СОДЕРЖАНИЕ

[Введение 6](#_Toc484670378)

[1 Анализ информационных источников 8](#_Toc484670379)

[1.1 Роль электронных систем очередей 8](#_Toc484670380)

[1.2 Оборудование и программное обеспечение электронных очередей 10](#_Toc484670381)

[1.3 Обзор аналогов существующих систем 14](#_Toc484670382)

[2 Моделирование предметной области 18](#_Toc484670383)

[2.1 Алгоритм работы системы электронной очереди 18](#_Toc484670384)

[2.2 Обоснование необходимости разработки 20](#_Toc484670385)

[2.3 Разработка диаграммы вариантов использования 21](#_Toc484670386)

[2.4 Разработка информационной модели системы 22](#_Toc484670387)

[3 Раазработка программного модуля 24](#_Toc484670388)

[3.1 Постановка задачи и описание системы 24](#_Toc484670389)

[3.2 Обоснование проектных решений по информационному обеспечению 24](#_Toc484670390)

[3.3 Обоснование проектных решений по программному обеспечению 26](#_Toc484670391)

[3.4 Разработка моделей представления программного модуля 29](#_Toc484670392)

[3.5 Организация технологии сбора, передачи, обработки информации 34](#_Toc484670393)

[3.6 Разработка структурной схемы программного модуля 35](#_Toc484670394)

[4 Тестирование программного модуля 38](#_Toc484670395)

[5 Руководство пользователя 39](#_Toc484670396)

[6 Технико-экономическое обоснование 42](#_Toc484670397)

[6.1 Характеристика программного модуля и исходные данные 42](#_Toc484670398)

[6.2. Расчет затрат и отпускной цены программного средства 43](#_Toc484670400)

[6.3. Расчет сметы затрат и цены заказного ПО 45](#_Toc484670401)

[6.4. Оценка экономической эффективности применения программного средства у пользователя 49](#_Toc484670402)

[Заключение 54](#_Toc484670403)

[Список использованных источников 56](#_Toc484670405)

[Приложение А Исходный код программы 57](#_Toc484670406)

# ВВЕДЕНИЕ

документооборот управленческий электронный автоматизированный

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 внутренней навигации — это программно-аппаратное решение, дополняющее функциональность электронной очереди. Эта система, позволяет задавать правила навигации внутри помещения, где развернута система управления очередью. Минимальный функционал такой системы выглядит следующим образом — на главное табло выводятся стрелки, указывающие направление к соответствующему оператору. Однако в помещениях сложной конфигурации в таком виде система не применима. В таком случае статическая карта навигации отображается на отдельном табло или возможно динамическое отображение на киоске, если в соответствующем меню ввести номер талона после вызова.

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 на операторов, а также использовать информационные табло для отображения рекламной информации. 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.

Клиент подходит к пульту регистрации и получает талон с номером очереди. Далее он, находясь в зоне ожидания, наблюдает за прохождением очереди. Оператор вызывает посетителей с помощью пульта оператора. При вызове очередного посетителя на главном табло и табло оператора появляется номер очереди клиента и номер стойки (номер табло оператора), также номер очереди объявляется в системе голосового оповещения. После оказания услуги оператор вызывает следующего клиента. Клиент может оценить качество оказания услуги с помощью аппаратного пульта системы оценки качества или на пункте регистрации.

Целью исследования является обобщение и систематизация опыта совершенствования формирования электронной очереди в организации, разработка и создание автоматизированной системы управления формирования электронной очереди.

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

Для достижения цели при выполнении дипломного проекта были поставлены следующие задачи:

1. раскрыть понятие и назначение формирования электронной очереди;
2. проанализировать существующие системы формирования электронной очереди;
3. произвести оценку существующей системы формирования электронной очереди;
4. выявить резервы повышения эффективности функционирования системы формирования электронной очереди;
5. разработать предложения по повышению эффективности работы системы формирования электронной очереди.

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

1. запись в очередь на прием к начальникам и заместителям начальников организации;
2. управление очередью секретарем, заместителями и начальниками, а именно удаление, изменение порядка;
3. 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.

При написании работы использовалась литература по экономике предприятия, по анализу хозяйственной деятельности любого предприятия.

# 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 (ГАИ) и др. Системы управления очередью изготавливаются на базе электронных табло.

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, пульты 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, когда никто не помешает конфиденциальной работе банковского служащего с клиентом. В любой момент сотрудники могут получить практически любую статистическую информацию об обслуживании клиентов и работе операционистов. 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 где-нибудь да возникают очереди.

А ведь именно эмоции, которые выносят клиенты из очередей, – главное, что определяет их отношение к уровню полученного сервиса в целом и перспективы повторного обслуживания (покупки) в этом месте.

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, привносящую порядок, предсказуемость и справедливость в очередь.

Наряду с решением задач управления потоками клиентов имеется возможность формировать с помощью этой системы различные статистические базы данных.

Применение системы электронной очереди в операционно-кассовом зале 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 операциониста (для оценки производительности труда каждого кассира 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 кассира, количестве работающих касс, 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. И все это при отсутствии «живой» очереди со всеми ее отрицательными моментами.

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 цивилизованного подхода к решению проблем, связанных с обслуживанием клиентов, только подтверждает, что за такими системами будущее.

Электронная очередь имеет два главных преимущества:

* комфорт клиента: можно получить информацию о требующейся услуге, распечатать талончик с указанием времени вызова по электронной очереди, записаться на прием, 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 системы управления очередью входят: пункт регистрации, центральное информационное табло, мини-табло для каждого окна оператора, сервер (в качестве которого может выступать сенсорный киоск), система WEB регистрации, система администрирования СУО.

Пульт регистрации – устройство, позволяющее клиенту выбрать услугу и получить номер очереди (талон с номером). Пульты регистрации бывают сенсорные и кнопочные. Сенсорные пульты 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.

На экране пульта выбора услуг возможен вывод любой графической информации, в частности, с использованием элементов фирменного стиля организации/учреждения, а также вывод любого мультимедиа - контента (бегущая строка, видеоролик).

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



Рисунок 1.1 – Пульт регистрации

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 применяется в случае, если политика безопасности учреждения не допускает использования существующей ЛВС и строго регламентирует состав программ, которые могут быть инсталлированы на ПК операторов.

Физический пульт оператора устанавливается на рабочем месте оператор, 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 ЛВС и ПК операторов, то возможно использование программного модуля Пульт оператора, возможности которого гораздо шире, чем у физического пульта [1].



Рисунок 1.2 – Пульт оператора

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» (ПППО). Тот, кто приходит первым, тот и обслуживается первым, пришедший следующим ждёт, пока обслуживание первого не будет закончено, и так далее.

На светодиодное информационное табло 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/ учреждения), а также вывод любой графической информации и любого мультимедиа-контента (бегущая строка, видеоролик) [1].

Табло оператора – устройство, позволяющее продублировать информацию главного табло по конкретному вызванному клиенту. 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 [1].

Табло оператора может быть выполнено на основе матричных светодиодных табло.

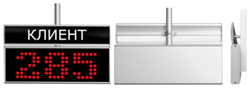


Рисунок 1.3 – Светодиодное табло

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 пульты 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 качеств

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. Нажимая ту или иную копку, посетитель оценивает качество услуг, предоставленных сотрудником организации. Так же может быть реализована система оценки качества услуг при выходе посетителя, т.е. – для разблокировки турникета, находящегося при выходе, посетитель должен 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/рекомендацию [1].

# 1.3 Обзор аналогов существующих систем

AKIS Micro является эффективным решением для организаций, в которых все услуги «объединены в одну очередь». Компактный терминал выдаёт следующий билет автоматически. Движение очереди отображается на светодиодных экранах и табло. Каждый оператор может быть оборудован рабочим светодиодным дисплеем для отображения номера вызываемого клиента. Система поддерживает до шести рабочих мест операторов.



Рисунок 1.5 – Аппаратный пульт системы AKIS Micro

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

Электронная очередь Meta-Q – необходимый инструмент организации клиентского потока и повышения качества клиентского сервиса для любой организации, чья деятельность связана с предоставлением услуг.

Упорядочить потоки порой требуется не только в офисах обслуживания. Задачи по управлению очередью и навигации в ходе бизнес- процесса возникают в самых разных организациях.

Решить любые задачи по организации очереди позволяют продукты Meta-Q.

Meta-Q–широкая гамма решений и огромный опыт в области организации складской логистики, сфере обслуживания и многоуровневого сервиса.

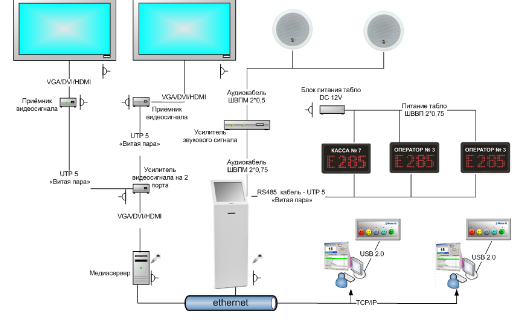


Рисунок 1.6 – Аппаратный пульт системы Meta-Q

Система управления очередью IBA QueueMASTER предназначена для управления очередью, обеспечения удобства и комфорта клиентам, позволяет оптимизировать работу по обслуживанию клиентов.

Функции:

* 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 (ветераны войны и труда, постоянные клиенты, VIP-клиенты).

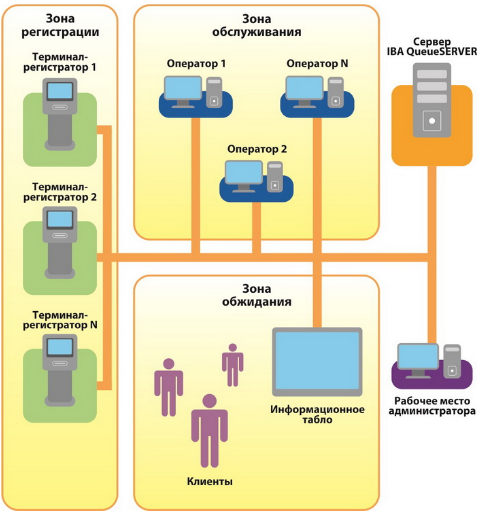


Рисунок 1.7 – Аппаратный пульт системы IBA QueueMASTER

**1.4 Постановка задачи разрабатываемого модуля**

Программное средство предназначено для управления очередями к руководителям государственного предприятия.

Программное средство должно обеспечивать работу с пятью типами пользователей:

1. сотрудник – может просматривать список доступных очередей (руководителей), становиться в очередь, просматривать список очередей, в которых он стоит, и его позицию в этих очередях
2. руководитель – может просматривать свою очередь, удалять из нее, изменять порядок следования сотрудников в очереди, предоставлять и забирать доступ к очереди делопроизводителям
3. заместитель руководителя – идентичен руководителю; обладает более высоким приоритетом, чем сотрудник
4. делопроизводитель – может управлять (аналогично владельцу очереди) другими очередями, доступ к очередям делопроизводителю дается владельцем очереди либо администратором;
5. администратор – обладает доступом к просмотру и изменению всей информации (пользователи, должности, очереди).

При постановке в очередь позиция определяется в соответствии с типом пользователя. Руководители обладают наибольшим приоритетом, заместители меньшим, сотрудники наименьшим. При этом, любое количество руководителей и заместителей не должно блокировать остальных сотрудников.

Поэтому, определение позиции должно осуществляется следующим образом:

1. Сотрудники становятся в конец.
2. Заместители: в очереди начиная с начала выбирается первая непрерывная последовательность из 3-х сотрудников, заместитель становится на место последнего в этой последовательности.
3. Руководители: аналогично, но с размером последовательности, равным 2.

Каждый пользователь обладает свойствами, которые они указывают при регистрации. Эти свойства могут изменяться самим пользователем и администратором. Свойства пользователей включают в себя:

1. имя;
2. отчество;
3. фамилия;
4. должность - может иметь приоритет. Этот приоритет не учитывается при постановке в очередь, но виден при просмотре очереди, и может использоваться владельцем или делопроизводителем для изменения позиции сотрудника;
5. логин;
6. пароль.

Доступ в систему осуществляется с помощью логина и пароля. Длина пароля 6-16 символов, должен содержать цифры и латинские символы, может содержать спецсимволы.

Логин и пароль могут сохраняться клиентским приложением в диспетчере учетных данных Windows (Windows Credential Manager). 5 неудачных попыток входа приводят к блокировке, которая может быть снята администратором.

Программное средство должно поддерживаться ОС семейства Windows.

Клиентское приложение реализовано на технологии WPF. Серверное приложение реализовано на основе ASP.NET Web API, для доступа к данным используется технология Entity Framework.

Язык программирования C#, для разработки используется среда Microsoft Visual Studio 2015. Данные должны хранится в реляционной базе данных СУБД Microsoft SQL Server.

# 2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## 2.1 Алгоритм работы системы электронной очереди

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 «Получить талон». Пульт выбора вопросов выдает ему талон, на котором напечатан список вопросов, индивидуальный номер очереди посетителя и штрих-код, соответствующий этому номеру, а также любая другая 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 (заполнить бланки, подготовить документы и пр.)

Посетитель проходит в зону ожидания и следит за информационным табло и слушает сообщения «автоинформатора».

На рабочем месте оператора (оператора), установлен компьютер со сканером штрих-кода. На компьютере запущена программа «Пульт электронной очереди», которая позволяет:

* вызывать следующего посетителя;
* повторять вызов еще раз (если тот не подошел);
* 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.

Система приглашает посетителя сообщением автоинформатора:

«Посетитель номер 34, пройдите к стойке 2 повторяю, посетитель номер 34, пройдите к стойке 2.

Сообщение автоинформатора 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 гражданину в приеме. Если номера совпадают, то в рабочем окне программы отображается информация о подошедшем гражданине, по каким вопросам он подошел и.т.д. Ориентируясь на эту информацию, оператор может правильно проинформировать посетителя о его месте в очереди.

Если подошел не тот посетитель или не подошел никто, оператор нажимает клавишу «Повтор».

Если к оператору не подошел вызываемый посетитель и после повторного вызова, то оператор должен нажать клавишу «Отложить». В этом случае система переносит номер вызываемого посетителя в отложенную очередь. После этого система выжидает определенное время (например, 10 мин.) и повторяет вызов посетителя из отложенной очереди. В этом случае сообщение автоинформатора будет следующим:

Повторный вызов для посетителя номер 34 пройдите к стойке номер 2 !

Если к оператору и после этого сообщения не подошел вызываемый посетитель, оператор снова нажимает клавишу «Отложить». После этого 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 во 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.

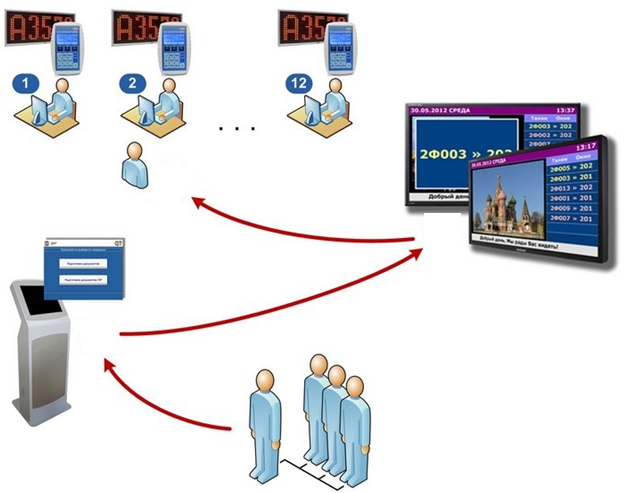


Рисунок 2.1– Алгоритм работы со стороны клиента

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 (обеды, технические перерывы и т.п.) Система управления электронной очередью генерирует множество отчетов, на основании которых можно предпринимать те или иные управленческие решения.

Система сама 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.

## 2.2 Обоснование необходимости разработки

Как известно из технического задания на разработку дипломного проекта, при постановке в очередь позиция определяется в соответствии с типом пользователя. Руководители обладают наибольшим приоритетом, заместители меньшим, сотрудники наименьшим. При этом, любое количество руководителей и заместителей не должно блокировать остальных сотрудников.

Сотрудники становятся в конец очереди, заместители, начиная с начала выбирается первая непрерывная последовательность из 3-х сотрудников, заместитель становится на место последнего в этой последовательности, а руководители аналогично, но с размером последовательности равную двум. Рассмотрим подробно алгоритм приоритетной очереди.

Очередь нужна для обработки периодов временных перегрузок, когда сетевое устройство не может передавать пакеты на выходной интерфейс в том темпе, в котором они поступают для выполнения такого продвижения. Главным по степени влияния на возникновение очередей фактором является коэффициент нагрузки устройства (utilization) - отношение средней интенсивности входного трафика устройства к средней интенсивности продвижения пакетов на выходной интерфейс.

Механизм приоритетной обработки трафика основан на разделении всего сетевого трафика на небольшое количество классов, а затем назначении каждому классу некоторого числового признака - приоритета. Пакеты могут разбиваться на приоритетные классы в соответствии с типом сетевого протокола - например, IP, IPX или DECnet, на основании адресов назначения и источника, идентификатора приложения и любых других комбинаций признаков, которые содержатся в пакетах.

Независимо от выбранного способа классификации трафика, в сетевом устройстве имеется несколько очередей в соответствии с количеством приоритетных классов. Поступивший пакет в периоды перегрузок помещается в очередь, соответствующую его приоритетному классу. На рис. ниже приведен пример использования четырех приоритетных очередей: с высоким, средним, нормальным и низким приоритетом. Данный вид очереди не подходит, исходя из того, что руководители обладают наибольшим приоритетом, заместители меньшим, сотрудники наименьшим. При этом, любое количество руководителей и заместителей не должно блокировать остальных сотрудников.

## 2.3 Разработка диаграммы вариантов использования

Диаграмма вариантов использования представлена на рисунке 2.2.

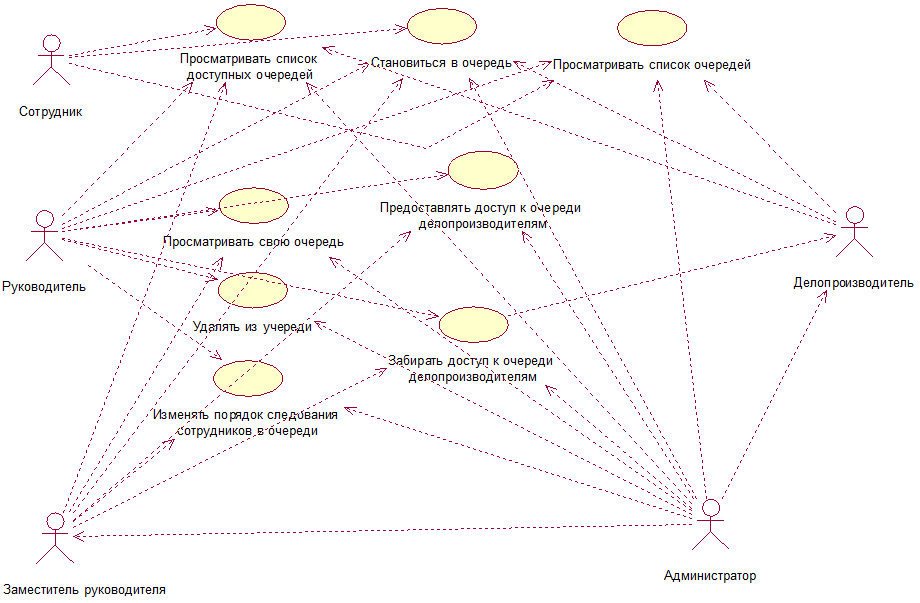


Рисунок 2.2 – Диаграмма вариантов использования

Диаграмма отображает следующие данные:

* сотрудник – может просматривать список доступных очередей (руководителей), становиться в очередь, просматривать список очередей, в которых он стоит, и его позицию в этих очередях
* руководитель – может просматривать свою очередь, удалять из нее, изменять порядок следования сотрудников в очереди, предоставлять и забирать доступ к очереди делопроизводителям
* заместитель руководителя – идентичен руководителю, обладает более высоким приоритетом, чем сотрудник
* делопроизводитель – может управлять (аналогично владельцу очереди) другими очередями, доступ к очередям делопроизводителю дается владельцем очереди либо администратором;
* администратор – обладает доступом к просмотру и изменению всей информации (пользователи, должности, очереди).

## 2.4 Разработка информационной модели системы

Информационная модель системы представлена на рисунке 2.3.



Рисунок 2.3 – Информационная модель системы

Подробно опишем информационную модель:

* ​Employee - информация о пользователях. Поля FirstName, MiddleName, LastName содержат ФИО, PositionId - должность (ссылка на запись в таблице Position). Остальные поля содержат информацию об учетной записи пользователя, Многие из них не используются, существуют только для совместимости с системой аутентификации ASP.NET Identity 2.0;
* Position - информация о должностях. JobTitle - название, Priority – приоритет;
* Role - роли (типы) пользователей: секретарь, заместитель, руководитель, администратор;
* UserRole - таблица содержит информацию о принадлежности пользователей к ролям, отсутствие роли означает обычного сотрудника;
* UserClaim, UserLogin не используются и существуют только для совместимости с ASP.NET Identity.
* Queue - очереди. Эта таблица содержит информацию о всех вхождениях во все очереди. Manager\_Id - ссылка на владельца очереди (руководителя или заместителя, в очередь к кому становятся). Employee\_Id - ссылка на сотрудника, ставшего в очередь. Order - позиция (порядковый номер) этого сотрудника в очереди.
* QueueSecretary - таблица содержит информацию о секретарях, которые могут управлять чужими очередями. SecretaryId - ссылка на секретаря, ManagerId - ссылка на руководителя или заместителя, очередью которого управляет этот секретарь.

# 3 РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ

## 

## 3.1 Постановка задачи и описание системы

Программный модуль предназначен для управления очередями к руководителям государственного предприятия.

Программное средство должно обеспечивать работу с пятью типами пользователей:

* сотрудник – может просматривать список доступных очередей (руководителей), становиться в очередь, просматривать список очередей, в которых он стоит, и его позицию в этих очередях
* руководитель – может просматривать свою очередь, удалять из нее, изменять порядок следования сотрудников в очереди, предоставлять и забирать доступ к очереди делопроизводителям
* заместитель руководителя – идентичен руководителю; обладает более высоким приоритетом, чем сотрудник
* делопроизводитель – может управлять (аналогично владельцу очереди) другими очередями, доступ к очередям делопроизводителю дается владельцем очереди либо администратором;
* администратор – обладает доступом к просмотру и изменению всей информации (пользователи, должности, очереди).

При постановке в очередь позиция определяется в соответствии с типом пользователя. Руководители обладают наибольшим приоритетом, заместители меньшим, сотрудники наименьшим. При этом, любое количество руководителей и заместителей не должно блокировать остальных сотрудников.

Доступ в систему осуществляется с помощью логина и пароля. Длина пароля 6-16 символов, должен содержать цифры и латинские символы, может содержать спецсимволы.

Логин и пароль могут сохраняться клиентским приложением в диспетчере учетных данных Windows (Windows Credential Manager). 5 неудачных попыток входа приводят к блокировке, которая может быть снята администратором.

Программное средство должно поддерживаться ОС семейства Windows.

Потребность в разработке программного модуля предназначена для управления очередями, а именно для управления очередями к руководителям государственного предприятия, обусловлена жесткой конкуренцией на рынке подобных услуг и необходимостью обеспечить более высокую производительность труда, большую надежность и достоверность информации, лучшую ее сохранность.

## 3.2 Обоснование проектных решений по информационному обеспечению

Документы, получаемые от внешних объектов и передаваемые им, должны храниться в базе данных.

Должна использоваться система управления базами данных (СУБД), основанная на реляционной модели баз данных (БД).

При вводе должен производиться контроль вводимых данных путем проверки установленных ограничений, выполнение заданных условий.

Для обеспечения хранения данных можно использовать локальную или распределенную базу данных. В данном дипломном проекте целесообразно использовать локальную базу данных, применение которой позволит уменьшить сложность разработки и настройки прикладного программного обеспечения.

Предполагаются следующие информационные решения, касающиеся разрабатываемого программного средства:

− сбор исходной информации, вводимой в базу данных, осуществляется локально;

− ввод информации в базу данных осуществляется вручную с бумажных носителей;

− обработка данных осуществляется в диалоговом режиме;

− пользователь получает информацию из базы данных на экран, кроме того, информация может выдаваться на принтер.

Схема представляет собой декомпозицию общего процесса решения задачи на отдельные процедуры преобразования массивов, именуемыми.

Структуру системы можно описать основными блоками, представленными на рисунке 3.1 [7].

Управление очередью

Главная форма

Выход

Формирование входной информации

Формирование списка очереди

Рисунок 3.1 − Структура системы

Модуль «Главная форма» предназначен для запуска основных функций программы и завершения работы с программой.

Назначение модуля «Управление очередью» является управление всем процессом электронной очереди, а именно постановка в очередь, движение очереди, выход с очереди, приоритет очереди.

Модуль «Формирование входной информации» предназначен для ввода первичных данных и просмотра ранее занесенных данных.

Модуль «Формирование документов» выполняет функции формирования информации о номере очереди. Отчеты формируются с использованием запросов, которые обрабатывают исходную информацию в соответствии с заданными параметрами пользователя.

Структура пакета прикладных программ содержит [8]:

− управляющий модуль, который предназначен для преобразования входных данных и действий пользователя в последовательность вызовов обрабатывающих модулей − формирование меню пользователя и обеспечение соединения пунктов меню с обрабатывающими и обслуживающими модулями;

− обрабатывающие и одновременно обслуживающие модули, которые предназначены для взаимодействия пользователя с пакетом управляющих модулей, а также для реализации алгоритма решения задач, т.е. для преобразования данных, составляющих информационную базу пакетов.

Обрабатывающие модули, для реализации алгоритма решения задач, т.е. для преобразования данных, составляющих информационную базу пакетов.

## 3.3 Обоснование проектных решений по программному обеспечению

Программное обеспечение включает совокупность компьютерных программ, описаний и инструкций по их применению на ПК. Программное обеспечение делится на две составляющие: общее (операционные системы, оболочки, компиляторы, интерпретаторы, СУБД, сетевые программы и т.д.) и специальное (совокупность прикладных программ, разработанных для конкретных задач) [7].

Для решения комплекса задач необходимо программное обеспечение, которое можно настроить на особенности конкретной предметной области, а также использование которого будет максимально эффективным для компании.

К операционной системе можно выдвинуть следующие требования [7]:

− требование надежности (критически важные структуры ядра системы, коды драйверов устройств должны быть доступны только для чтения, операционная система должна позволять пользователям и системным администраторам восстанавливать предыдущее состояние компьютерной системы без потери данных);

− требование понятности пользователю (интерфейс операционной системы должен быть «дружественным», изменение настроек системы может производиться в диалоговом режиме);

− требование защиты информации (операционная система должна иметь средства аутентификации и идентификации, иметь встроенные средства аудита доступа к информации);

− требование минимизации затрат на сопровождение и поддержку;

− требование эргономичности.

Менять существующую операционную систему Windows XP Professional на рабочих станциях нет необходимости, т.к. она удовлетворяет всем требованиям надежности, понятности пользователю, защиты информации, мобильности, масштабируемости, минимизации затрат на сопровождение и поддержку, эргономичности, описанным выше. Эта система вполне подходит под нужды автоматизированного комплекса задач.

К СУБД, используемой для разработки и использования базы данных, можно выдвинуть следующие требования [9]:

− требование надежности (СУБД должна позволять пользователям и системным администраторам восстанавливать предыдущее состояние СУБД без потери данных);

− требование защиты информации (СУБД должна иметь средства аутентификации и идентификации, иметь возможность разграничения доступа к объектам базы данных – различные группы пользователей должны иметь различные права на доступ к объектам базы данных);

− требование модифицируемости (база данных должна быть легко расширена при помощи добавления новых объектов);

− требование минимизации затрат на сопровождение и поддержку;

− требование эргономичности.

Основываясь на перечисленных выше критериях выбора СУБД был сделан выбор в пользу MSSQL Server, поскольку необходима СУБД, которая будет использоваться локально, объемы хранимой информации относительно невелики (измеряются мегабайтами), надежно работающая на сервере с техническими характеристиками обычного ПК.

Система SQL Server 2008 отталкивается от концепции платформы данных Майкрософт: она упрощает управление любыми данными в любом месте и в любой момент времени. Она позволяет хранить в базах данных информацию, полученную из структурированных, полуструктурированных и неструктурированных источников, таких как изображения и музыка. В SQL Server 2008 имеется большой набор интегрированных служб, расширяющих возможности использования данных: можно составлять запросы, выполнять поиск, проводить синхронизацию, делать отчеты, анализировать данные. Все данные хранятся на основных серверах, входящих в состав центра обработки данных. К ним осуществляется доступ с настольных компьютеров и мобильных устройств. Таким образом, осуществляется контроль над данными вне зависимости от того, где они хранятся.

Система SQL Server 2008 позволяет обращаться к данным из любого приложения, разработанного с применением технологий Microsoft .NET и Visual Studio. Сотрудники, отвечающие за сбор и анализ информации, могут работать с данными, не покидая привычных приложений, которыми они пользуются каждый день, например приложений выпуска 2007 системы Microsoft Office. SQL Server 2008 позволяет создать надежную, производительную, интеллектуальную платформу, отвечающую всем требованиям по работе с данными.

Динамическое шифрование всей базы данных, файлов данных и файлов журналов не потребует никаких изменений в имеющихся приложениях. Преимущества очевидны: поиск по зашифрованным данным как по диапазонам, так и нечетким поиском; поиск защищенных данных у неавторизованных пользователей; использование шифрования без каких-либо изменений в существующих приложениях.

Регулятор ресурсов позволяет обеспечивать стабильное и предсказуемое время отклика на запросы конечных пользователей. С его помощью можно определять пределы использования ресурсов и устанавливать приоритеты для разных рабочих нагрузок. Это обеспечивает ровную производительность одновременно выполняемых задач.

Возможность зафиксировать планы выполнения запросов позволяет достичь большей ровности и предсказуемости их выполнения. Стабильные планы запросов не потеряются при замене оборудования, обновлениях сервера и промышленном развертывании.

Данные хранятся более эффективно и занимают меньше места в хранилище.

Сжатие также обеспечивает заметный прирост производительности на задачах, связанных с большим количеством операций ввода-вывода, как, например, при работе с хранилищами данных.

SQL Server 2008 привносит значительные улучшения в жизненный цикл обслуживания SQL Server. Переработке подверглась архитектура установки, настройки и конфигурирования. Теперь физическая установка на оборудование отделена от конфигурирования, что позволяет самим предприятиям и их компаниям-партнерам создавать рекомендованные к установке конфигурации.

Запросы к данным можно осуществлять с помощью управляемых языков программирования, таких как C# или VB.NET, а не командами SQL. Строго типизированные, ориентированные на наборы данных запросы, написанные на языках .NET, могут легко выполняться к ADO.NET (LINQ to SQL), ADO.NET DataSets (LINQ to DataSets), платформе ADO.NET Entity Framework (LINQ to Entities) и к поставщику Entity Data Service Mapping. Новый поставщик LINQ to SQL позволяет через LINQ напрямую обращаться к таблицам и столбцам SQL Server 2008.

Слой объектных служб ADO.NET позволяет осуществлять материализацию, отслеживание изменений и сохранение данных как объектов CLR. При использовании платформы ADO.NET можно программировать базу данных в терминах таких объектов, управляемых ADO.NET. SQL Server 2008 поддерживает эту технологию более эффективно, что повышает производительность и упрощает разработку.

## 3.4 Разработка моделей представления программного модуля

Для описания представлений системы необходимо построить UML диаграммы. Построим диаграммы:

* структурная схема программного модуля на уровне сборок;
* диаграмму классов;
* диаграмму деятельности;
* диаграмму компонентов.

Структурная схема программного модуля на уровне сборок представлена на рисунке 3.2.

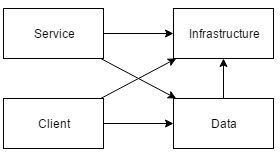


Рисунок 3.2 − Структурная схема программного модуля на уровне сборок

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, использующиеся в обеих частях приложения, а именно модели данных, передающихся между клиентом и сервером.

Диаграммы классов представлены на рисунках 3.3-3.6.

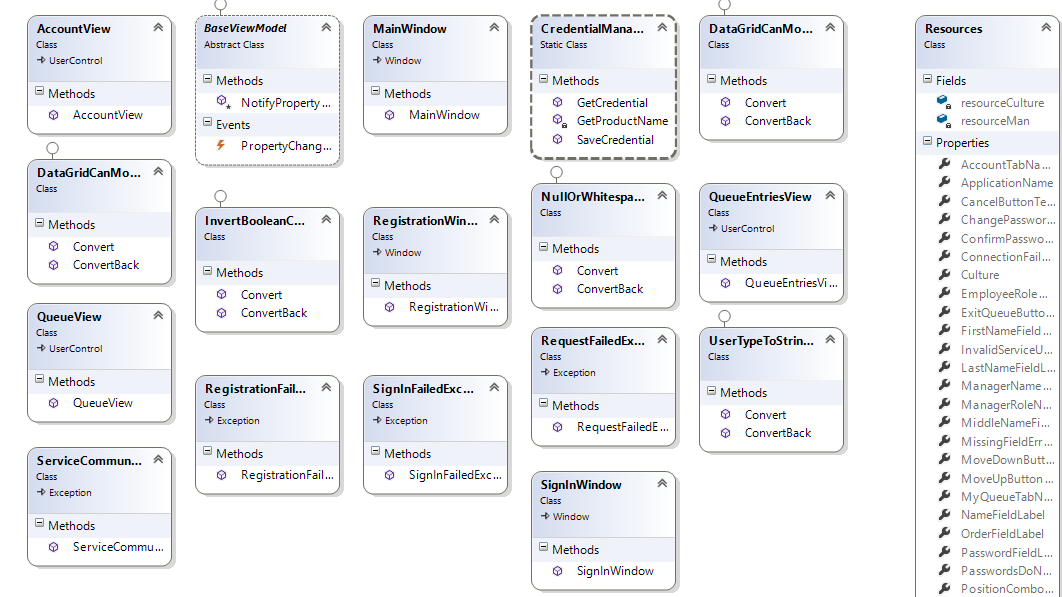


Рисунок 3.3 − Диаграмма классов клиентского приложение Client

Данная диаграмма содержит следующие классы и интерфейсы клиентского приложение на основе WPF, предоставляет пользовательский интерфейс для данных, общается с сервером по протоколу HTTP системы:

* класс AccountView – аккаунты клиентов;
* класс QueueView – просмотр очереди;
* класс ServiceCommunicationException – исключения, относящиеся к взаимодействию с сервером;
* класс RegistrationFailedException – исключения, происходящие в процессе регистрации 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.

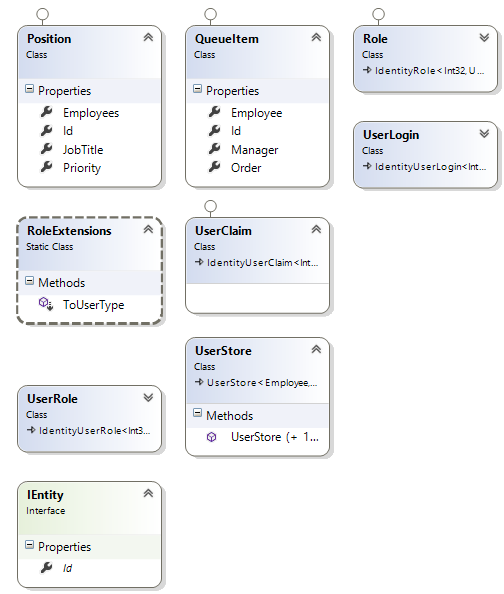


Рисунок 3.4 − Диаграмма классов Data

Данная диаграмма классов 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.

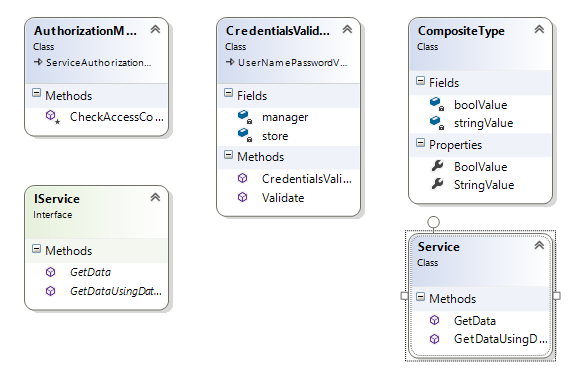


Рисунок 3.5 − Диаграмма классов Services

Данная диаграмма содержит классы и интерфейсы серверной части приложения. Представляет собой RESTful сервис на основе ASP.NET Web API, обрабатывает запросы от клиентов:

* 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.

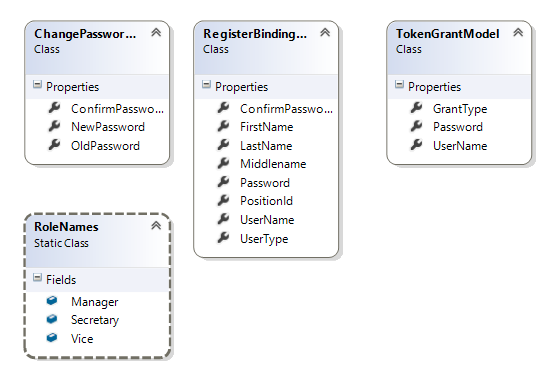


Рисунок 3.6 − Диаграмма классов Infrastructure

Диаграмма классов содержит компоненты, использующиеся в обоих частях приложения, а именно модели данных, передающихся между клиентом и сервером.

* 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.

Диаграмма деятельности (Activity Diagram) представлены на рисунках 3.7.

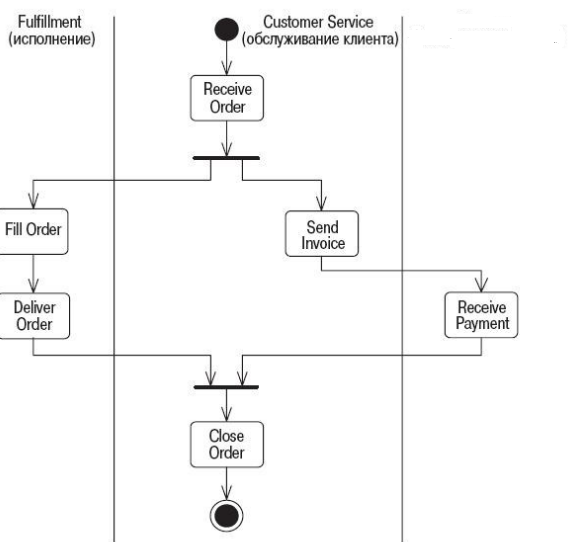


Рисунок 3.7 − Диаграмма деятельности «Обслуживание клиентов»

Диаграмма компонентов представлена на рисунке 3.8. На данной диаграмме отображаются компоненты и модули разрабатываемой системы.

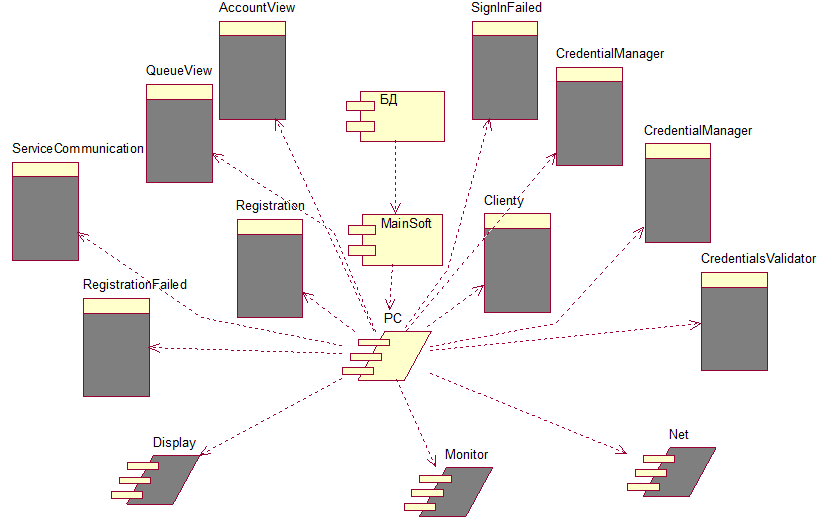


Рисунок 3.8 − Диаграмма компонентов

После проектирования структуры приложения и построения основных моделей приложения, касающихся его архитектуры, можно переходить к следующему этапу – описанию основных алгоритмов программных модулей.

## 3.5 Организация технологии сбора, передачи, обработки информации

Сбор информации осуществляется следующим образом: информация вносится сразу после внедрения проекта и в будущем заносится по мере поступления новых сведений. Оперативная информация вносится в базу данных по мере поступления соответствующих документов и сведений.

Алгоритмы работы программы являются стандартными алгоритмами работы с базой данных. В основном все алгоритмы работы связаны с вводом данных от пользователя, проверке введенной информации на предмет нарушения целостности данных и занесение введенной информации в саму базу, если введенные сведения не нарушают целостности.

Алгоритм информационного взаимодействия клиентской и серверной части (в данном случае при работе с базой данных) представлен на рисунке 3.9.



Рисунок 3.9 − Схема информационного взаимодействия приложения с базой данных

## 3.6 Разработка структурной схемы программного модуля

Интерфейс приложения можно отнести к стандартному интерфейсу MS Windows. Доступ ко всем экранным формам приложения осуществляется как из главной формы приложения, так и из других форм и главного меню программы.

В процессе проектирования разработана двухуровневая структура пользовательского интерфейса системы. На первом уровне расположена форма «Сотрудники». На втором уровне расположена форма «Очередь».

Формы персонала предприятия «Клиенты», «Руководители», «Менеджеры», имеют свой особый приоритет очереди.

Для определения приоритета очереди, имеется форма «Администрирование очереди».

Очередь и учетные записи содержатся в базе данных.

Для отчетов и получения статистических данных необходимо выбрать соответствующий пункт «Статистика».

В процессе работы использованы следующие принципы организации пользовательского интерфейса [14]:

– естественность и интуитивность (отсутствие у пользователя сложностей в поиске необходимых директив или элементов интерфейса для управления процессом решения поставленной задачи);

– непротиворечивость;

– отсутствие избыточности (должен обеспечиваться ввод минимально необходимого объёма данных для решения задач или управления системой; не должен требоваться повторный ввод данных или ввод вычисляемых данных);

– структурирование информации на экране (количество элементов и данных на экране должно быть минимальным; информация на экране должна быть сгруппирована и упорядочена с помощью цветового кодирования, рамок, негативного изображения или других методов привлечения внимания);

– выделение элементов интерфейса яркостью и цветом;

– стандартизация (однотипные данные должны размещаться в одной и той же области экрана);

* информация, на которую следует немедленно обратить внимание, должна быть выделена цветом или яркостью, и всегда отображаться в видном месте, чтобы захватить внимание пользователя.

Для уменьшения ошибок при вводе информации в некоторых полях задаются условия на значение. В самом простом случае условие на значение должно гарантировать, что из-за ошибки ввода в числовом поле не окажутся буквенные символы.

Другие условия могут определять область или диапазоны допустимых значений. Заданное условие на значение всегда будет проверяться при вводе или изменении значения поля в таблице. Кроме того, для уменьшения ошибок при вводе данных используется маска ввода.

Маска ввода удобна при использовании полей, размер и смысловая нагрузка которых заранее известна.

Для взаимодействия с пользователем программы предполагается использование меню, подсказок, полей, отвечающих за ввод информации, а также кнопок, результатом нажатия на которые будет отображение того или иного запроса к базе данных.

Программа является событийно-управляемой. Структура программы ветвящаяся, переход по каждой из ветвей инициируется событием.

Структурная схема системы приведена на рисунке 3.10.



Рисунок 3.10 − Структурная схема программы

# 4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

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

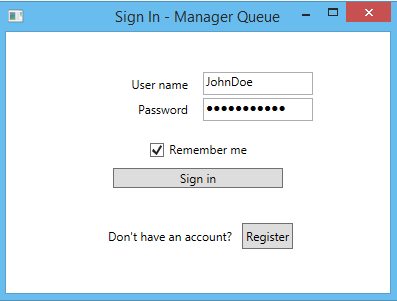


Рисунок 5.1 – Авторизация в системе

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.

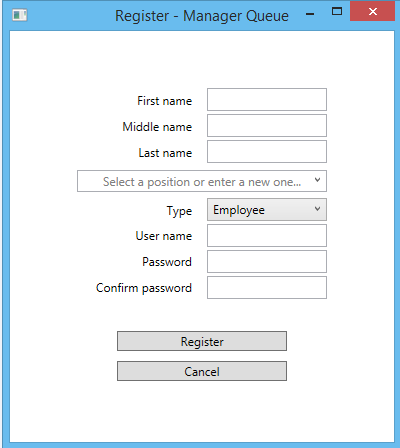


Рисунок 5.2 – Регистрация в системе

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

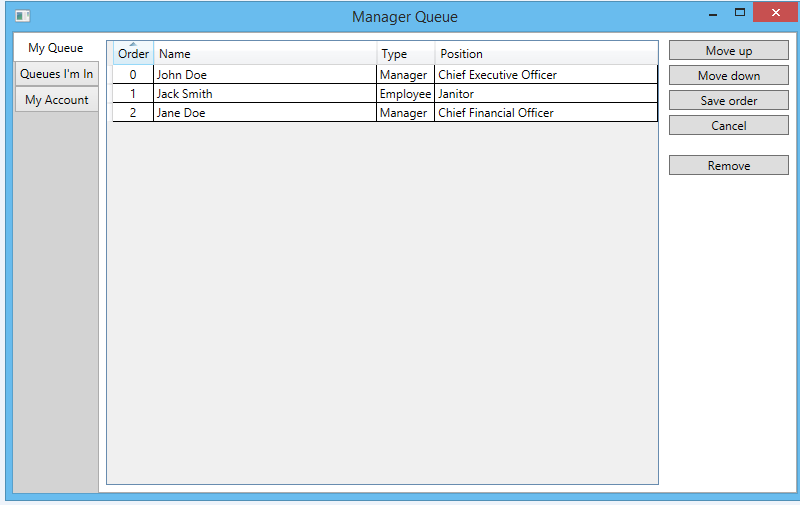


Рисунок 5.3 – Управление очередью

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

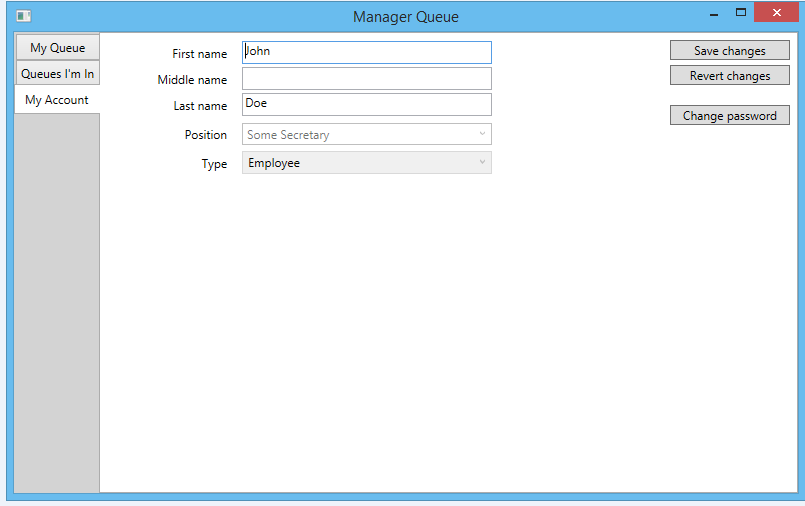


Рисунок 5.4 – Управление учетной записью

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

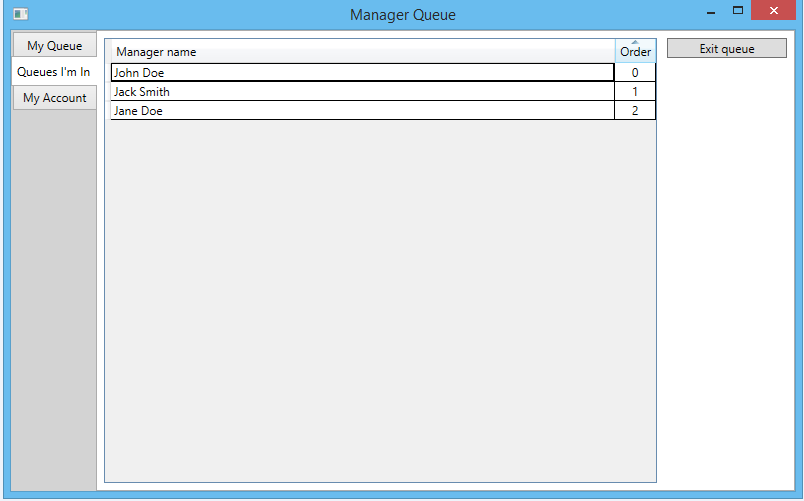


Рисунок 5.5 – Просмотр очередей

# 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, Кт=0,8.

Таблица 6.1 – Исходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Обозначения | Единицы измерения | Количество |
| Коэффициент новизны | Кн | единиц | 0,7 |
| Группа сложности | - | единиц | 3 |
| Дополнительный коэффициент сложности | Кс | единиц | 0,06 |
| Поправочный коэффициент, учитывающий использование типовых программ | Кт | единиц | 0,8 |
| Установленная плановая продолжительность разработки | Тр | лет | 0,33 |
| Продолжительность рабочего дня | Тч | ч | 8 |
| Тарифная ставка 1-го разряда | Тм1 | руб. | 31 |
| Коэффициент премирования | Кп | единиц | 1,2 |
| Норматив дополнительной заработной платы исполнителей | Нд | % | 20 |
| Отчисления в фонд социальной защиты населения | Зсз | % | 34 |
| Отчисления в Белгосстрах | Нне | % | 0,6 |
| Налог на прибыль | Нп | % | 18 |
| Норматив расходов на сопровождение и адаптацию | Нс | % | 20 |
| Налог на добавленную стоимость | Ндс | % | 20 |
| Уровень рентабельности ПС | УРПi | % | 15 |

# 6.2. Расчет затрат и отпускной цены программного средства

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

где Vy – уточненный объем ПО;

Vyi – уточненный объем отдельной функции ПО (LОС);

n– общее число функций.

Таблица 6.2 – Перечень и объем функций программного модуля

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание) функции | Объем функции (LOC) | |
| Vi | Vyi |
| 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 (С#), программное обеспечение функционального назначения.

Vу = 10000 LOC.

6.2.2. Трудоемкость разработки ПО

Общая трудоемкость небольшого проекта высчитывается по формуле 6.2:

Tо = Tн⋅Kс⋅Kт⋅Kн = 228⋅1,06⋅0,8⋅0,7 = 136 чел./дн. (6.2)

где Tо – общая трудоемкость;

Tн – нормативная трудоемкость ПО, равная228 чел./дн.;

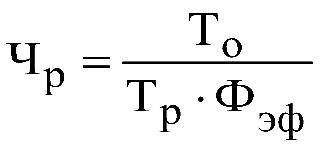
Kс– дополнительный коэффициент сложности;

Kт– поправочный коэффициент, учитывающий использование типовых программ;

Kн –коэффициент новизны.

4.2.3. Численность исполнителей проекта

Численность исполнителей проекта:

, (6.3)

где Чр – численность исполнителей проекта;

Tр – срок разработки проекта (лет);

Фэф – эффективный фонд времени работы одного работника.

Эффективный фонд времени работы одного работника:

Фэф = Дг−Дп−Дв−До = 236 дн., (6.4)

где Фэф – эффективный фонд времени работы одного работника;

Дг – количество дней в году;

Дп – количество праздничных дней в году;

Дв – количество выходных дней в году;

До – количество дней отпуска.

# 6.3. Расчет сметы затрат и цены заказного ПО

6.3.1 Основная заработная плата () исполнителей на конкретное ПО рассчитывается по формуле:

где n − количество исполнителей, занятых разработкой конкретного ПО;

Зci −среднедневная заработная плата i-го исполнителя (д.е.);

Фрi − плановый фонд рабочего времени i-го исполнителя (дн.);

К  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%), Зо | - | - | - | 5472 |

6.3.2. Дополнительная заработная плата исполнителей проекта определяется по формуле:

где Зд – дополнительная заработная плата;

НД – норматив дополнительной заработной платы (20%).

Дополнительная заработная плата составит:

Зд = 5472·20/100 = 1094,4 руб.

6.3.3. Отчисления в фонд социальной защиты населения и на обязательное страхование (Зс) определяются в соответствии с действующими законодательными актами по формуле 1.7.

где Нсз – 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:

Зсз = (5472+1094,4)·34,6/100 = 2271,98 руб.

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

где Рм – расходы по статье «Машинное время»;

Цм – цена одного машино-часа;

Vу – объём кода;

Нмв – 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:

Рм = 1,4·10000/100·12 = 1680 руб.

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

где Нрн – норматив накладных расходов (50%).

Накладные расходы составят:

Рн = 5472·0.5 = 2736 руб.

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

, (6.10)

Рассчитаем сумму расходов по всем статьям сметы:

Сn = 5472+1094,4+2271,98+1680+2736 = 13254,38 руб.

Прибыль по создаваемому ПС рассчитывается по формуле:

где По – прибыль от реализации ПС заказчику (руб.);

Урп – уровень рентабельности ПС (15%);

Сп – себестоимость ПС (руб.).

Рассчитаем прибыль по создаваемому ПС:

По =13254,38·15/100 = 1988,16 руб.

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

, (6.12)

Рассчитаем прогнозируемую цену ПС без налогов:

Цп = 13254,38 + 1988,16 = 15242,54 руб.

Налог на добавленную стоимость (НДСi):

где Hдc− норматив НДС (20%).

Рассчитаем НДС:

НДС= 15242,54 ∙ 20 / 100 = 3048,51 руб.

Прогнозируемая отпускная цена (Цo):

, (6.14)

Рассчитаем прогнозируемую отпускную цену

Цo = 15242,54 + 3048,51 = 18291,05 руб.

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

где Нс – норматив расходов на сопровождение и адаптацию ПС;

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

Рассчитаем затраты на сопровождение ПС:

Рс = 13254,38 · 20 / 100 = 2650,88 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 | Обозна-  чения | Единицы измерения | Значение показателя | | Наименование источника информации |
| в базовом варианте | в новом варианте |
| 1. Капитальные вложения, включая затраты пользователя на приобретение | Kпp | руб. |  | 18291,05 | Договор заказчика с разработчиком |
| 2. Затраты на сопровождение ПО | Кс | 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 | Kнз |  | 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 | Tc1,Тс2 | Человеко-часов | 2,0 | 1,0 | Рассчитывается по данным пользователя |
| 11. Количество часов работы ты в день | Tч | ч | 8 | 8 | Принято для расчета |
| 12. Ставка налога на прибыль | Hп | % |  | 18 |  |

Экономия затрат на заработную платув расчете на 1 задачу (Cзe):

где Зcм − среднемесячная заработная плата одного программиста (руб.);

Tc1, Tc2 − снижение трудоемкости работ в расчете на 1 задачу (человеко-часов);

Tч − количество часов работы в день (ч);

Дp − среднемесячное количество рабочих дней.

Рассчитаем экономию затрат на заработную платув расчете на 1 задачу:

Сзе = 430(2-1)/(821.5) = 2,5 руб

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

Сз = СзеА2, (6.17)

где Сз− экономия заработной платы;

А2− количество типовых задач, решаемых за год (задач).

Рассчитаем экономию заработной платы при использовании нового ПО:

Сз = 2,51806 = 4515 руб.

Экономия с учетом начисления на зарплату (Сн):

Cн = Cзּ Kнз, (6.18)

Рассчитаем экономию с учетом начисления на зарплату:

Cн= 4515**ּ** 1,5 *=* 6772,5 руб.

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

где Дрг – плановый фонд работы сервиса (дней).

Сс = 50-10) / 60 = 7543,33 руб.

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

Co = Cн + Cc, (6.20)

Co = 6772,5 +7543,33 = 14315,83 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 (ΔПЧ), которая определяется по формуле:

где Hп − ставка налога на прибыль (18%).

ΔПЧ = 14315,83 – (14315,83 18) / 100 = 11738,98 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 |
| *Результаты* |  |  |  |  |  |
| Прирост прибыли за счет экономии затрат (Пч) | руб. |  | 11738,98 | 11738,98 | 11738,98 |
| То же с учетом фактора времени | руб. |  | 10208,22 | 8875,85 | 7718,38 |
| *Затраты* |  |  |  |  |  |
| 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Экономический эффект* |  |  |  |  |  |
| 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 приступать 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., Фомин В.И. Информационные системы в экономике: основы информационного бизнеса. Учебное пособие СПб.: СПБГИЭА, 1999.
2. Смирнова, Г.Н. Проектирование экономических информационных систем: Учеб. для вузов / Г.Н. Смирнова, А.А. Сорокин, Ю.Ф. Тельнов; Под ред. Ю.Ф. Тельнова. -М. : Финансы и статистика, 2002. - 512 с. : ил.
3. Мишенин, А.И. Теория экономических информационных систем: Учеб. для вузов / А.И. Мишенин.- 4-е изд., доп. и перераб. -М. : Финансы и статистика, 2001. - 240 с. : ил.
4. Липаев В.В Управление разработкой программных средств. Методы, стандарты, технология. – М.: Финансы и статистика, 2003.
5. Хотяшов Э.Н. Проектирование машинной обработки экономической информации. М.:Финансы и статистика,2001.-246 с.
6. Оскерко В.С., Пунчик З.В. Практикум по технологиям баз данных. – Мн.: «БГЭУ», 2004. – 170 с.
7. Дейт К. Введение в системы баз данных/Пер. с англ. М.: Наука, 2003. 463 с.
8. Грох М., Ирвин М., Пауэлл Г., Праг К., Рирдон Д., Стокман Д. Microsoft Office Access 2007: Библия пользователя. – М.: «Диалектика», 2009. – 1023 с.
9. Архангельский А.Я. Программирование в Delphi 7. – М.: Бином, 2004. −118 с.
10. Вендров А.М. CASE−технологии. Современные методы и средства проектирования информационных систем. − М.: Финансы и статистика, 2000.
11. Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD. М.: Русская редакция, 2000.
12. Когаловский М.Р. Энциклопедия технологий баз данных. - М.: Финансы и статистика, 2002.
13. Фуфаев Д.Э., Фуфаев Э.В. Базы данных. – М.: “Академия”, 2005. – 320 с.

# 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.ManagerId == id).ToArrayAsync());

}

// PUT: api/Queue/5

public async Task<IHttpActionResult> Put(int queueId, [FromBody]IEnumerable<QueueItem> queueItems)

{

var queueIds = queueItems.Select(queueItem => queueItem.ManagerId).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.Order).ToArray();

var queueItem = new QueueItem()

{

ManagerId = queueManager.Id,

EmployeeId = entrant.Id

};

var entrantType = await UserManager.GetUserType(entrant.Id);

if (entrantType == UserType.Employee || entrantType == UserType.Secretary || 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) == UserType.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().GetUserManager<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(ChangePasswordBindingModel model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

IdentityResult result = await UserManager.ChangePasswordAsync(Convert.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(SetPasswordBindingModel 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<TokenEndpointErrorResponse>();

}

catch (Exception)

{

}

if (error != null && !string.IsNullOrEmpty(error.ErrorDescription))

throw new SignInFailedException(error.ErrorDescription);

response.EnsureSuccessStatusCode();

}

try

{

var tokenInfo = await response.Content.ReadAsAsync<TokenEndpointGrantResponse>();

if (string.IsNullOrEmpty(tokenInfo.AccessToken))

throw new ServiceCommunicationException();

TokenInfo = tokenInfo;

}

catch (Exception exception)

{

throw new ServiceCommunicationException(Resources.SignInErrorExceptionMessage, exception);

}

HttpClient.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", TokenInfo.AccessToken);

try

{

CurrentUser = await Get<Employee>(ResourceUri.Account.Base);

}

catch (HttpRequestException exception)

{

throw new RequestFailedException(Resources.UserFetchFailedExceptionMessage, 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.ConnectionFailedExceptionMessage, 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.ConnectionFailedExceptionMessage, 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("userName")]

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.PersistanceType = PersistanceType.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.SecretaryRoleName),

new Tuple<UserType, string>(UserType.Vice, Resources.ViceManagerRoleName),

new Tuple<UserType, string>(UserType.Manager, Resources.ManagerRoleName)

};

selectedUserType = userTypes.First();

}

private async Task Register()

{

ErrorMessage = null;

if (!ValidateForm())

return;

IsBusy = true;

try

{

var position = SelectedPosition;

if (position == null && !string.IsNullOrEmpty(PositionComboBoxText))

{

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 RegistrationFinishedEventArgs(true));

}

catch (Exception exception)

{

ErrorMessage = exception.Message;

}

}

finally

{

IsBusy = false;

}

}

private async Task Cancel()

{

RegistrationFinished?.Invoke(this, new RegistrationFinishedEventArgs(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 SignInSucceded;

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.SecurePassword);

IsBusy = false;

ShowMainWindow();

SignInSucceded?.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;

}

}

}