, следует рассматривать возможность усовершенствования: улучшение интерфейса, добавления новых функций, которые сможет выполнять приложение.

Согласно проведенному технико-экономическому обоснованию разработанная система является экономически эффективной. По расчетам выявлено, что все дополнительные капитальные затраты на освоение, сопровождение и адаптацию нового ПО окупятся в течение трех лет. Положительный экономический эффект достигнут за счет экономии затрат на заработную плату.

В рамках дальнейшего развития проекта можно определить автоматизацию других подразделений компании в рамках единой информационной системы предприятия.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Бугорский, В.Н. Информационные системы в экономике: основы информационного бизнеса. Учебное пособие / В.Н. Бугорский, В.И. Фомин – СПб.: СПбГИЭА, 1999.
- [2] Электронная очередь – Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Электронная_очередь.
- [3] AKIS MICRO | AKIS Technologies [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.akistechnologies.com/solutions/organize-people/queue-management-systems/akis-micro/>.
- [4] Электронная очередь AKIS Micro [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.studioer.ru/products/elektron-naya-ochered/akis-micro/>.
- [5] Электронная система управления очередью | Meta-Q [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.meta-q.ru/products_meta-q/electronic_queue/.
- [6] Описание системы управления очередью Meta-Q V 2.28 [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.metatgroup.ru/projects/Meta-Q_v_2.28.pdf.
- [7] Система управления очередью IBA QueueMaster [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://ibarus.ru/products/queue/>.
- [8] C# 5.0 и платформа .NET 4.5 для профессионалов / К. Нейгел [и др.] – М.: «Диалектика», 2013. – 1440 с.
- [9] Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание / Э. Троелсен – М. «Вильямс», 2013. – 1312 с.
- [10] Общие сведения о платформе .NET Framework [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://msdn.microsoft.com/ru-ru/library/zw4w595w\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/zw4w595w(v=vs.110).aspx).
- [11] Руководство по программированию на C# [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/>.
- [12] Мак-Доналд, М. Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов, 4-е издание / М. Мак-Доналд – М. «Вильямс», 2013. – 1024 с.
- [13] Lerman, J. Programming Entity Framework – 2nd Edition / J. Lerman – O'Reilly, 2010. – 920 p.

ПРИЛОЖЕНИЕ А (обязательное)

Исходный код программы

```
[Authorize]
public class QueueController : ApiController
{
    private ApplicationDbContext dbContext;
    private ApplicationUserManager userManager;

    public QueueController()
    {
    }

    public QueueController(ApplicationDbContext dbContext, ApplicationUserManager userManager)
    {
        DbContext = dbContext;
        UserManager = userManager;
    }

    public ApplicationDbContext DbContext
    {
        get
        {
            return dbContext ?? Request.GetOwinContext().Get<ApplicationDbContext>();
        }
        private set
        {
            dbContext = value;
        }
    }

    public ApplicationUserManager UserManager
    {
        get
        {
            return userManager ?? Request.GetOwinContext().GetUserManager<ApplicationUserManager>();
        }
        private set
        {
        }
    }
}
```



```

        userManager = value;
    }
}

// GET: api/Queue
[Authorize(Roles = RoleNames.Manager + ", " + RoleNames.Vice)]
public async Task<IEnumerable<QueueItem>> Get()
{
    var userId = User.Identity.GetUserId<int>();
    return await dbContext.Queue.Where(queueItem => queueItem.ManagerId
== userId).ToArrayAsync();
}

// GET: api/Queue/5
[Authorize(Roles = RoleNames.Manager + ", " + RoleNames.Vice + ", " +
RoleNames.Secretary + ", " + RoleNames.Administrator)]
[ResponseType(typeof(IEnumerable<QueueItem>))]
public async Task<IHttpActionResult> Get(int id)
{
    var userId = User.Identity.GetUserId<int>();
    if (!await UserHasAccessToQueue(userId, id))
        return Unauthorized();

    return Ok(await dbContext.Queue.Where(queueItem => queueItem.Man-
agerId == id).ToArrayAsync());
}

// PUT: api/Queue/5
public async Task<IHttpActionResult> Put(int queueId, [FromBody]IEnu-
merable<QueueItem> queueItems)
{
    var queueIds = queueItems.Select(queueItem => queueItem.Man-
agerId).Distinct().ToArray();
    if (queueIds.Length != 1 || queueIds[0] != queueId)
        return BadRequest();

    if (!Enumerable.SequenceEqual(queueItems.OrderBy(queueItem =>
queueItem.Order).Select(queueItem => queueItem.Order), Enumerable.Range(0,
queueItems.Count() - 1)))
        return BadRequest();

    var userId = User.Identity.GetUserId<int>();
    if (!await UserHasAccessToQueue(userId, queueId))
        return Unauthorized();

```

```

        var originalQueue = await dbContext.Queue.Where(queueItem =>
queueItem.ManagerId == queueId).OrderBy(queueItem => queueItem.EmployeeId).ToArrayAsync();
        if (originalQueue.Length != queueItems.Count())
            return BadRequest();

        queueItems = queueItems.OrderBy(queueItem => queueItem.Employee);
        if (!Enumerable.SequenceEqual(queueItems.Select(queueItem =>
queueItem.EmployeeId), originalQueue.Select(queueItem => queueItem.EmployeeId)))
            return BadRequest();

        originalQueue.Zip(queueItems.Select(queueItem => queueItem.Order),
(original, order) => original.Order = order);
        await dbContext.SaveChangesAsync();
        return Ok();
    }

    // POST: api/Queue/Entry
    [HttpPost]
    [Route("Entry")]
    public async Task<IHttpActionResult> PostEntry(AddQueueEntryModel
queueItem)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        if (queueItem.EntrantId == queueItem.QueueId)
            return BadRequest();

        var userId = User.Identity.GetUserId<int>();
        if (queueItem.EntrantId != userId && !await UserHasAccessToQueue(userId, queueItem.QueueId))
            return Unauthorized();

        var queueManager = await UserManager.FindByIdAsync(queueItem.QueueId);
        if (queueManager == null)
            return BadRequest();

        var entrant = await UserManager.FindByIdAsync(queueItem.EntrantId);
        if (entrant == null)
            return BadRequest();
    }

```

```

        return Ok(await AddQueueEntry(queueManager, entrant));
    }

    private async Task<QueueItem> AddQueueEntry(Employee queueManager,
Employee entrant)
    {
        var queue = queueManager.OwnQueueEntries.OrderBy(item => item.Or-
der).ToArray();

        var queueItem = new QueueItem()
        {
            ManagerId = queueManager.Id,
            EmployeeId = entrant.Id
        };

        var entrantType = await UserManager.GetUserType(entrant.Id);
        if (entrantType == UserType.Employee || entrantType == UserType.Secre-
tary || queue.Length == 0)
        {
            queueItem.Order = queue.Length;
        }
        else
        {
            var sliceSize = entrantType == UserType.Manager ? 2 : 3;
            if (queue.Length <= sliceSize)
                queueItem.Order = queue.Length;
            else
            {
                var currentSliceCount = 0;
                foreach (var item in queue)
                {
                    if (await UserManager.GetUserType(item.EmployeeId) == Us-
erType.Employee)
                    {
                        if (++currentSliceCount >= sliceSize)
                        {
                            queueItem.Order = item.Order;
                        }
                    }
                    else
                        currentSliceCount = 0;
                }
            }
        }
    }

```

```

    }

    for (int i = queueItem.Order + 1; i < queue.Length; i++)
        queue[i].Order += 1;

    dbContext.Queue.Add(queueItem);
    await dbContext.SaveChangesAsync();
    return queueItem;
}

private async Task<bool> UserHasAccessToQueue(int userId, int queueId)
{
    if (await UserManager.IsInRoleAsync(userId, RoleNames.Administrator))
        return true;

    if (await UserManager.IsInRoleAsync(userId, RoleNames.Manager) ||
        await UserManager.IsInRoleAsync(userId, RoleNames.Vice))
        return userId == queueId;

    var user = await UserManager.FindByIdAsync(userId);
    return user.ManagedQueues.Any(queue => queue.Id == queueId);
}

}

[Authorize]
[RoutePrefix("api/Account")]
public class AccountController : ApiController
{
    private ApplicationUserManager _userManager;

    public AccountController()
    {
    }

    public AccountController(ApplicationUserManager userManager)
    {
        UserManager = userManager;
    }

    public ApplicationUserManager UserManager
    {
        get
        {

```

```

        return _userManager ?? Request.GetOwinContext().GetUserMan-
ager<ApplicationUserManager>();
    }
    private set
    {
        _userManager = value;
    }
}

// POST api/Account/Logout
[Route("Logout")]
public IHttpActionResult Logout()
{
    Authentication.SignOut(User.Identity.AuthenticationType);
    return Ok();
}

// POST api/Account/ChangePassword
[Route("ChangePassword")]
public async Task<IHttpActionResult> ChangePassword(ChangePassword-
BindingModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    IdentityResult result = await UserManager.ChangePasswordAsync(Con-
vert.ToInt32(User.Identity.GetUserId()), model.OldPassword,
        model.NewPassword);

    if (!result.Succeeded)
    {
        return GetErrorResult(result);
    }

    return Ok();
}

// POST api/Account/SetPassword
[Authorize(Roles = RoleNames.Administrator)]
[Route("SetPassword")]
public async Task<IHttpActionResult> SetPassword(SetPasswordBinding-
Model model)

```

```

{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    IdentityResult result;
    if (UserManager.HasPassword(model.UserId))
    {
        result = await UserManager.RemovePasswordAsync(model.UserId);

        if (!result.Succeeded)
            return GetErrorResult(result);
    }

    result = await UserManager.AddPasswordAsync(model.UserId,
model.NewPassword);

    if (!result.Succeeded)
        return GetErrorResult(result);

    return Ok();
}

// POST api/Account/Register
[AllowAnonymous]
[Route("Register")]
public async Task<IHttpActionResult> Register(RegisterBindingModel
model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = new Employee()
    {
        UserName = model.UserName,
        FirstName = model.FirstName,
        Middlename = model.Middlename,
        LastName = model.LastName,
        PositionId = model.PositionId
    };

    var result = await UserManager.CreateAsync(user, model.Password);
    if (!result.Succeeded)

```

```

        return GetErrorResult(result);

        var role = ApplicationUserManager.GetRoleFromUserType(model.UserType);
        if (!string.IsNullOrEmpty(role))
            await UserManager.AddToRoleAsync(user.Id, role);

        return Ok();
    }

    // GET api/Account
    public async Task<Employee> Get()
    {
        var user = await UserManager.FindByIdAsync(User.Identity.GetUserId<int>());

        var userRoles = await UserManager.GetRolesAsync(user.Id);
        user.Type = ApplicationUserManager.GetUserTypeFromRoles(userRoles);
        user.IsAdministrator = userRoles.Contains(RoleNames.Administrator);

        user.PasswordHash = null;

        return user;
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing && _userManager != null)
        {
            _userManager.Dispose();
            _userManager = null;
        }

        base.Dispose(disposing);
    }

    #region Helpers

    private IAuthenticationManager Authentication
    {
        get { return Request.GetOwinContext().Authentication; }
    }

    private IHttpActionResult GetErrorResult(IdentityResult result)

```

```

{
    if (result == null)
    {
        return InternalServerError();
    }

    if (!result.Succeeded)
    {
        if (result.Errors != null)
        {
            foreach (string error in result.Errors)
            {
                ModelState.AddModelError("", error);
            }
        }

        if (ModelState.IsValid)
        {
            // No ModelState errors are available to send, so just return an empty
BadRequest.
            return BadRequest();
        }

        return BadRequest(ModelState);
    }

    return null;
}

#endregion
}

public class ServiceClient
{
    #region Constants

    private static class ResourceUri
    {
        public class Account
        {
            public static readonly string Base = "/api/Account";
            public static readonly string Register = Base + "/Register";
        }
    }
}

```



```

    public static readonly string Token = "/Token";
    public static readonly string Position = "/api/Position";
}

private static readonly string ServiceUriSettingName = "ServiceUri";

#endregion

private Lazy<HttpClient> httpClientInstance = new Lazy<HttpClient>(CreateHttpClient);
private HttpClient HttpClient => httpClientInstance.Value;

private TokenEndpointGrantResponse TokenInfo { get; set; }

public Employee CurrentUser { get; set; }

private ServiceClient()
{
}

#region Account

public async Task SignIn(string userName, string password)
{
    var content = new FormUrlEncodedContent(new[]
    {
        new KeyValuePair<string, string>("grant_type", "password"),
        new KeyValuePair<string, string>("username", userName),
        new KeyValuePair<string, string>("password", password)
    });

    HttpResponseMessage response;
    try
    {
        response = await HttpClient.PostAsync(ResourceUri.Token, content);
    }
    catch (HttpRequestException exception)
    {
        throw new ServiceCommunicationException(Resources.ConnectionFailedExceptionMessage, exception);
    }

    if (!response.IsSuccessStatusCode)
    {

```

```

        TokenEndpointErrorResponse error = null;
        try
        {
            error = await response.Content.ReadAsAsync<TokenEndpointError-
Response>();
        }
        catch (Exception)
        {
        }

        if (error != null && !string.IsNullOrEmpty(error.ErrorDescription))
            throw new SignInFailedException(error.ErrorDescription);

        response.EnsureSuccessStatusCode();
    }

    try
    {
        var tokenInfo = await response.Content.ReadAsAsync<TokenEndpoint-
GrantResponse>();
        if (string.IsNullOrEmpty(tokenInfo.AccessToken))
            throw new ServiceCommunicationException();
        TokenInfo = tokenInfo;
    }
    catch (Exception exception)
    {
        throw new ServiceCommunicationException(Resources.SignInErrorEx-
ceptionMessage, exception);
    }

    HttpClient.DefaultRequestHeaders.Authorization = new Sys-
tem.Net.Http.Headers.AuthenticationHeaderValue("Bearer", TokenInfo.Ac-
cessToken);

    try
    {
        CurrentUser = await Get<Employee>(ResourceUri.Account.Base);
    }
    catch (HttpRequestException exception)
    {
        throw new RequestFailedException(Resources.UserFetchFailedExcep-
tionMessage, exception);
    }

```

```

        if (CurrentUser == null)
            throw new RequestFailedException(Resources.UserFetchFailedExceptionMessage);
    }

    public async Task Register(RegisterBindingModel model)
    {
        try
        {
            await Post(ResourceUri.Account.Register, model);
        }
        catch (HttpRequestException exception)
        {
            throw new RequestFailedException(Resources.RegistrationFailedExceptionMessage, exception);
        }
    }

    #endregion

    #region Position

    public async Task<IEnumerable<Position>> GetPositions()
    {
        return await Get<IEnumerable<Position>>(ResourceUri.Position);
    }

    public async Task<Position> CreatePosition(Position position)
    {
        try
        {
            return await Post(ResourceUri.Position, position);
        }
        catch (HttpRequestException exception)
        {
            throw new RequestFailedException(string.Format(Resources.UnableToCreateErrorMessageTemplate, Resources.PositionFieldLabel), exception);
        }
    }

    #endregion

    #region Generic

```

```

private async Task<T> Get<T>(string resourceUri)
{
    HttpResponseMessage response;
    try
    {
        response = await HttpClient.GetAsync(resourceUri);
    }
    catch (HttpRequestException exception)
    {
        throw new ServiceCommunicationException(Resources.Connection-
FailedExceptionMessage, exception);
    }

    response.EnsureSuccessStatusCode();
    return await response.Content.ReadAsAsync<T>();
}

private async Task<T> Post<T>(string resourceUri, T item)
{
    HttpResponseMessage response;
    try
    {
        response = await HttpClient.PostAsJsonAsync(resourceUri, item);
    }
    catch (HttpRequestException exception)
    {
        throw new ServiceCommunicationException(Resources.Connection-
FailedExceptionMessage, exception);
    }

    if (response.IsSuccessStatusCode)
    {
        try
        {
            return await response.Content.ReadAsAsync<T>();
        }
        catch
        {
            return default(T);
        }
    }

    InvalidModelResponse modelResponse = null;
    try

```

```

        {
            modelResponse = await response.Content.ReadAsAsync<InvalidModelResponse>();
        }
        catch (Exception)
        {
        }

        if (modelResponse != null)
        {
            if (modelResponse.ModelState != null && modelResponse.ModelState.Any())
                throw new RequestFailedException(string.Join(Environment.NewLine, modelResponse.ModelState.SelectMany(kvp => kvp.Value)));
            if (!string.IsNullOrEmpty(modelResponse.Message))
                throw new RequestFailedException(modelResponse.Message);
        }

        response.EnsureSuccessStatusCode();
        return default(T);
    }

#endregion

#region Instance

    public static readonly Lazy<ServiceClient> Instance = new Lazy<ServiceClient>(CreateInstance);

    private static ServiceClient CreateInstance() => new ServiceClient();

    private static HttpClient CreateHttpClient()
    {
        var serviceUriSetting = ConfigurationManager.AppSettings[ServiceUriSettingName];
        if (string.IsNullOrEmpty(serviceUriSetting))
            throw new ServiceCommunicationException(Resources.ServiceUriNotSetExceptionMessage);

        Uri serviceUri;
        if (!Uri.TryCreate(serviceUriSetting, UriKind.RelativeOrAbsolute, out serviceUri))
            throw new ServiceCommunicationException(Resources.InvalidServiceUriExceptionMessage);
    }

```

```

        var httpClient = new HttpClient();
        httpClient.BaseAddress = serviceUri;

        return httpClient;
    }

#endregion

#region Models

private class TokenEndpointErrorResponse
{
    [JsonProperty("error")]
    public string Error { get; set; }
    [JsonProperty("error_description")]
    public string ErrorDescription { get; set; }
}

private class TokenEndpointGrantResponse
{
    [JsonProperty("access_token")]
    public string AccessToken { get; set; }
    [JsonProperty("token_type")]
    public string TokenType { get; set; }
    [JsonProperty("expires_in")]
    public int ExpiresIn { get; set; }
    [JsonProperty("user_name")]
    public string UserName { get; set; }
    [JsonProperty(".issued")]
    public DateTime Issued { get; set; }
    [JsonProperty(".expires")]
    public DateTime Expires { get; set; }
}

private class InvalidModelResponse
{
    public string Message { get; set; }
    public IDictionary<string, IEnumerable<string>> ModelState { get; set; }
}

#endregion
}

```

```

public static class CredentialManager
{
    public static Tuple<string, string> GetCredential()
    {
        var credential = new Credential();
        credential.Target = GetProductName();
        credential.Load();

        if (string.IsNullOrEmpty(credential.Username) || string.IsNullOrEmpty(credential.Password))
        {
            credential.Dispose();
            return null;
        }

        var result = new Tuple<string, string>(credential.Username, credential.Password);
        credential.Dispose();
        return result;
    }

    public static void SaveCredential(string userName, SecureString password)
    {
        var credential = new Credential(userName);
        credential.SecurePassword = password;
        credential.Target = GetProductName();
        credential.PersistantType = PersistantType.LocalComputer;

        credential.Save();

        credential.Dispose();
    }

    private static string GetProductName()
    {
        var versionInfo = FileVersionInfo.GetVersionInfo(Assembly.GetEntryAssembly().Location);
        return versionInfo.ProductName;
    }
}

private static ServiceClient ServiceClient => ServiceClient.Instance.Value;

#region Properties

```

```

private string firstName;
public string FirstName
{
    get
    {
        return firstName;
    }

    set
    {
        firstName = value;
        NotifyPropertyChanged(nameof(FirstName));
    }
}

private string middleName;
public string MiddleName
{
    get
    {
        return middleName;
    }

    set
    {
        middleName = value;
        NotifyPropertyChanged(nameof(MiddleName));
    }
}

private string lastName;
public string LastName
{
    get
    {
        return lastName;
    }

    set
    {
        lastName = value;
        NotifyPropertyChanged(nameof(LastName));
    }
}

```



```

}

private string userName;
public string UserName
{
    get
    {
        return userName;
    }

    set
    {
        userName = value;
        NotifyPropertyChanged(nameof(UserName));
    }
}

private PasswordBox passwordBox;
public PasswordBox PasswordBox
{
    get
    {
        return passwordBox;
    }

    set
    {
        passwordBox = value;
        NotifyPropertyChanged(nameof(PasswordBox));
    }
}

private PasswordBox confirmPasswordBox;
public PasswordBox ConfirmPasswordBox
{
    get
    {
        return confirmPasswordBox;
    }

    set
    {
        confirmPasswordBox = value;
        NotifyPropertyChanged(nameof(ConfirmPasswordBox));
    }
}

```

```

    }
}

private bool isBusy;
public bool IsBusy
{
    get
    {
        return isBusy;
    }

    set
    {
        isBusy = value;
        NotifyPropertyChanged(nameof(IsBusy));
    }
}

private string errorMessage;
public string ErrorMessage
{
    get
    {
        return errorMessage;
    }

    set
    {
        errorMessage = value;
        NotifyPropertyChanged(nameof(ErrorMessage));
    }
}

private IEnumerable<Position> positions;
public IEnumerable<Position> Positions
{
    get
    {
        return positions;
    }

    set
    {
        positions = value;
    }
}

```

```

        NotifyPropertyChanged(nameof(Positions));
    }
}

private Position selectedPosition;
public Position SelectedPosition
{
    get
    {
        return selectedPosition;
    }

    set
    {
        selectedPosition = value;
        NotifyPropertyChanged(nameof(SelectedPosition));
    }
}

private string positionComboBoxText;
public string PositionComboBoxText
{
    get
    {
        return positionComboBoxText;
    }

    set
    {
        positionComboBoxText = value;
        NotifyPropertyChanged(nameof(PositionComboBoxText));
    }
}

private IEnumerable<Tuple<UserType, string>> userTypes;
public IEnumerable<Tuple<UserType, string>> UserTypes
{
    get
    {
        return userTypes;
    }

    set
    {

```

```

        userTypes = value;
        NotifyPropertyChanged(nameof(UserTypes));
    }
}

private Tuple<UserType, string> selectedUserType;
public Tuple<UserType, string> SelectedUserType
{
    get
    {
        return selectedUserType;
    }

    set
    {
        selectedUserType = value;
        NotifyPropertyChanged(nameof(SelectedUserType));
    }
}

#endregion

#region Commands

public ICommand RegisterCommand => new AsyncDelegateCommand(Register);

public ICommand CancelCommand => new AsyncDelegateCommand(Cancel);

#endregion

public event EventHandler<RegistrationFinishedEventArgs> RegistrationFinished;

public RegistrationViewModel()
{
    Task.Run(new Action(LoadPositions));

    userTypes = new[]
    {
        new Tuple<UserType, string>(UserType.Employee, Resources.EmployeeRoleName),

```

```

        new Tuple<UserType, string>(UserType.Secretary, Resources.Secretar-
yRoleName),
        new Tuple<UserType, string>(UserType.Vice, Resources.ViceManager-
RoleName),
        new Tuple<UserType, string>(UserType.Manager, Resources.Manager-
RoleName)
    };
    selectedUserType = userTypes.First();
}

private async Task Register()
{
    ErrorMessage = null;
    if (!ValidateForm())
        return;

    IsBusy = true;
    try
    {
        var position = SelectedPosition;
        if (position == null && !string.IsNullOrEmpty(PositionComboBox-
Text))
        {
            position = await CreatePosition(PositionComboBoxText);
            if (position == null)
                return;
        }

        var model = new RegisterBindingModel()
        {
            FirstName = FirstName,
            Middlename = MiddleName,
            LastName = LastName,
            PositionId = position.Id,
            UserType = selectedUserType.Item1,
            UserName = UserName,
            Password = PasswordBox.Password,
            ConfirmPassword = ConfirmPasswordBox.Password
        };

        try
        {
            await ServiceClient.Register(model);

```

```

        RegistrationFinished?.Invoke(this, new RegistrationFinishedEven-
tArgs(true));
    }
    catch (Exception exception)
    {
        ErrorMessage = exception.Message;
    }
}
finally
{
    IsBusy = false;
}
}

private async Task Cancel()
{
    RegistrationFinished?.Invoke(this, new RegistrationFinishedEven-
tArgs(false));
}

private async void LoadPositions()
{
    IEnumerable<Position> positions;
    try
    {
        positions = await ServiceClient.GetPositions();
    }
    catch (Exception exception)
    {
        ErrorMessage = exception.Message;
        return;
    }

    Positions = positions;
}

private async Task<Position> CreatePosition(string positionComboBoxText)
{
    var position = new Position() { JobTitle = positionComboBoxText };

    try
    {
        return await ServiceClient.CreatePosition(position);
    }
}

```

```

        catch (Exception exception)
        {
            ErrorMessage = exception.Message;
            return null;
        }
    }

    private bool ValidateForm()
    {
        var errorList = new List<string>();
        var validationResult = true;

        if (string.IsNullOrEmpty(FirstName))
        {
            errorList.Add(string.Format(Resources.MissingFieldErrorMessageTemplate, Resources.FirstNameFieldLabel));
            validationResult = false;
        }

        if (string.IsNullOrEmpty(LastName))
        {
            errorList.Add(string.Format(Resources.MissingFieldErrorMessageTemplate, Resources.LastNameFieldLabel));
            validationResult = false;
        }

        if (string.IsNullOrEmpty(LastName))
        {
            errorList.Add(string.Format(Resources.MissingFieldErrorMessageTemplate, Resources.PositionFieldLabel));
            validationResult = false;
        }

        if (string.IsNullOrEmpty(UserName))
        {
            errorList.Add(string.Format(Resources.MissingFieldErrorMessageTemplate, Resources.UserNameFieldLabel));
            validationResult = false;
        }

        if (string.IsNullOrEmpty>PasswordBox.Password))
        {
            errorList.Add(string.Format(Resources.MissingFieldErrorMessageTemplate, Resources.PasswordFieldLabel));
        }
    }

```

```

        validationResult = false;
    }

    if (!string.Equals>PasswordBox.Password, ConfirmPasswordBox.Password, StringComparison.Ordinal))
    {
        errorList.Add(Resources.PasswordsDoNotMatchErrorMessage);
        validationResult = false;
    }

    if (errorList.Any())
        ErrorMessage = string.Join(Environment.NewLine, errorList);

    return validationResult;
}

public class RegistrationFinishedEventArgs : EventArgs
{
    public bool RegistrationSucceeded { get; set; }

    public RegistrationFinishedEventArgs()
    {
    }

    public RegistrationFinishedEventArgs(bool registrationSucceeded)
    {
        RegistrationSucceeded = registrationSucceeded;
    }
}

{
    private static ServiceClient ServiceClient => ServiceClient.Instance.Value;

    #region Properties

    private string userName;
    public string UserName
    {
        get
        {
            return userName;
        }
    }
}

```



```

        set
        {
            userName = value;
            NotifyPropertyChanged(nameof(UserName));
        }
    }

    private PasswordBox passwordBox;
    public PasswordBox PasswordBox
    {
        get
        {
            return passwordBox;
        }

        set
        {
            passwordBox = value;
            NotifyPropertyChanged(nameof(PasswordBox));
            InitializeForm();
        }
    }

    private bool rememberMe;
    public bool RememberMe
    {
        get
        {
            return rememberMe;
        }

        set
        {
            rememberMe = value;
            NotifyPropertyChanged(nameof(RememberMe));
        }
    }

    private bool isBusy;
    public bool IsBusy
    {
        get
        {
            return isBusy;
        }
    }

```

```

    }

    set
    {
        isBusy = value;
        NotifyPropertyChanged(nameof(IsBusy));
    }
}

private string errorMessage;
public string ErrorMessage
{
    get
    {
        return errorMessage;
    }

    set
    {
        errorMessage = value;
        NotifyPropertyChanged(nameof(ErrorMessage));
    }
}

#endregion

#region Commands

    public ICommand SignInCommand => new AsyncDelegateCommand(SignIn);

    public ICommand RegisterCommand => new AsyncDelegateCommand(Register);

#endregion

public event EventHandler SignInSucceeded;

private async Task SignIn()
{
    ErrorMessage = null;
    IsBusy = true;

    try

```

```

        {
            await ServiceClient.SignIn(UserName, PasswordBox.Password);
        }
        catch (Exception exception)
        {
            ErrorMessage = exception.Message;
            IsBusy = false;
            return;
        }

        if (RememberMe)
            CredentialManager.SaveCredential(UserName, PasswordBox.Secure-
Password);

        IsBusy = false;
        ShowMainWindow();
        SignInSucceeded?.Invoke(this, EventArgs.Empty);
    }

    private async Task Register()
    {
        var registrationWindow = new RegistrationWindow();
        registrationWindow.ShowDialog();
    }

    private void ShowMainWindow()
    {
        var mainWindow = new MainWindow();
        mainWindow.Show();
        mainWindow.Activate();
    }

    private void InitializeForm()
    {
        var credential = CredentialManager.GetCredential();
        if (credential != null)
        {
            RememberMe = true;
            UserName = credential.Item1;
            PasswordBox.Password = credential.Item2;
        }
    }
}

```