Государственное профессиональное образовательное учреждение
Ярославской области
«Ярославский автомеханический колледж»

ДОПУСКАЮ К ЗАЩИТЕ:

Зам. директора по учебной работе
_____ А.И. Ёлкин
«___» _____20__г.

ДИПЛОМНЫЙ ПРОЕКТ

ДП 09.02.03.19.0616.000 ПЗ

Тема задания: Разработка приложения ведения заметок для предприятия

Разработал:

Круглов А.Н.
Подпись Фамилия, инициалы

« » 20 г

Руководитель проекта:

Маслова Е.А.
Подпись Фамилия, инициалы

« » 20 г

Нормоконтролер:

Маслова Е.А.
Подпись Фамилия, инициалы

« » 20 г

Председатель ПЦК:

Скворцова С.А.
Подпись Фамилия, инициалы

« » 20 г

Зав. отделением:

Байдина Е.В.
Подпись Фамилия, инициалы

« » 20 г

2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Любой пользователь регулярно сталкивается с необходимостью сохранения разнообразной информации. Зачастую такая информация представляет собой не объемные материалы, а небольшой текст или изображение с какими-то важными данными. Например, это могут быть заметки, сделанные в ходе совещания или изучения важного документа, фрагмент веб-страницы с полезной информацией, телефонный номер, по которому нужно будет пару раз позвонить, адрес страницы в Интернете, которую хотелось бы посетить, и т.п.

Многие пользователи сохраняют подобные текстовые заметки в текстовых файлах, графику сбрасывают в папку «Мои рисунки», а некоторые данные и вовсе записывают на клочках бумаги. Однако это не лучшее решение, поскольку при накоплении материалов найти что-то нужное становится всё сложнее. Гораздо удобнее для таких случаев держать под рукой электронную записную книжку, которая позволит быстро сохранять необходимую информацию и обеспечит оперативный доступ к интересующим данным. Теоретически в качестве электронной записной книжки могут служить самые разные решения — от обычных «липких листочков» на рабочем столе до тяжеловесных PIM (персональных информационных менеджеров). Правда стикеры хороши только для ведения очень коротких текстовых заметок и напоминаний, а PIM, наоборот, отличаются весьма широкой функциональностью и больше подходят для планирования и управления задачами и контактами, но не особенно удобны для ведения заметок. Более предпочтительным вариантом является менеджер заметок, который может представлять собой программное решение либо сервис. Несколько таких приложений и сервисов мы и рассмотрим в данной статье.

На рынке предлагается немало интересных программ для ведения заметок. Наиболее солидным, функциональным, но, увы, слишком дорогим для большинства пользователей решением является приложение Microsoft OneNote.

РАЗДЕЛ 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ предметной области

Заметка — это информационный жанр журналистики, подразумевающий краткое сообщение, в котором излагается факт или ставится конкретный вопрос. Тут нет ответа на вопрос и оценки данному событию. В отличие от новости заметка содержит меньше подробностей, комментариев и дополнительной информации.

Признаки заметки:

- Небольшой объем (около 2 000 знаков).
- Материал подается в виде ответов на вопросы (что, где и когда произошло). Это 3 основных вопроса, на которые нужно ответить в заметке. Обратите внимание, здесь нет вопроса почему это произошло или какие будут последствия.
- Идет только событие без выводов и оценок. Не важно, что думает сам автор.
- Более подробно то или иное событие освещается в других жанрах. Например, в репортаже, комментарии или статьи.
- Достаточно жесткая структура.

Структура заметки

Вот как выглядит структура заметки:

- Заголовок.
- Подзаголовок (не всегда может встречаться).
- Лид (введение) — первый абзац с кратким содержанием текста.
- Основной текст.

Виды заметки

Давайте рассмотрим основные виды заметок:

1. Информационная заметка — стандартный вид данного жанра, в котором излагаются только факты и сведения о событии. Часто используется в событийном поджанре.

2. Хроникальная заметка — используется для описания различных историй. В определенной последовательности дается хронология всех событий. К примеру, в поджанре мини-история такой вид часто может использоваться автором.

3. Расширенная заметка — здесь подается достаточно больше информации, чем в остальных видах. К примеру, если это событийный поджанр, то тут будет достаточно больше фактов и информации по данной новости. Если говорить о других поджанрах, то тут дополнительно может быть какое-то небольшое интервью или мини-вывод.

Поджанры

Ниже идут поджанры, которые могут использоваться разными видами заметок:

1. Событийная заметка

В событийной (новостной) заметке предметом отображения являются различные события или положения дел. Здесь наблюдается строгий ответ на то, что, где и когда произошло.

Новостная заметка — это своего рода краткое информационное сообщение. Стандартный объем 4 — 5 предложений. Как правило, пишется в официальном стиле.

Первое предложение - топ-лайн - должно отвечать на вопрос, о чем это. Не стоит перегружать сообщение именами собственными и терминологией. Не используйте более двух цифр в предложении и не округляйте их если это касается жертв. В конце можно использовать информацию о прошлых и грядущих событиях.

2. Анонс

Анонс образует простые заметки, представляющие собой превентивные сообщения о будущих всевозможных культурных мероприятиях, выставках, концертах и прочее.

3. Аннотация

Аннотация — это основной и первоначальный источник информации о статье или книге. И чем грамотнее и точнее вы напишите аннотацию, тем больше вероятности, что кто-нибудь захочет прочитать весь текст статьи.

Вы можете потратить много времени на проведение исследований и интерпретацию результатов. Но указав в аннотации 2 — 3 общих предложения, есть риск, что статью так никто и не прочтет.

Предметом отображения в данном жанре журналистики выступает определенное, уже состоявшееся информационное явление. Прежде всего, это книги и статьи.

Цель заключается не столько, чтобы известить аудиторию о существовании какого-либо нового издания, сколько в том, чтобы описать кратко его качества.

Аннотация включает в себя характеристику основной темы, проблемы объекта, цели исследования, основные методы, результаты исследования и выводы. Здесь указывают, что нового несет в себе материал по сравнению с другими.

Поэтому информация должна быть информативной, оригинальной, содержательной, структурированной и компактной. Объем как правило, от 150 до 250 слов.

Важно знать, что аннотация готовится уже после написания основного текста статьи или книги.

Удобнее всего писать структурированную аннотацию по структурированной статье. При этом нужно выбирать из каждого раздела статьи только самые важные сведения. Они в совокупности должны составлять полное представление об информации, изложенной в статье.

В аннотацию не включают ссылки на источники из полного текста. Также не желательно использование аббревиатур и сокращений. Но если вам просто необходимо использовать какую-то аббревиатуру, тогда не забудьте раскрыть ее смысл в самой аннотации.

4. Мини-рецензия

Мини-рецензия представляет собой оценочную, иногда рекламную заметку. Ее предметом выступает какое-то информационное явление (книга, кинофильм, пьеса и так далее).

Цель публикации данного типа заключается в том, чтобы сообщить читателю о впечатлении, полученном ее автором в ходе знакомства с отображаемым предметом. При этом мнение автора не обосновывается какими-то доказательствами, а представляет собой простое изложение эмоций.

Здесь не важен анализ. Здесь важны только отношения автора, его мнение и оценка. Повествование обычно идет от первого лица.

5. Блиц-портрет

Эта разновидность заметки, где можно найти краткие сведения о человеке. Тут дается первичное представление о личности.

Чаще всего блиц-портрет не является каким-то отдельным жанром. Он сопровождает большой материал, где выдается отдельной отмеченной врезкой. Например, краткая информация о том, где родился человек, чем занимался и так далее.

6. Мини-обозрение

Здесь сравнивается какая-то совокупность событий. Критерием этого сравнения выступает какой-то общий признак.

Например, эти события могли произойти в одно и тоже время. Или же они могли быть посвящены одной определенной тематике.

7. Мини-история

В основе мини-истории лежит определенная интрига. Очень часто такие короткие исторические заметки публикуют женские журналы. В них рассказывают о профессиональных, любовных или семейных приключениях.

8. Мини-совет

Основным содержанием мини-совета является программная информация. Это то, как нужно что-то сделать. Своего рода, алгоритм действий для достижения результата.

Приемы компоновки информации

Как можно из большой новости или какой-то информации сделать компактную заметку? Для начала, это удаление вводных слов и конструкций.

Также нужно убирать оценочные выражения. Например, авторитетный, новейший, суперкачественный и так далее. Помним, что оценка автора в данном жанре никому не интересна.

Еще удаляйте бытовые штампы. Например, на каждом шагу, волшебным образом, в конце концов. Это также не несет в себе никакого смысла и информативности.

Также удаляйте штампы бюрократизма. Такие слова очень трудны для восприятия. Они никак не сделают вашу заметку более легкой.

Компьютерные программы

Компьютерная программа — это последовательность инструкций, которая предназначена для исполнения вычислительной машиной. Образ программы, чаще всего, хранится в памяти машины (например, на диске) как исполняемый модуль (один или несколько файлов). Из образа на диске с помощью специального программного загрузчика может быть построена исполняемая программа уже в оперативной памяти машины.

Термин «компьютерная программа» в зависимости от своего контекста, может применяться также к исходным текстам (или кодам) программы. Их примеры могут быть просмотрены в специальных каталогах исходников. Вместе с правилами и процедурами, а также с документацией по функционированию программных систем обработки данных, компьютерные программы составляют понятие программного обеспечения.

В системном программировании имеет место более формальное определение программы как машинных кодов и данных, загруженных в

оперативную память компьютера, и исполняемых процессором машины для достижения поставленной цели. В этом определении подчеркиваются две особенности компьютерной программы: нахождение ее в памяти и исполнение процессором машины.

Процесс создания компьютерной программы называется «программированием», а люди, занимающиеся этим видом деятельности, называются программистами. При разработке компьютерных программ в них довольно часто возникают ошибки. Считается, что в программе содержатся ошибки, если для каких-то данных программа дает неправильные результаты, сбои или отказы. Если программа выдает правильные результаты обработки для всех возможных входных данных, то можно считать, что она не содержит ошибок.

Процесс поиска ошибок в программах и их исправления называется отладкой программ. Обычно, заранее неизвестно, сколько ошибок содержит программа. По этой причине заранее неизвестна и продолжительность отладки программ.

Запись исходных текстов компьютерных программ при помощи специальных языков программирования (ЯП) облегчает человеку понимание и редактирование программ. Этому, также, помогают комментарии, допускаемые синтаксисом большинства языков программирования. Для выполнения программы на компьютере ее готовый исходный текст преобразуется (компилируется или интерпретируется) в машинный код, исполняемый процессором.

Программы с исходными текстами, доступными для прочтения и изменения любым желающим, называются открытыми программами. Любая компьютерная программа является объектом авторского права. Авторы или собственники программ имеют право ограничивать и даже полностью закрывать доступ к их исходным текстам, которые являются интеллектуальной собственностью правообладателей.

Некоторые языки программирования (интерпретируемые) позволяют обойтись без предварительной компиляции написанных на них программ, и специальные программы-интерпретаторы переводят такие программы в машинный код уже во время исполнения программы. Этот процесс называется интерпретированием или динамической компиляцией. Он позволяет улучшить переносимость программ между различными программными и аппаратными платформами. Интерпретируемые программы часто называются сценариями или скриптами.

В большинстве распространенных ЯП исходные тексты программ состоят из списков инструкций, описывающих заложенный в программе алгоритм. Такой подход называется императивным. Но применяются и иные методологии программирования. Так, например, в декларативном программировании описываются исходные и требуемые характеристики обрабатываемых данных, а выбор подходящего алгоритма решения описанной задачи поручается специализированной программе-интерпретатору. Применяются также логическое и функциональное программирование.

1.2 Необходимость разработки программного продукта

Основной целью разработки данного программного обеспечения является помощь пользователям в удобном хранении и передачи заметок.

В связи с этим программа должна отвечать следующим задачам:

- Простой и понятный интерфейс.
- Простота в использовании.
- Надежность.
- Передача и отображение данных осуществляется онлайн.

Главная цель программы - стать инструментом редактирования, добавления, удаления, вывода информации о заметках. Определение целей создания программы, круга функций, которые она должна выполнять, необходимо для лучшего понимания истинных задач, которые собираются решать с помощью программы его владельцы. Правильно, конкретно поставленная цель позволяет сделать программу эффективным инструментом для пользователя. От продуманности формулировки этой задачи зависит эффективность работы программы в долгосрочной перспективе. Цель моего дипломного проекта является разработка и создание программы для широкого круга пользователей.

1.3/ Обоснование выбора программного обеспечения

C# – простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

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

Вот лишь несколько функций языка C#, обеспечивающих надёжность и устойчивость приложений: сборка мусора автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами; обработка исключений даёт структурированный и расширяемый способ выявлять и обрабатывать ошибки; строгая типизация языка не позволяет обращаться к неинициализированным переменным, выходить за пределы массива или выполнять неконтролируемое приведение типов.

В C# существует единая система типов. Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Таким образом, все типы используют общий набор операций, и значения любого типа можно хранить, передавать и обрабатывать схожим образом. Кроме того, C# поддерживает пользовательские ссылочные типы и типы

значений, позволяя как динамически выделять память для объектов, так и хранить упрощённые структуры в стеке.

Чтобы обеспечить совместимость программ и библиотек C# при дальнейшем развитии, при разработке C# много внимания было уделено управлению версиями. Многие языки программирования обходят вниманием этот вопрос, и в результате программы на этих языках ломаются чаще, чем хотелось бы, при выходе новых версий зависимых библиотек. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Развитие языка

Далёкие предки C# появились ещё в 60-х годах. Все началось с появления языка B, который в 1969 году был создан коллективом разработчиков из Технологического института Массачусетса (MIT). Главным автором B является Кен Томпсон. Тогда команда работала над операционной системой UNIX. Уже существовавший язык PL/I, применявшийся в то время для мэйнфреймов производства компании IBM, был достаточно громоздким и меньше подходил для поставленной задачи. Поэтому учёные решили создать новый язык, который и получил название B. Он является типичным представителем ранних императивных языков программирования.

После B, как это ни странно, последовал C, который был изобретён в 1972 году. Основой для нового языка послужил сам B.

Создателями C были Кен Томпсон и Денис Ритчи, которые работали в исследовательской лаборатории компании AT&T (AT&T Bell Telephone Laboratories). В 1971 году Ритчи начал создавать расширенную версию B. Сначала он назвал её NB (New B), но когда язык стал сильно отличаться от B, название сменили на C. B расширился за счёт явного использования типов, структур и ряда новых операций.

По поводу возникновения языка Си Питер Мойлан в своей книге «The case against C» писал: «Нужен был язык, способный обойти некоторые жёсткие

правила, встроенные в большинство языков высокого уровня и обеспечивающие их надёжность. Нужен был такой язык, который позволил бы делать то, что до него можно было реализовать только на ассемблере или на уровне машинного кода».

В 1984 году Бьярне Страуструп (Bell Labs) выступил с проектом языка C++. Когда Страуструп занимался исследованиями в фирме, ему потребовалось написать несколько имитационных программ для моделирования распределённых вычислений. SIMULA-67 — объектно-ориентированный язык – мог бы стать идеальным инструментом для решения подобных задач, если бы не его сравнительно низкая скорость выполнения программ.

Так был создан язык программирования C++, первоначально получивший название «Си с классами» (C with classes). Название «C++» придумал Рик Мэсчитти. "++" — это оператор инкремента в C, который как бы намекает на то, что язык C++, нечто больше, чем просто C.

Microsoft решила отметить Миллениум выпуском новых программных продуктов. К 2000 году компания подготовила промышленные версии новых компонентных технологий и решений в области обмена сообщениями и данными, а также создания Internet-приложений (COM+, ASP+, ADO+, SOAP, Biztalk Framework). В поддержку этих новшеств Microsoft выпустила инструментарий для разработки приложений – платформу .NET. Она также объединяла «под одной крышей» несколько языков программирования, что было в новинку для того времени.

Ещё одним новшеством платформы .NET была технология активных серверных страниц ASP.NET (Active Server Page). С её помощью можно было относительно быстро разработать веб-приложения, взаимодействующие с базами данных.

Язык программирования C# был создан специально для ASP.NET. На C# полностью была написана и сама ASP.NET.

Название «Си шарп» (от англ. sharp — диез) несёт «сакральный» смысл. Знак «#» (в музыкальной нотации читается как «диез») означает повышение

высоты звука на полтона. С другой стороны, название «C#» получается путём следующей «эволюционной цепочки»: C → C++ → C++++(C#), так как символ «#» можно составить из 4-х знаков «+».

Авторами этого языка программирования стали Скотт Вилтамут и Андерс Хейльсберг — создатель Турбо Паскаля и Дельфи, перешедший в 1996 году в Microsoft.

C# поддерживает все три «столпа» объектно-ориентированного программирования: инкапсуляцию, наследование и полиморфизм. Кроме того, в нем была реализована автоматическая «сборка мусора», обработки исключений, динамическое связывание.

Особенности языка

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

Название языка

Название «Си шарп» (от англ. sharp — диез) происходит от буквенной музыкальной нотации, где латинской букве «C» соответствует нота «До», а знак диез (англ. sharp) означает повышение соответствующего ноте звука на полутон, что аналогично названию языка C++, где «++» обозначает инкремент

переменной. Название также является игрой с цепочкой C → C++ → C++++(C#), так как символ «#» можно представить состоящим из 4 знаков «+».

Из-за технических ограничений на отображение (стандартные шрифты, браузеры и т. д.) и того, что знак диеза ♯ не представлен на стандартной клавиатуре компьютера, при записи имени языка программирования используют знак решётки (#). Это соглашение отражено в Спецификации языка C# ECMA-334. Тем не менее, на практике (например, при размещении рекламы и коробочном дизайне), «Майкрософт» использует знак диеза.

Названия языков программирования не принято переводить, поэтому язык называют, используя транскрипцию, — «Си шарп».

Microsoft SQL Server

Microsoft SQL Server — система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с небольшими и средними по размеру базами данных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL является общепринятым интерфейсом к базам данных. «Все промышленные базы — Oracle, Microsoft SQL Server, PostgreSQL, MySQL — работают на SQL.

История

Исходный код MS SQL Server (до версии 7.0) основывался на коде Sybase SQL Server, и это позволило Microsoft выйти на рынок баз данных для предприятий, где конкурировали Oracle, IBM, и, позже, сама Sybase. Microsoft, Sybase и Ashton-Tate первоначально объединились для создания и выпуска на рынок первой версии программы, получившей название SQL Server 1.0 для OS/2 (около 1989 года), которая фактически была эквивалентом Sybase SQL Server 3.0 для Unix, VMS и др. Microsoft SQL Server 4.2 был выпущен в 1992 году и входил в состав операционной системы Microsoft OS/2 версии 1.3.

Официальный релиз Microsoft SQL Server версии 4.21 для ОС Windows NT состоялся одновременно с релизом самой Windows NT (версии 3.1). Microsoft SQL Server 6.0 был первой версией SQL Server, созданной исключительно для архитектуры NT и без участия в процессе разработки Sybase.

К тому времени, как вышла на рынок ОС Windows NT, Sybase и Microsoft разошлись и следовали собственным моделям программного продукта и маркетинговым схемам. Microsoft добивалась исключительных прав на все версии SQL Server для Windows. Позже Sybase изменила название своего продукта на Adaptive Server Enterprise во избежание путаницы с Microsoft SQL Server. До 1994 года Microsoft получила от Sybase три уведомления об авторских правах как намёк на происхождение Microsoft SQL Server.

После разделения компании сделали несколько самостоятельных релизов программ. SQL Server 7.0 был первым сервером баз данных с настоящим пользовательским графическим интерфейсом администрирования. Для устранения претензий со стороны Sybase в нарушении авторских прав, весь наследуемый код в седьмой версии был переписан.

Версия SQL Server 2005 — была представлена в ноябре 2005 года. Запуск версии происходил параллельно запуску Visual Studio 2005. Существует также «урезанная» версия Microsoft SQL Server — Microsoft SQL Server Express; она доступна для скачивания и может бесплатно распространяться вместе с использующим её программным обеспечением.

С момента выпуска предыдущей версии SQL Server (SQL Server 2000) было осуществлено развитие интегрированной среды разработки и ряда дополнительных подсистем, входящих в состав SQL Server 2005. Изменения коснулись реализации технологии ETL (извлечение, преобразование и загрузка данных), входящей в состав компонента SQL Server Integration Services (SSIS), сервера оповещения, средств аналитической обработки многомерных моделей данных (OLAP) и сбора релевантной информации (обе службы входят в состав Microsoft Analysis Services), а также нескольких служб сообщений, а именно

Service Broker и Notification Services. Помимо этого, были произведены улучшения в производительности.

Функциональность

Microsoft SQL Server в качестве языка запросов использует версию SQL, получившую название Transact-SQL (сокращённо T-SQL), являющуюся реализацией SQL-92 (стандарт ISO для SQL) с множественными расширениями. T-SQL позволяет использовать дополнительный синтаксис для хранимых процедур и обеспечивает поддержку транзакций (взаимодействие базы данных с управляющим приложением). Microsoft SQL Server и Sybase ASE для взаимодействия с сетью используют протокол уровня приложения под названием Tabular Data Stream (TDS, протокол передачи табличных данных). Протокол TDS также был реализован в проекте FreeTDS с целью обеспечить различным приложениям возможность взаимодействия с базами данных Microsoft SQL Server и Sybase.

Microsoft SQL Server также поддерживает Open Database Connectivity (ODBC) — интерфейс взаимодействия приложений с СУБД. Версия SQL Server 2005 обеспечивает возможность подключения пользователей через веб-сервисы, использующие протокол SOAP. Это позволяет клиентским программам, не предназначенным для Windows, кроссплатформенно соединяться с SQL Server. Microsoft также выпустила сертифицированный драйвер JDBC, позволяющий приложениям под управлением Java (таким как BEA и IBM WebSphere) соединяться с Microsoft SQL Server 2000 и 2005.

SQL Server поддерживает зеркалирование и кластеризацию баз данных. Кластер сервера SQL — это совокупность одинаково конфигурированных серверов; такая схема помогает распределить рабочую нагрузку между несколькими серверами. Все сервера имеют одно виртуальное имя, и данные распределяются по IP-адресам машин кластера в течение рабочего цикла. Также в случае отказа или сбоя на одном из серверов кластера доступен автоматический перенос нагрузки на другой сервер.

SQL Server поддерживает избыточное дублирование данных по трем сценариям:

- Снимок: Производится «снимок» базы данных, который сервер отправляет получателям.
- История изменений: Все изменения базы данных непрерывно передаются пользователям.
- Синхронизация с другими серверами: Базы данных нескольких серверов синхронизируются между собой. Изменения всех баз данных происходят независимо друг от друга на каждом сервере, а при синхронизации происходит сверка данных. Данный тип дублирования предусматривает возможность разрешения противоречий между БД.

В SQL Server 2005 встроена поддержка .NET Framework. Благодаря этому, хранимые процедуры БД могут быть написаны на любом языке платформы .NET, используя полный набор библиотек, доступных для .NET Framework, включая Common Type System (система обращения с типами данных в Microsoft .NET Framework). Однако, в отличие от других процессов, .NET Framework, будучи базисной системой для SQL Server 2005, выделяет дополнительную память и выстраивает средства управления SQL Server вместо того, чтобы использовать встроенные средства Windows. Это повышает производительность в сравнении с общими алгоритмами Windows, так как алгоритмы распределения ресурсов специально настроены для использования в структурах SQL Server.

Разработка приложений

Microsoft и другие компании производят большое число программных средств разработки, позволяющих разрабатывать бизнес-приложения с использованием баз данных Microsoft SQL Server. Microsoft SQL Server 2005 включает в себя также Common Language Runtime (CLR) Microsoft .NET, позволяющий реализовывать хранимые процедуры и различные функции приложениям, разработанным на языках платформы .NET (например, VB.NET или C#). Предыдущие версии средств разработки Microsoft использовали только API для получения функционального доступа к Microsoft SQL Server.

РАЗДЕЛ 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Описание программного продукта

Информационная система «Mercury» строится на клиент-серверной архитектуре. Информационные системы с клиент-серверной архитектурой позволяют избежать проблем файл-серверных приложений. При такой архитектуре сервер базы данных, расположенный на компьютере-сервере, обеспечивает выполнение основного объема обработки данных. Клиентское приложение формирует запросы к серверу базы данных, как правило, в виде инструкций языка SQL. Сервер извлекает из базы запрошенные данные и передает на компьютер клиента. Главное достоинство такого подхода — значительно меньший объем передаваемых данных.

Большинство конфигураций информационных систем типа «клиент-сервер» использует двухуровневую модель, в которой клиент обращается к серверу (Нисунок 1).

Рисунок 1 – Структура информационной системы с сервером базы данных

Обеспечение безопасности данных — очень важная функция для успешной работы информационной системы. Если у базы данных слабая система безопасности, любой достаточно подготовленный пользователь может нанести серьезный ущерб работе предприятия. Следует отметить, что защита

данных в файл-серверной информационной системе изначально не может быть обеспечена на должном уровне.

Безопасность же современных серверов баз данных, организованная в нескольких направлениях: с помощью самой операционной системы; с использованием схем, имен входов, ролей, шифрования базы данных и т. д.; путем ограничения доступа пользователей через представления, заслуживает похвалы. В настоящее время архитектура «клиент-сервер» широко признана и находит применение для организации работы приложений как для рабочих групп, так и для информационных систем масштаба предприятия.

База данных информационной системы состоит из 11-ти таблиц:

- сессия;
- папка;
- пользователь;
- код подтверждения;
- уведомления;
- безопасность пользователя;
- безопасность папки;
- значок папки;
- безопасность;
- значок;
- категория.

Таблицы связаны между собой первичными ключами, для обеспечения каскадного обновления и удаления данных. На схеме данных, представлены все связи между таблицами (Рисунок 2).

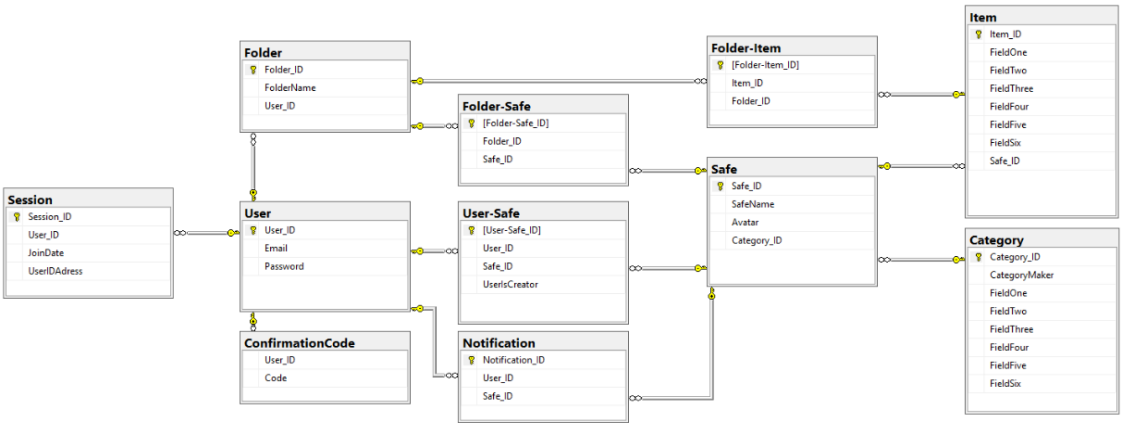

Рисунок 2 – Схема данных

Приложение состоит из следующих окон:

- Основное меню (Рисунок 3).

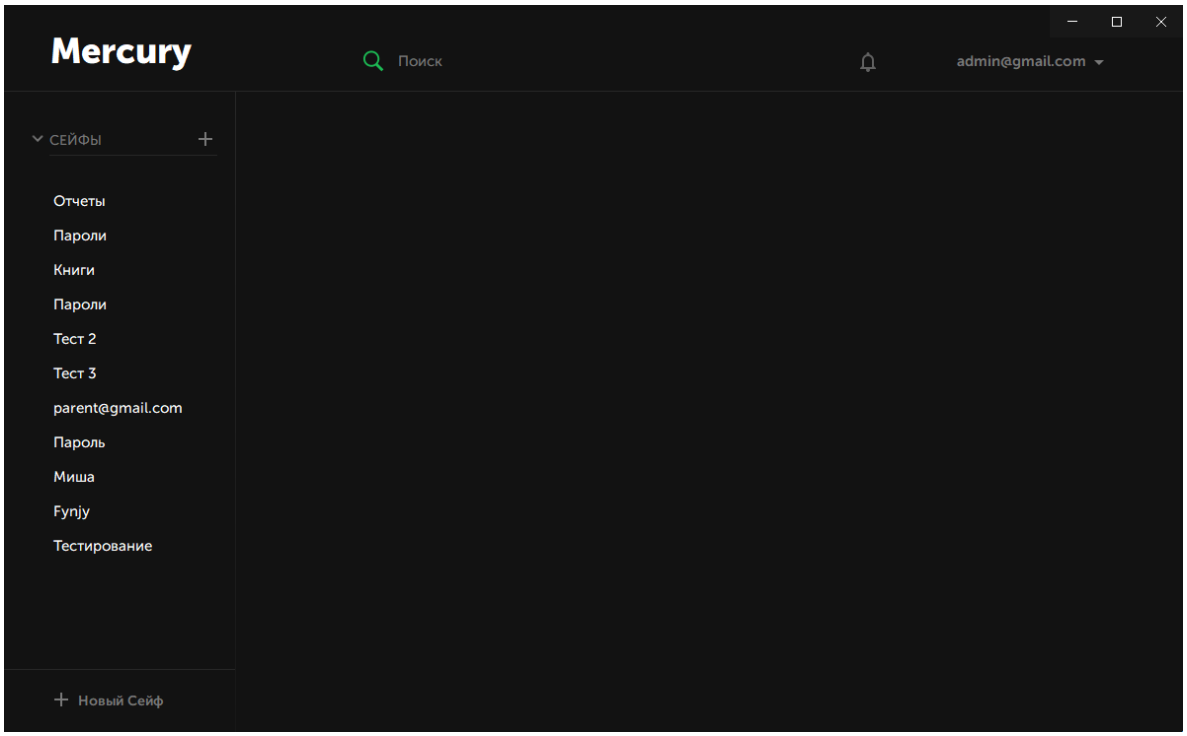

Рисунок 3 – Основное меню

В главном окне программы можно:

1. Создавать сейф.
2. Открывать сейф.
3. Изменять учётную запись.
4. Приглашать других пользователей.
5. Редактировать сейф.

6. Удалять сейф.
7. Искать сейфы и их элементы.
8. Создавать папки.
9. Редактировать папки.
10. Удалять папки.
11. Выгонять других пользователей.

- окно создания нового сейфа (Рисунок 4).

При создании нового сейфа указывается его название и поля.

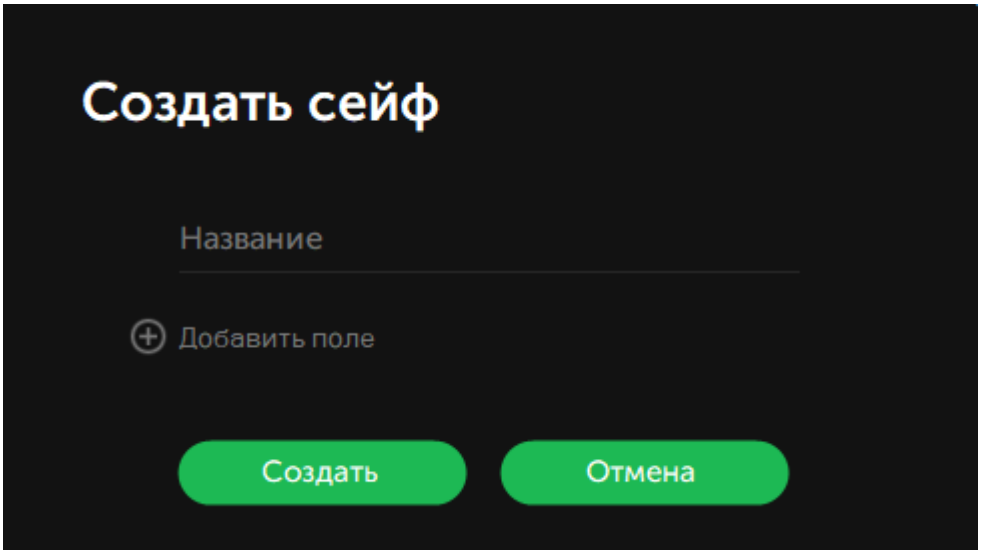

Рисунок 4 – Окно создания сейфа

- Окно создания папки (Рисунок 5).

При создании папки указывается её название.

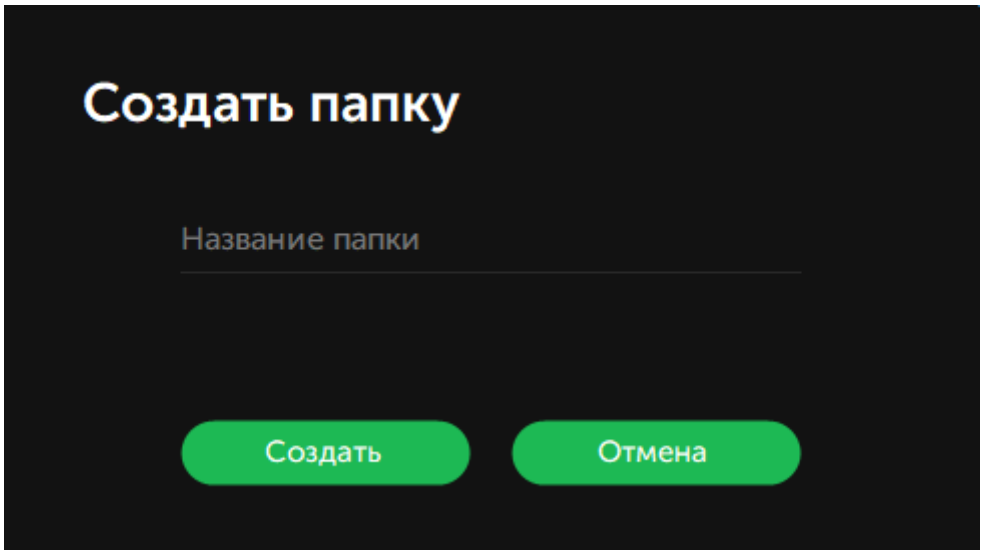

Рисунок 5 – Окно создания папки

- Окно приглашения новых пользователей (Рисунок 6).

При приглашении новых пользователей указывается адрес электронной почты.

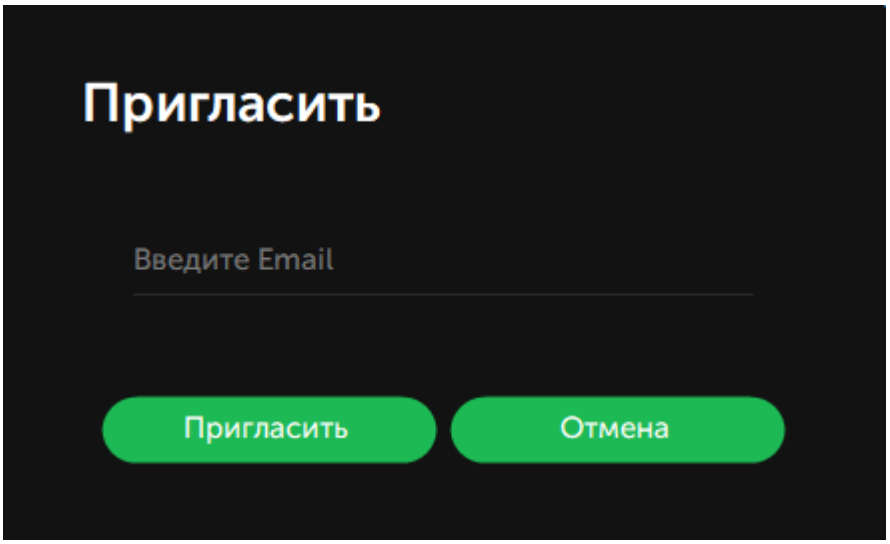

Рисунок 6 – Окно приглашения новых пользователей

- Окно редактирования сейфа (Рисунок 7).

При редактировании сейфа есть возможность изменить название сейфа и его полей.

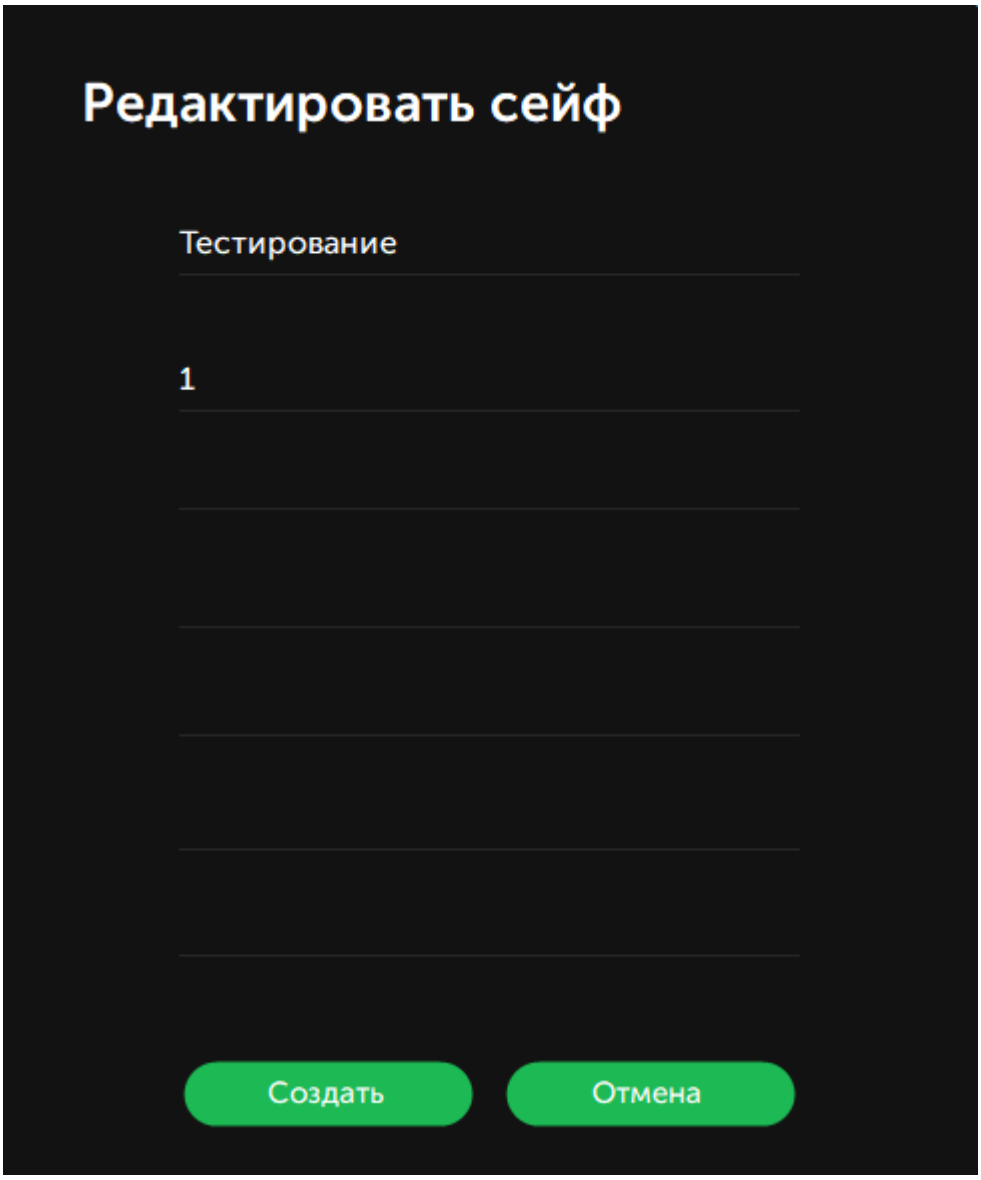

Рисунок 7 – Окно редактирования сейфа

В каждом сейфе есть возможность выгонять пользователей (Рисунок 8).

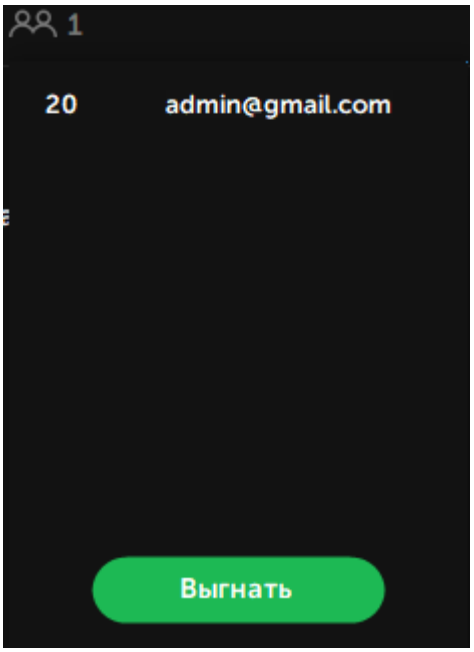

Рисунок 8 – Выгнать пользователя

В приложении присутствуют уведомления (Рисунок 9).

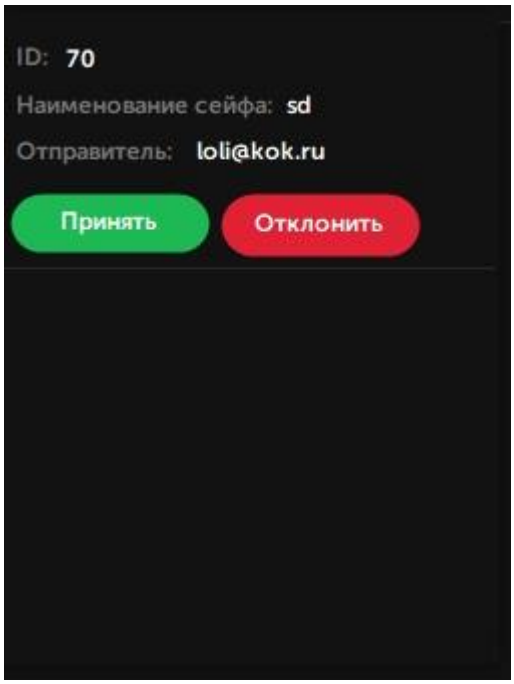

Рисунок 9 – Уведомление

2.2. Тестирование программного продукта

При первом открытии программы нас просят авторизоваться или пройти регистрацию (Рисунок 10).

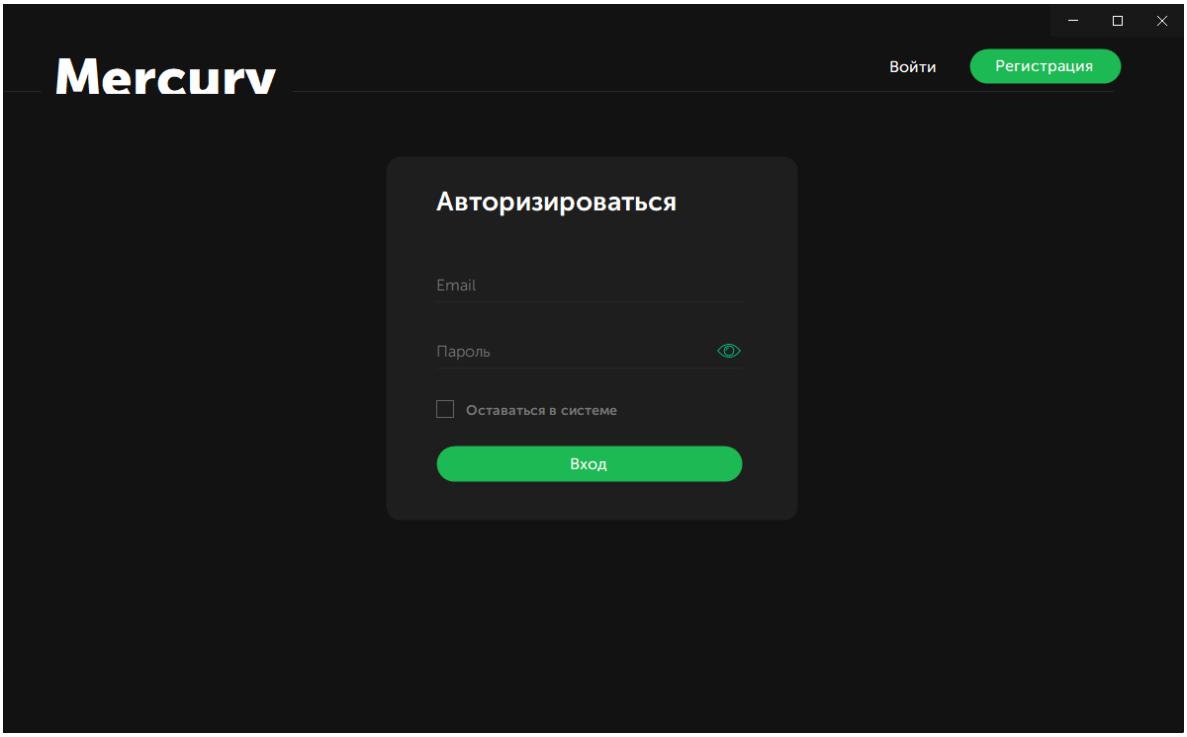

Рисунок 10 – Авторизация

Попробуем ввести недействующий логин и пароль, и нажать кнопку «Вход» (Рисунок 11).

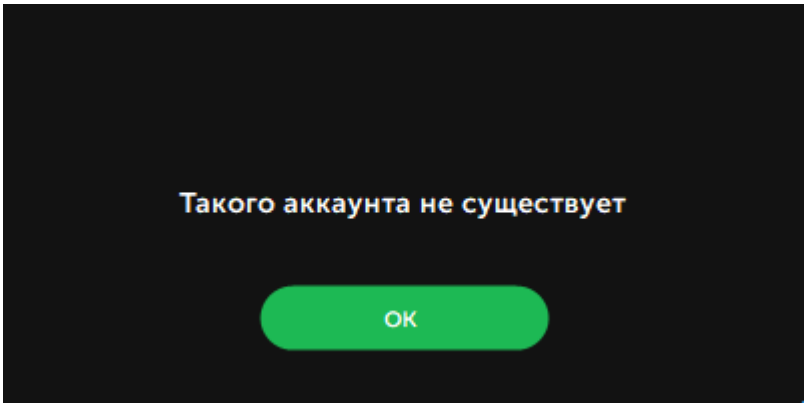

Рисунок 11 – Авторизация

Зарегистрируем новый аккаунт, для этого нажмём на кнопку «Регистрация» (Рисунок 12).

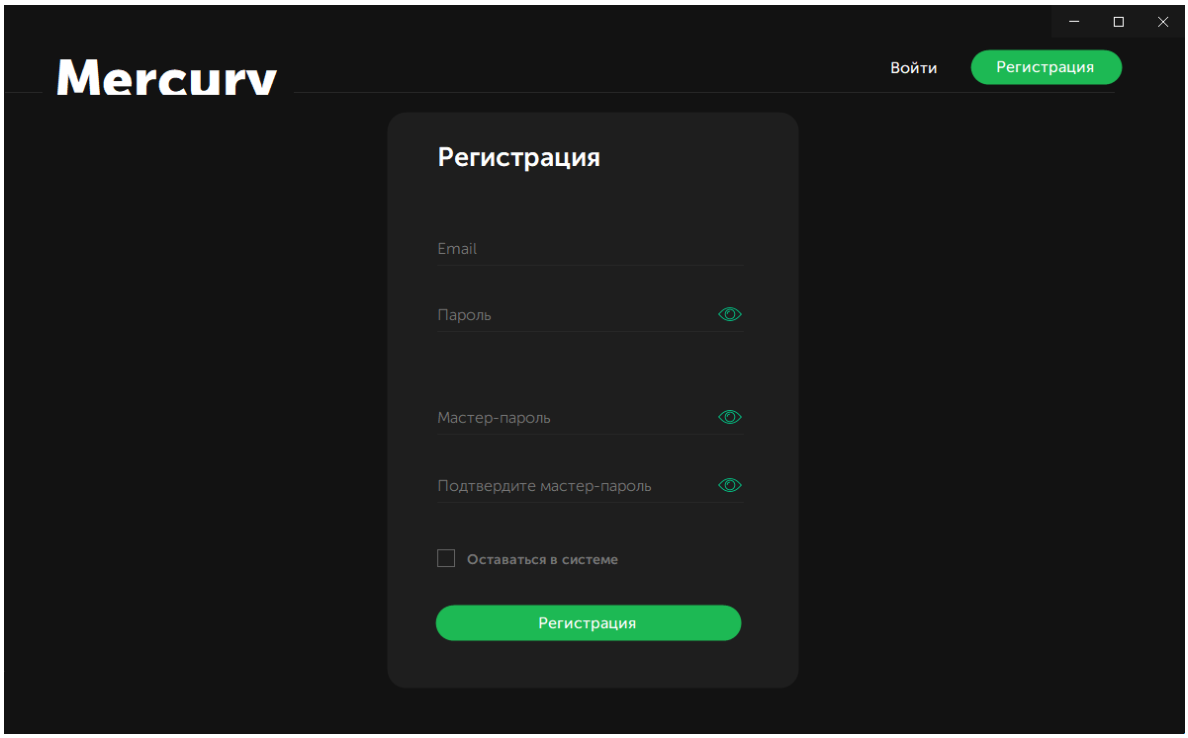

Рисунок 12 – Регистрация

Введём данные и нажмём на кнопка «Регистрация» (Рисунок 13).

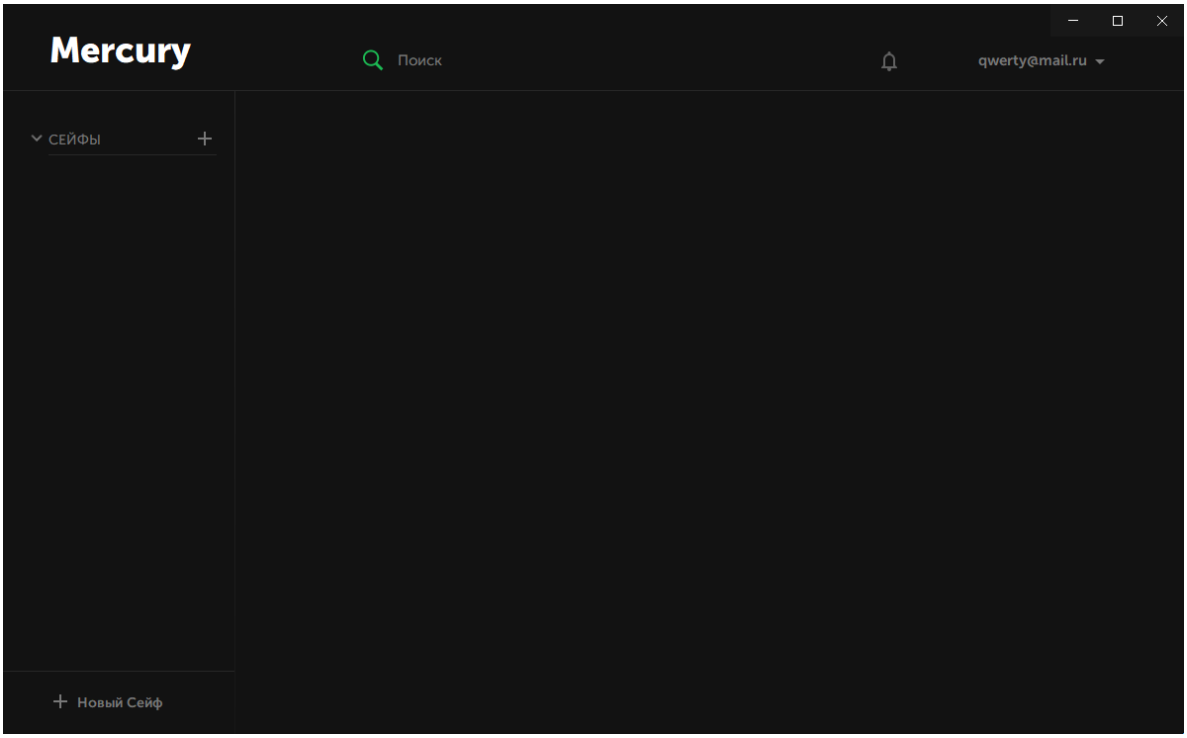

Рисунок 13 – Регистрация

Попробуем создать сейф. Для этого нажмём на кнопку «Новый сейф» (Рисунок 14).

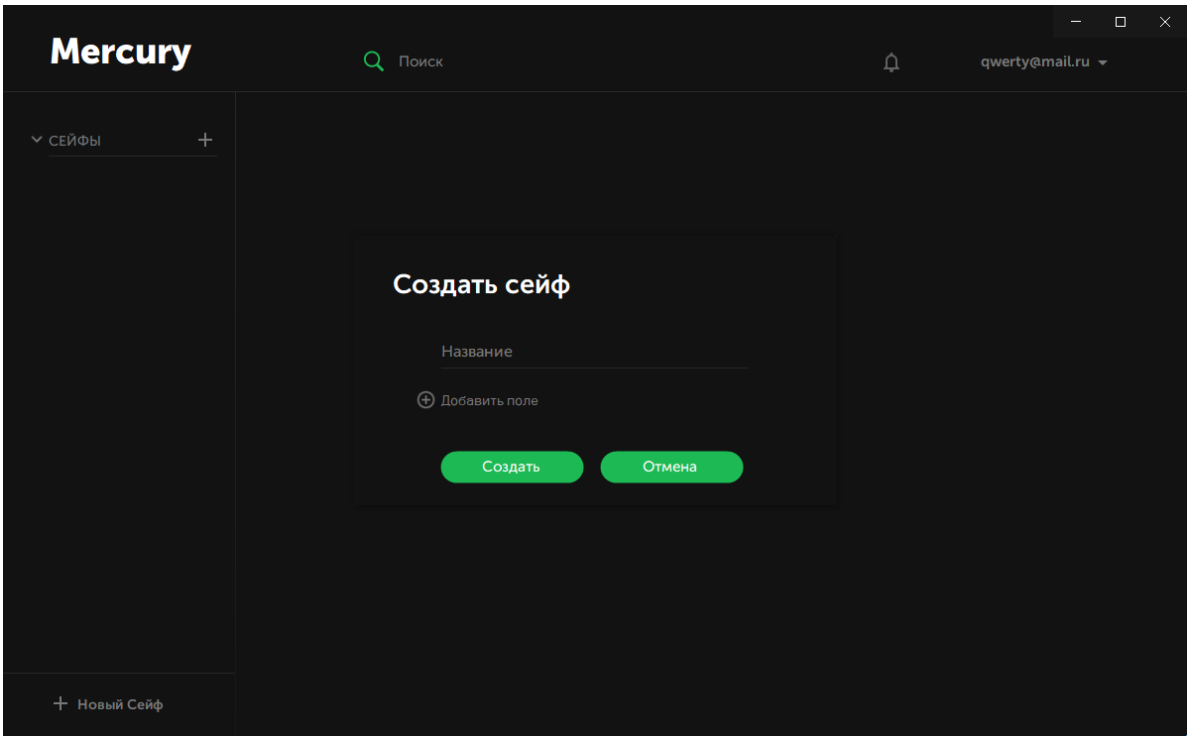

Рисунок 14 – Создание сейфа

Введём название и добавим поля. После этого нажмём на кнопку «Создать» (Рисунок 15).

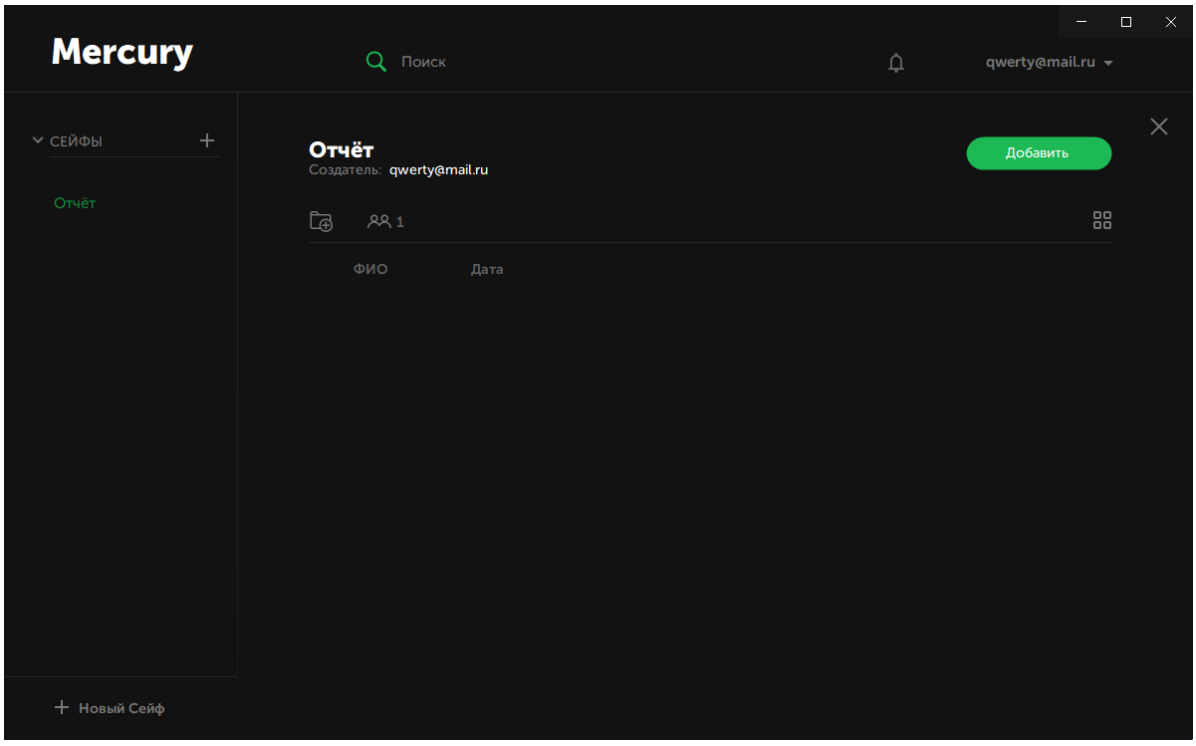

Рисунок 15 — Создание сейфа

Добавим запись в наш сейф, для этого нажмём на кнопку «Добавить» (Рисунок 16).

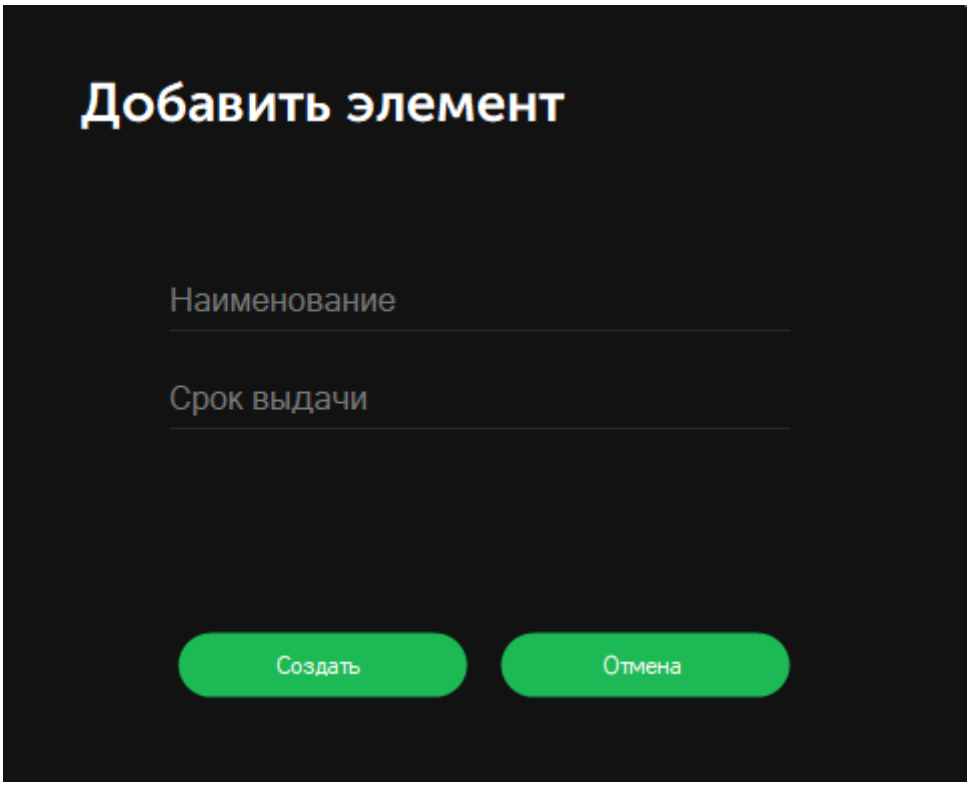

Рисунок 16 – Создание записи

Введём нужные данные и нажмём кнопку «Создать» (Рисунок 17).

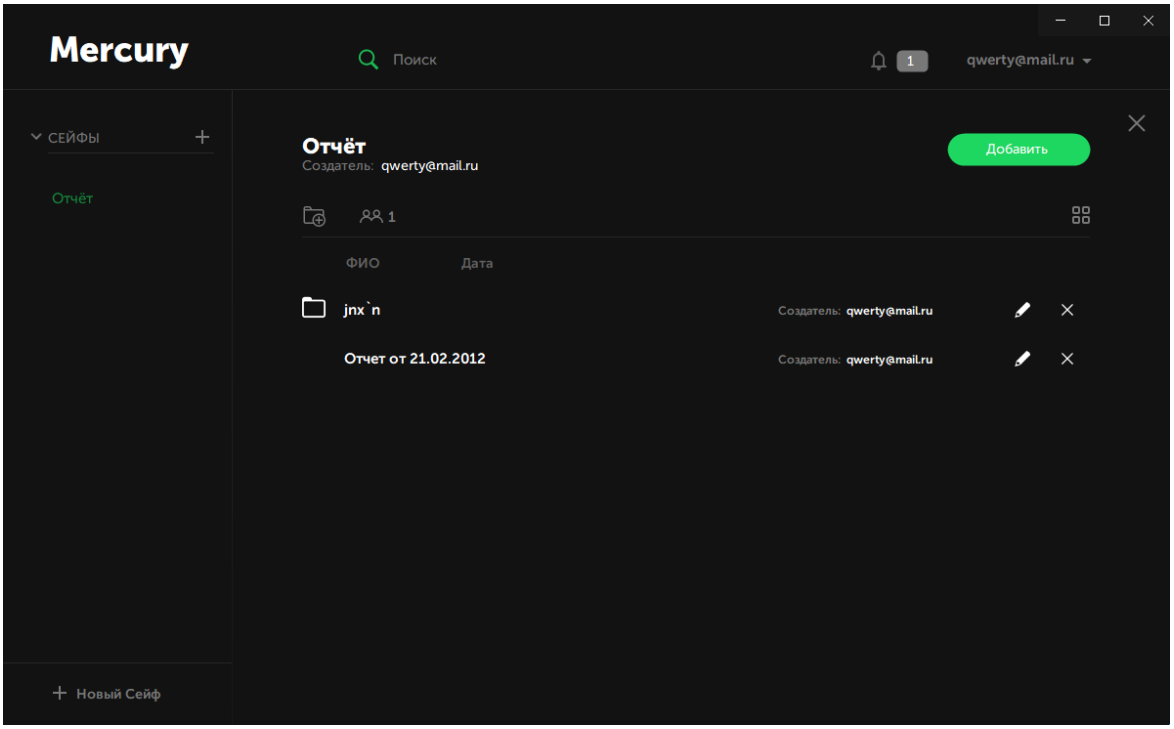

Рисунок 17 – Новая запись создана

В верхней части экрана нам пришло уведомление, это приглашение в сейф (Рисунок 18).

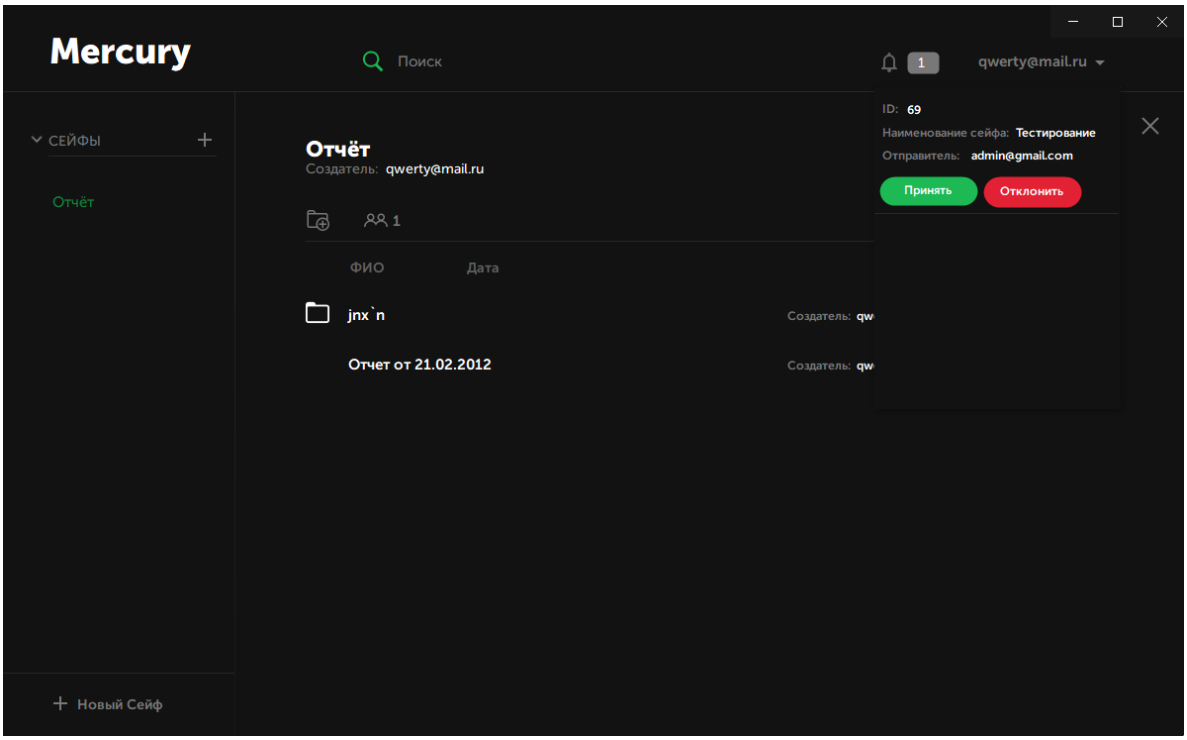

Рисунок 18 – Уведомление

Нажмём на кнопку «Принять» и посмотрим на результат (Рисунок 19).

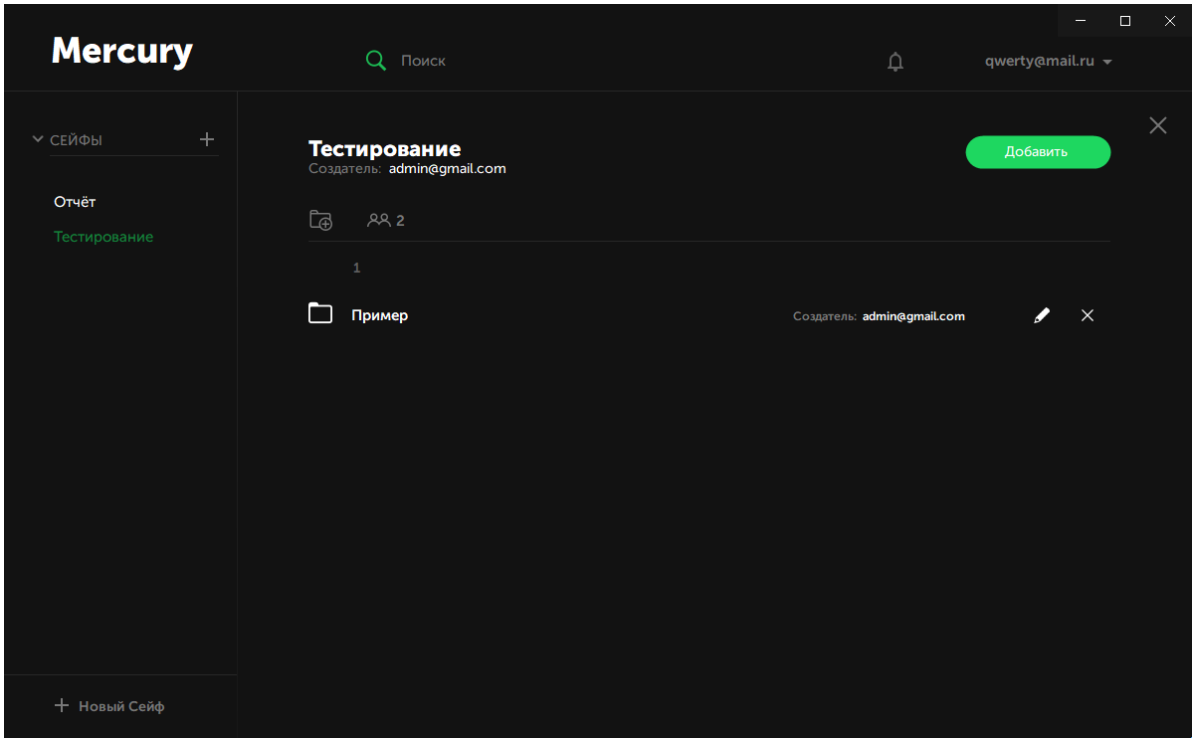

Рисунок 19 – Приглашение

Попробуем удалить папку в сейфе «Тестирование», для этого нажмём на крестик справа от папки (Рисунок 20).

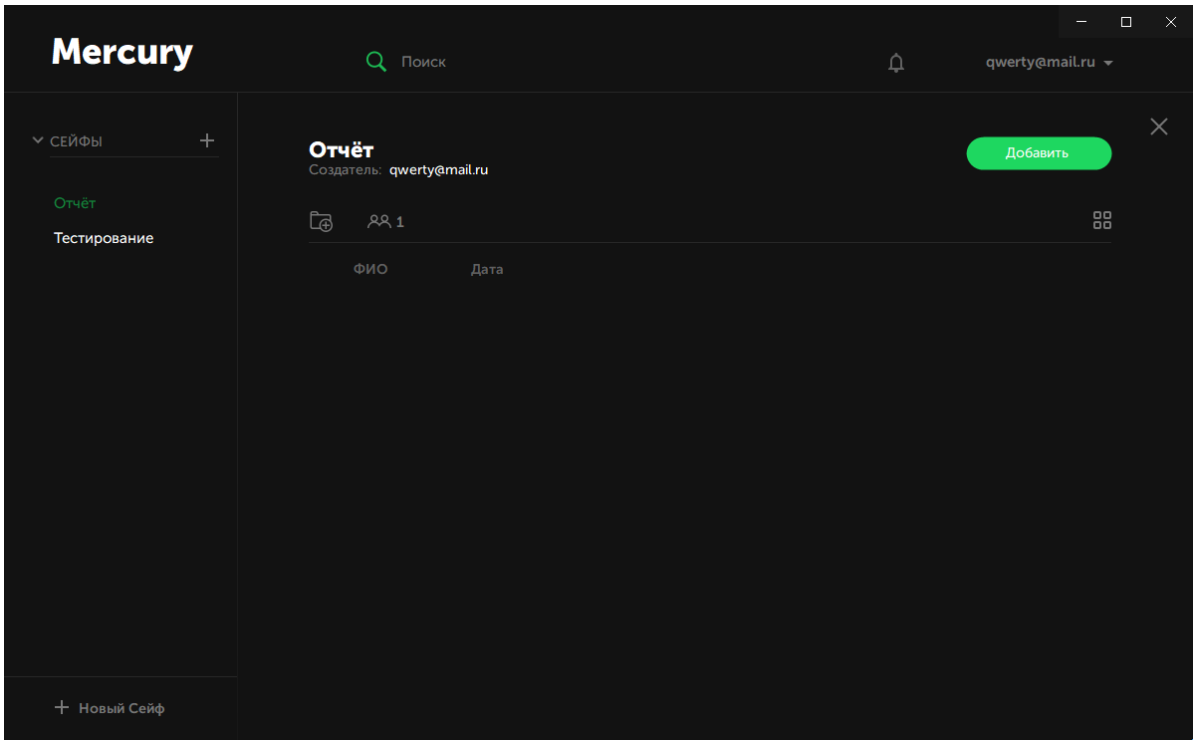

Рисунок 20 – Удаление

Добавим папку в наш сейф, для этого нажмём на кнопку добавления папки (Рисунок 21).

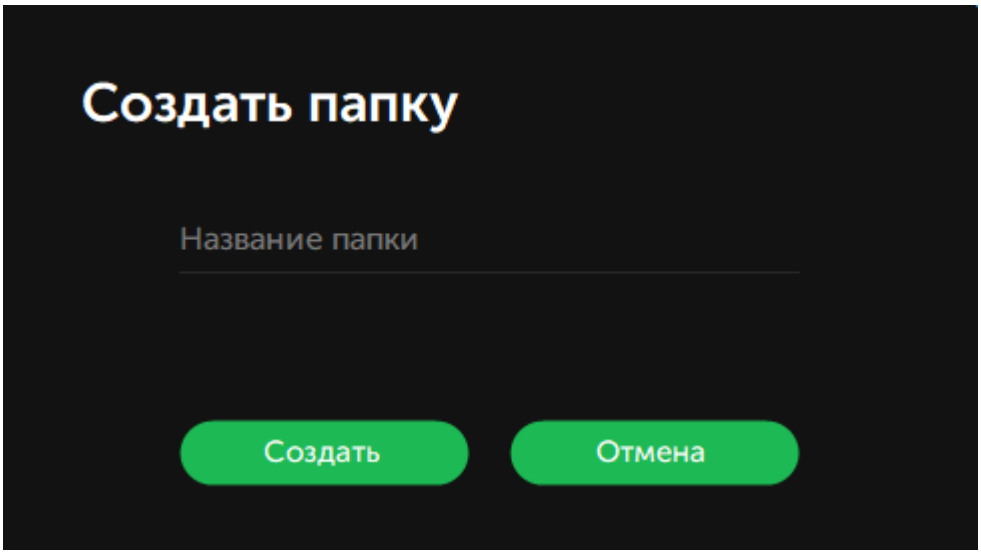

Рисунок 21 – Создание папки

Введём название и нажмём на кнопку «Создать» (Рисунок 22).

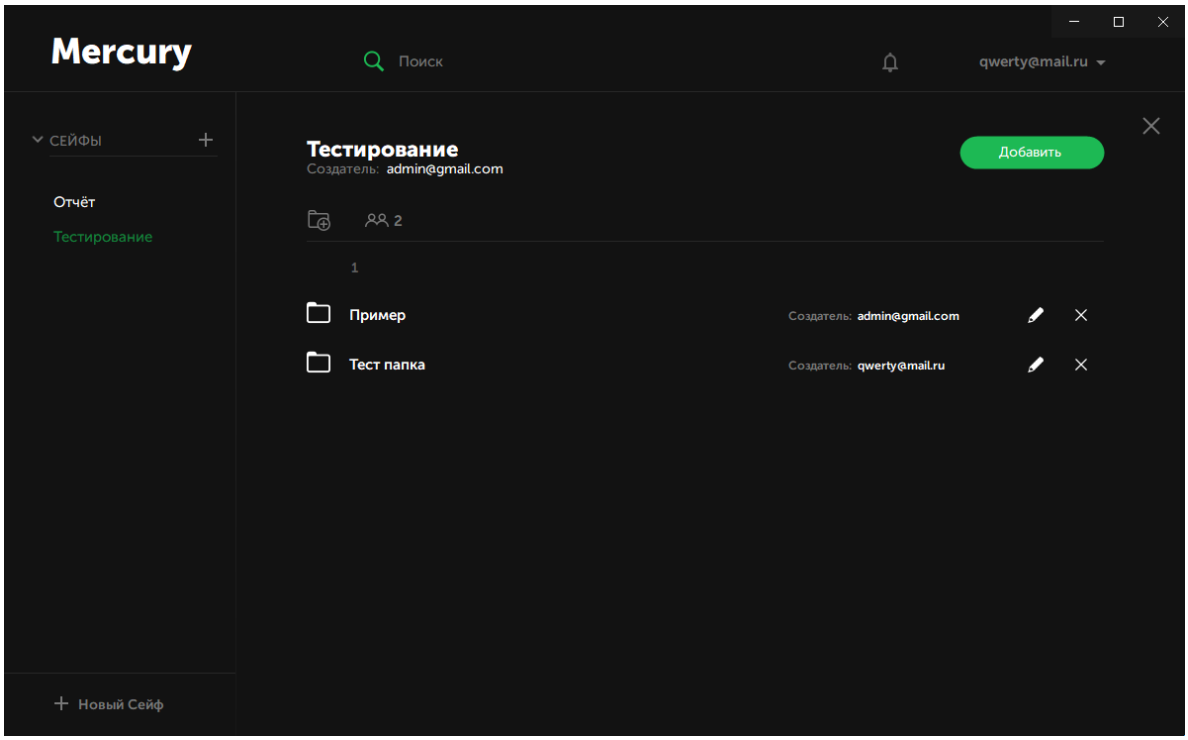

Рисунок 22 – Создание папки

При создании сейфа попробуем не ввести данные для заполнения (Рисунок 23).

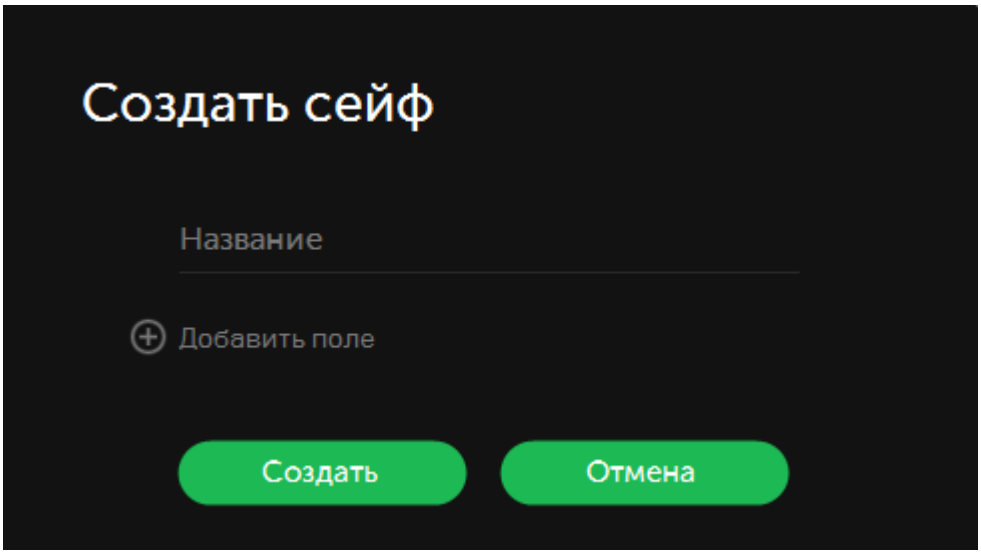

Рисунок 23 – Проверка при создание сейфа

Нажимаем кнопку «Создать» и смотрим результат (Рисунок 24).

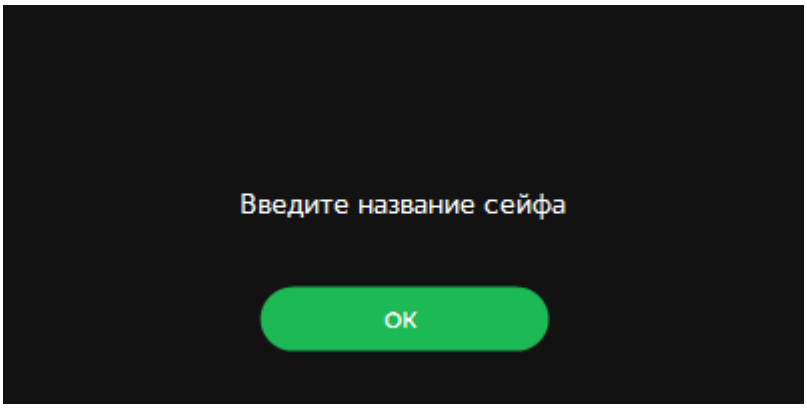

Рисунок 24 – Проверка при создание сейфа

Введём информацию в поле «Название», а поле «Добавить поле» оставим не тронутым (Рисунок 25).

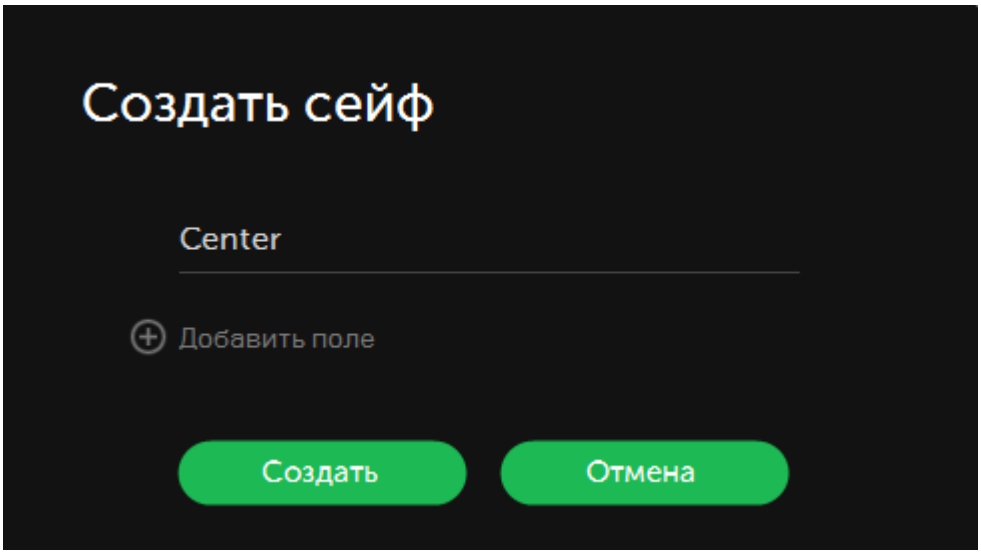

Рисунок 25 – Проверка при создание сейфа

Нажимаем кнопку «Создать» и смотрим результат (Рисунок 26).

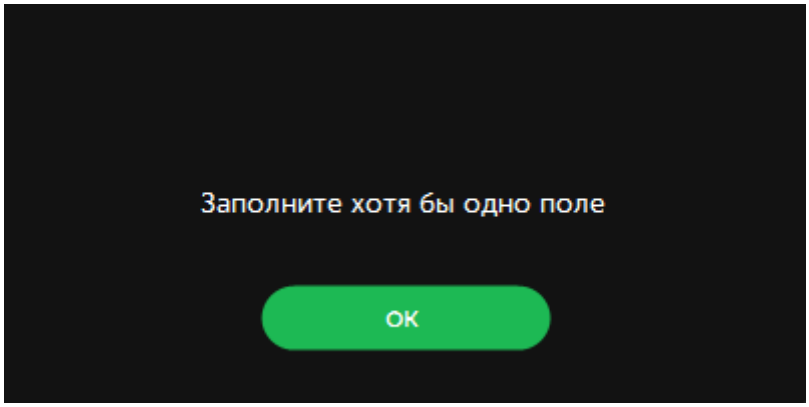

Рисунок 26 – Проверка при создание сейфа

Попробуем пригласить нового пользователя с помощью встроенной функции. Для этого зайдём в сейф, нажмём на встроенную кнопку «Меню сейфа» и нажмём пригласить. Введём несуществующий Email (Рисунок 27).

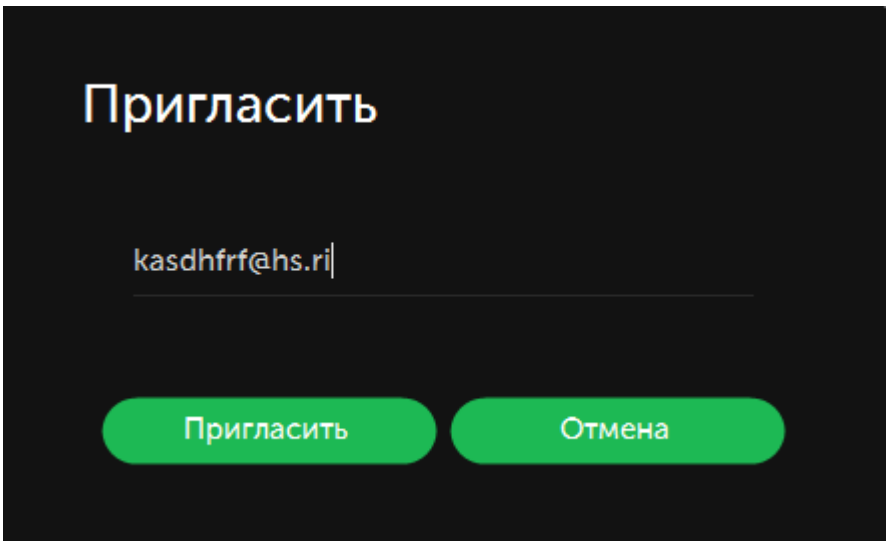

Рисунок 27 – Проверка при приглашении нового пользователя

Нажимаем кнопку «Пригласить» и смотрим результат (Рисунок 28).

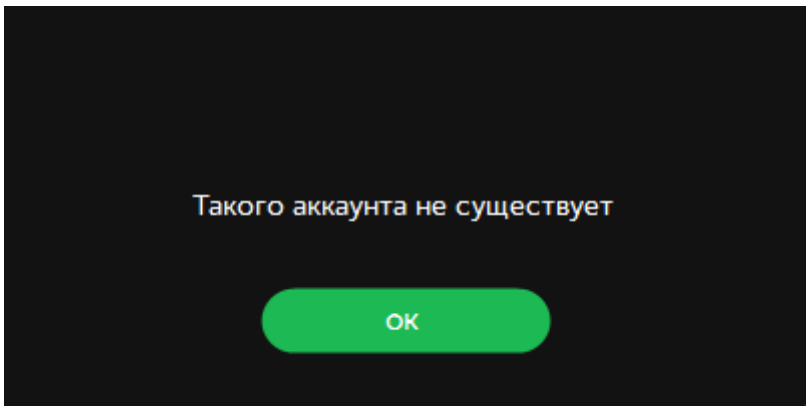

Рисунок 28 – Проверка при приглашении нового пользователя

Попробуем ввести пользователя который уже участвует в развитии сейфа и пригласить его, для этого введём пользователя в Email-ом qwerty@mail.ru (Рисунок 29).

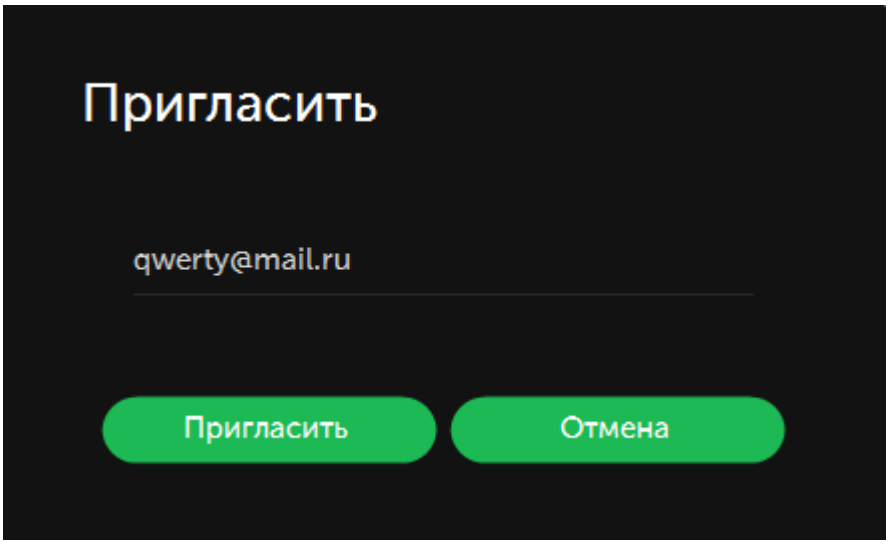

Рисунок 29 – Проверка при приглашении нового пользователя

Нажимаем кнопку «Пригласить» и смотрим результат (Рисунок 30).

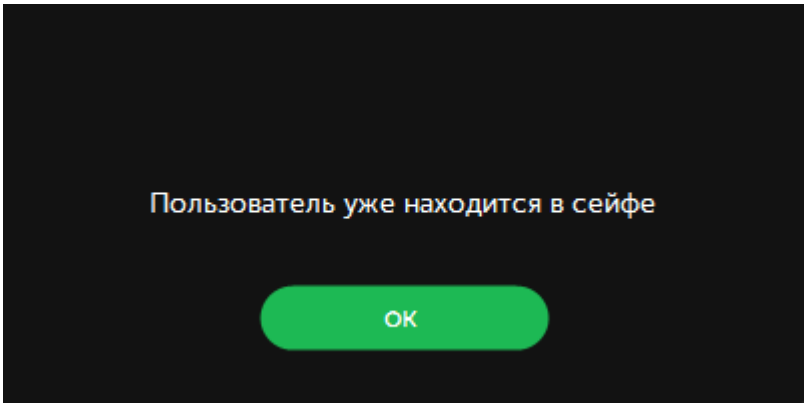

Рисунок 30 – Проверка при приглашении нового пользователя

ЗАКЛЮЧЕНИЕ

В современном мире электронные технологии применяются уже давно и обширно. Они применяются как в современном образовании, так и в простом использовании.

Каждый человек хранит огромное количество данных требуемых ему. Специально для таких целей была разработана программа, которая позволяет хранить любые данные и делиться с ними другим пользователям для общего просмотра или редактирования.

В качестве языка программирования использовался всем известный CSharp (C#). Он имеет понятный, а главное приятный синтаксис и множество полезный и необходимых для современной разработки функций. Язык постоянно обновляется, что даёт ему не стоять на месте в эпохе прогресса и инноваций.

В процессе разработки данного дипломного проекта я как закрепил уже знающий материал, и получил множество мелких тонкостей разработки программного обеспечения.

Что дало мне больше опыта в разработке программного обеспечения на будущее и доведение реализации новых алгоритмов до автоматизма.

СПИСОК ЛИТЕРАТУРЫ

1. Компьютерное тестирование в образовании. Форма доступа: http://koi.tspu.ru/koi_books/samolyuk/lek11.htm

2. Многоклиентский сетевой протокол на C#. Форма доступа: https://habr.com/ru/post/142819/

3. Описание языка C#. Форма доступа: https://ru.wikipedia.org/wiki/C_Sharp

4. Передача данных. Форма доступа: https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85

5. Сетевое программирование в C# и .NET. Форма доступа: https://metanit.com/sharp/net/

6. Рой Ошероу - Искусство автономного тестирования с примерами на C# - ДМК Пресс, 2016, 360 стр.

7. Троелсен Эндрю, Джепикс Филипп - Язык программирования C# 7 и платформы .NET и .NET Core – Вильямс, 2018, 1328 стр.

ПРИЛОЖЕНИЯ

```
using System.Drawing;
using System.Drawing.Text;
using pivyLab.UserForms;
using System.IO;
using System.Windows.Forms;
using System;
using System.Collections.Generic;
using System.Linq;
using Mercury.WorkingScripts;
using System.Runtime.Serialization.Formatters.Binary;
using Mercury.CustomControls;

namespace Mercury
{


    public partial class Main : FormTwo
    {

        #region Хуки и глобальные переменные

        // Создаем коллекцию
        PrivateFontCollection pr = new PrivateFontCollection();
        // Переменная, которая говорит о том, активно ли меню выхода
        bool emailRightSideMenu = false;
        // Правое меню на верхней панели
        CustomControls.RightSideMenu rightSideMenu = new
CustomControls.RightSideMenu();
        // Панель создания сейфа
        CustomControls.CreateSafeForm createSafe = new CustomControls.CreateSafeForm();
        // Панель участников сейфа
        MembersInSafe members = new MembersInSafe();
        // Панель уведомлений
```

```
public NotificationForm notification = new NotificationForm();
// Панель меню сейфа
public SafeMenu safeMenu = new SafeMenu();
// Панель редактирования сейфа
public EditSafe editSafe;
// Количество открытых меню
public int countOpenMenu = 0;
// Хук количества открытых меню сейфа
public int countOpenSafeMenu = 0;
// Количество панелей создания сейфа
public int countOpenMenuCreateSafe = 0;
// Хук открытого меню сейфа
public bool safeMenuIsOpen = false;
// Хук открытого меню участников сейфа
public bool membersMenuIsOpen = false;
// Хук количества открытых меню сейфа
public int countOpenSafeMembers = 0;
// Хук открытого меню уведомлений
public bool notificationMenuIsOpen = false;
// Хук количества открытых меню уведомлений
public int countOpenNotificationMenu = 0;

// Активный сейф
public Safe activeSafe;

#endregion

#region Списки

// Список созданных сейфов
public List<Safe> safeCollection = new List<Safe>();
public List<Folder> folderCollectionInActiveSafe = new List<Folder>();
public List<Notification> notificationCollection = new List<Notification>();

#endregion
```

```
        #region Вспомогательные методы

        #region Использование шрифтов

        /// <summary>
        /// Использует шрифт из файла.
        /// </summary>
        private void UseFonts()
        {
            // Сохраняем путь к папке со шрифтами
            Properties.Settings.Default.PathForFonts = Directory.GetCurrentDirectory()
                .Remove(Directory.GetCurrentDirectory().Length - 10) + "\\Fonts\\";
            Properties.Settings.Default.Save();

            // Добавляем шрифты в коллекцию
            pr.AddFontFile(Properties.Settings.Default.PathForFonts + "MuseoSansCyrl-
300.ttf");
            pr.AddFontFile(Properties.Settings.Default.PathForFonts + "MuseoSansCyrl-
500.ttf");
            pr.AddFontFile(Properties.Settings.Default.PathForFonts + "MuseoSansCyrl-
700.ttf");
            pr.AddFontFile(Properties.Settings.Default.PathForFonts + "MuseoSansCyrl-
900.ttf");
            pr.AddFontFile(Properties.Settings.Default.PathForFonts + "CircularStd-Black.otf");

            FontFamily[] fontFamilies = pr.Families;

            // Кнопка "Войти"
            loginButton.Font = new Font(fontFamilies[2], 12);
            // Кнопка "Регистрация"
            registrationButton.Font = new Font(fontFamilies[2], 12);
            // Надпись / логотип
            textLogo.Font = new Font(fontFamilies[4], 42);
            // Надпись / логотип (Main)
```

```
        textLogoMain.Font = new Font(fontFamilies[4], 27);
        // Поле "Поиск"
        searchText.Font = new Font(fontFamilies[2], 11);
        // Поле email'a
        emailText.Font = new Font(fontFamilies[3], 11);
        // Поле лого-лейбла на левой панели
        leftSideLogoLabel.Font = new Font(fontFamilies[2], 10);
        // Надпись "Новый сейф"
        createSafeButtonLabel.Font = new Font(fontFamilies[3], 10);
        // Наименование сейфа на панеле отображения элементов сейфа
        safeItemView_SafeName.Font = new Font(fontFamilies[4], 16);
        // Кнопка "Добавить" элемент на панели элементов сейфа
        safeItemView_AddItem.Font = new Font(fontFamilies[2], 10);
        // Поле создателя сейфа
        safeItemView_SafeCreator.Font = new Font(fontFamilies[2], 10);
        // Поле создателя сейфа
        safeItemView_SafeCreatorPerson.Font = new Font(fontFamilies[2], 10);
        // Поля отображения на панели сейфа
        safeItemView_Field1.Font = new Font(fontFamilies[3], 10);
        safeItemView_Field2.Font = new Font(fontFamilies[3], 10);
        safeItemView_Field3.Font = new Font(fontFamilies[3], 10);
        safeItemView_Field4.Font = new Font(fontFamilies[3], 10);
        safeItemView_Field5.Font = new Font(fontFamilies[3], 10);
        safeItemView_Field6.Font = new Font(fontFamilies[3], 10);
        // Количество участников в сейфе
        safeItemView_MembersCount.Font = new Font(fontFamilies[3], 11);
        // Количество уведомлений
        notificationCount.Font = new Font(fontFamilies[2], 10);
    }

    #endregion

    #region Верхняя панель

    /// <summary>
```

```
        /// Метод, который записывает email пользователя на верхнюю панель
        /// </summary>
        private void WriteEmail()
        {
            // Показываем email в нижнем регистре
            emailText.Text = Properties.Settings.Default.userEmail.ToLower();
            // Меняем положение email'a
            emailText.Location = new Point(emailIcon.Location.X - emailText.Width,
emailText.Location.Y);
            // Привязываем email к верхнему правому краю
            emailText.Anchor = (AnchorStyles.Top | AnchorStyles.Right);
        }

        /// <summary>
        /// Метод, который записывает количество уведомлений и подстраивает
отображение
        /// </summary>
        public void WriteNotification()
        {
            notificationPanel.Location = new Point(emailText.Location.X -
notificationPanel.Width - 20, notificationPanel.Location.Y);
            notificationPanel.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

            var count =
DateBase.GetNotificationCount(DateBase.GetUserID(Properties.Settings.Default.userEmail));

            if (count == 0)
                notificationCount.Visible = false;
            else
            {
                notificationCount.Visible = true;
                notificationCount.Text = DateBase.GetNotificationCount
                    (DateBase.GetUserID(Properties.Settings.Default.userEmail)).ToString();
            }
```

```
            LoadNotification();
        }

        /// <summary>
        /// Показывает меню участников сейфа
        /// </summary>
        public void OpenNotificationForm()
        {
            notification.Owner = this;
            notification.BringToFront();
            // Показываем меню
            notification.Show();
            // Меняем позицию меню уведомлений
            notification.Location = new Point(this.Location.X + notificationPanel.Location.X,
                this.Location.Y + notificationPanel.Location.Y + notificationPanel.Height + 10);
        }

        /// <summary>
        /// Скрывает меню участников сейфа
        /// </summary>
        public void CloseNotificationForm()
        {
            notification.Hide();
            notificationMenuIsOpen = false;
            countOpenNotificationMenu = 0;
        }

        /// <summary>
        /// Подгружает список уведомлений
        /// </summary>
        public void LoadNotification()
        {
            this.notificationCollection =
DateBase.GetNotificationList(DateBase.GetUserID(Properties.Settings.Default.userEmail));
        }
```

```
        /// <summary>
        /// Скрывает количество уведомлений
        /// </summary>
        public void HideNotificationCount()
        {
            notificationCount.Visible = false;
            CloseNotificationForm();
        }

        #endregion

        #region Меню на верхней панели

        /// <summary>
        /// Открываем правое меню из верхней панели
        /// </summary>
        private void OpenRightSideMenu()
        {
            // Выдаем права на форму
            rightSideMenu.Owner = this;
            // Показываем меню
            rightSideMenu.Show();
            // Меняем позицию правого меню
            rightSideMenu.Location = new Point(this.Location.X + (emailIcon.Location.X +
emailIcon.Width - rightSideMenu.Width),
                this.Location.Y + (emailIcon.Location.Y + emailIcon.Height + 10));
        }

        /// <summary>
        /// Закрываем правое меню из верхней панели
        /// </summary>
        private void CloseRightSideMenu() => rightSideMenu.Hide();

        /// <summary>
```

```
/// Выполняет выход их аккаунта
/// </summary>
public void Logout()
{
    // Показываем панель входа / регистрации
    startPanel.Location = new Point(1, 33);
    startPanel.Visible = true;
    // Меняем цвет текста email'a
    emailText.ForeColor = Color.FromArgb(120, 120, 120);
    // Меняем иконку
    emailIcon.Image = Properties.Resources.emailIconGrayDown;
    // Обнуляем хук
    emailRightSideMenu = false;
    // Обнуляем хук количества элементов меню
    countOpenMenu = 0;

    countOpenSafeMenu = 0;
    safeMenuIsOpen = false;
    CloseSafeMenu();

    // Обнуляем данные пользователя
    Properties.Settings.Default.userEmail = string.Empty;
    Properties.Settings.Default.userPassword = string.Empty;
    Properties.Settings.Default.Save();
    // Обнуляем хук
    Properties.Settings.Default.saveSession = false;

    // Разворачиваем контролы на весь экран
    login.Size = new Size(this.Width, this.Height - 80);
    registration.Size = new Size(this.Width, this.Height - 80);
    separator1.Size = new Size(this.Width, 1);

    // Чистим список сейфов
    safeList.Controls.Clear();
```

```
    safeItemView.Visible = false;
}

#endregion

#region Создание сейфа

/// <summary>
/// Метод, который проверяет сейфы на повторные названия
/// </summary>
/// <param name="name">Наименование сейфа</param>
public bool ValidateSafeName(string name)
{
    bool result = true;

    foreach (var item in safeList.Controls)
    {
        // Если сейф с таким названием уже существует
        if ((item as Label).Text == name)
            result = false;
    }

    return result;
}

/// <summary>
/// Метод, который обнуляет хук количества открытых панелей создания сейфа
/// </summary>
public void ZeroingCountCreateSafeMenu() => this.countOpenMenuCreateSafe = 0;

/// <summary>
/// Метод, который закрывает все окна
/// </summary>
public void ClearScreenOfWindow()
{
```

```
    // Закрываем окно создания сейфа
    createSafe.Close();
    // Чистим хук количества открытых меню
    ZeroingCountCreateSafeMenu();
}

/// <summary>
/// Возвращает шрифт для сейфа
/// </summary>
private Font GetFontForSafe() => new Font(pr.Families[2], 11);

/// <summary>
///  Получаем числовой показателя для следующего лейбла / Объекта сейфа
/// </summary>
private int GetLocationForSafe()
{
    // Если количество элементов в списке контролов
    // равно нулю, то возвращаем позицию первого контрола
    if (safeList.Controls.Count == 0)
        return 20;
    // Иначе возвращаем позицию в результате расчета
    // позиции последнего контрола в списке
    else
        return safeList.Controls[safeList.Controls.Count - 1].Location.Y + 35;
}

/// <summary>
/// Возвращает количество сейфов
/// </summary>
/// TODO: Проверить, правильно ли записываются названия сейфов!!!
private int GetCountSafe => safeList.Controls.Count + 1;

/// <summary>
///  Метод, который добавляет сейф на панель и добавляет его в список сейфов
/// </summary>
```

```
/// <param name="safe">Сейф</param>
public void CreateSafe(Safe safe)
{
    // Записываем данные в базу
    InsertSafeAndCategoryInDB(safe);
    // Записываем safeID
    safe.SafeID = DateBase.GetLastIDFromSafe();
    // Чистим список сейфов
    safeList.Controls.Clear();
    // Заполняем список сейфов
    FillSafeList();
    // Передаем фокус на сейф
    FocusedSafe(safe);
}

/// <summary>
/// Добавляет данные в таблицы Category и Safe
/// </summary>
/// <param name="safe"></param>
private void InsertSafeAndCategoryInDB(Safe safe)
{
    // Массив полей
    string[] fieldCategory = new string[6];
    // Массив значений
    string[] fieldValue = new string[6];

    // В зависимости от количества введенных полей - заполняем значения
    switch (safe.Fields.Count)
    {
        case 1:

            fieldCategory = new string[]
            {
                "CategoryMaker",
                "FieldOne"
```

```
        };
        fieldValue = new string[]
        {
            Properties.Settings.Default.userEmail,
            safe.Fields[0]
        };

        break;

    case 2:

        fieldCategory = new string[]
        {
            "CategoryMaker",
            "FieldOne",
            "FieldTwo"
        };
        fieldValue = new string[]
        {
            Properties.Settings.Default.userEmail,
            safe.Fields[0],
            safe.Fields[1]
        };

        break;

    case 3:

        fieldCategory = new string[]
        {
            "CategoryMaker",
            "FieldOne",
            "FieldTwo",
            "FieldThree"
        };
```

```
            fieldValue = new string[]
            {
                Properties.Settings.Default.userEmail,
                safe.Fields[0],
                safe.Fields[1],
                safe.Fields[2]
            };

            break;

        case 4:

            fieldCategory = new string[]
            {
                "CategoryMaker",
                "FieldOne",
                "FieldTwo",
                "FieldThree",
                "FieldFour",
            };
            fieldValue = new string[]
            {
                Properties.Settings.Default.userEmail,
                safe.Fields[0],
                safe.Fields[1],
                safe.Fields[2],
                safe.Fields[3]
            };

            break;

        case 5:

            fieldCategory = new string[]
            {
```

```
            "CategoryMaker",
            "FieldOne",
            "FieldTwo",
            "FieldThree",
            "FieldFour",
            "FieldFive"
        };
        fieldValue = new string[]
        {
            Properties.Settings.Default.userEmail,
            safe.Fields[0],
            safe.Fields[1],
            safe.Fields[2],
            safe.Fields[3],
            safe.Fields[4]
        };

        break;

    case 6:

        fieldCategory = new string[]
        {
            "CategoryMaker",
            "FieldOne",
            "FieldTwo",
            "FieldThree",
            "FieldFour",
            "FieldFive",
            "FieldSix"
        };
        fieldValue = new string[]
        {
            Properties.Settings.Default.userEmail,
            safe.Fields[0],
```

```
                    safe.Fields[1],
                    safe.Fields[2],
                    safe.Fields[3],
                    safe.Fields[4],
                    safe.Fields[5]
                };

                break;
            }

            // Заносим в базу категорию
            DateBase.InsertData("Category", fieldCategory, fieldValue);

            // Собираем данные
            int catID = WorkingScripts.DateBase.GetLastIDFromCategory();
            string[] value = new string[]
            {
                safe.SafeName,
                catID.ToString()
            };
            DateBase.InsertData("Safe", new string[] { "SafeName", "Category_ID" }, value);

            // Собираем данные
            int safeID = DateBase.GetLastIDFromSafe();
            int userID =
WorkingScripts.DateBase.GetUserID(Properties.Settings.Default.userEmail);
            value = new string[]
            {
                userID.ToString(),
                safeID.ToString(),
                true.ToString()
            };
            DateBase.InsertData("User-Safe", new string[] { "User_ID", "Safe_ID",
"UserIsCreator" }, value);
        }
```

```
        #endregion

        #region Панель сейфа

        /// <summary>
        /// Показывавает сейф изнутри
        /// </summary>
        /// <param name="safe">Объект класса</param>
        public void ShowSafeItemView(Safe safe)
        {
            // Ставим название сейфа
            safeItemView_SafeName.Text = safe.SafeName;
            // Ставим создателя сейфа
            safeItemView_SafeCreatorPerson.Text = safe.Creator;
            // Чистим поля
            HideFieldInSafeView();
            // Заполняем поля
            FillFieldInSafeView(safe.Fields);
            // Показываем панель
            safeItemView.Visible = true;
            // Обновляем хук активного сейфа
            this.activeSafe = safe;
            // Заполянем список папок в активном сейфе
            FillFolderList();
            // Показываем количество участников
            // REF: Заполнение количества участников в сейфе
            safeItemView_MembersCount.Text =
DateBase.GetCountMembersInSafe(safe.SafeID).ToString();

            // Если пользователь не создатель сейфа
            if (safe.Creator != Properties.Settings.Default.userEmail)
                safeItemView_Act.Visible = false;
            else
                safeItemView_Act.Visible = true;
```

```
}

/// <summary>
/// Заполняет поля панели
/// </summary>
/// <param name="list">список полей</param>
public void FillFieldInSafeView(List<string> list)
{
    for (int i = 1; i <= list.Count; i++)
    {
        safeItemView.Controls["safeItemView_Field" + i].Text = list[i - 1];
        safeItemView.Controls["safeItemView_Field" + i].Visible = true;
    }
}

/// <summary>
/// Скрывает поля в сейфе
/// </summary>
public void HideFieldInSafeView()
{
    for (int i = 1; i < 7; i++)
    {
        safeItemView.Controls["safeItemView_Field" + i].Visible = false;
    }
}

/// <summary>
////Передает фокус на сейф
/// </summary>
/// <param name="safe"></param>
public void FocusedSafe(Safe safe)
{
    // Чистим список элементов на панели отображения итемов сейфа
    safeItemView_ItemPanel.Controls.Clear();
    // Показываем сейф
```

```
            ShowSafeItemView(safe);
            foreach (var item in safeList.Controls)
            {
                if ((item as Label).Text == safe.SafeName)
                {
                    (item as Label).ForeColor = Color.FromArgb(20, 131, 59);
                }
            }
        }

        /// <summary>
        /// Возвращает шрифт для сейфа
        /// </summary>
        private Font GetFontForFolder() => new Font(pr.Families[3], 11);

        /// <summary>
        ///  Получаем числовой показателя для следующего лейбла / Объекта сейфа
        /// </summary>
        private int GetLocationForFolder()
        {
            // Если количество элементов в списке контролов
            // равно нулю, то возвращаем позицию первого контрола
            if (safeItemView_ItemPanel.Controls.Count == 0)
                return 5;
            // Иначе возвращаем позицию в результате расчета
            // позиции последнего контрола в списке
            else
                return
safeItemView_ItemPanel.Controls[safeItemView_ItemPanel.Controls.Count - 1].Location.Y + 50;
        }

        /// <summary>
        /// Получает количество папок
        /// </summary>
        private int GetFolderCount()
```

```
        {
            // REF: Аналогия для элементов
            int i = 0;
            foreach (var item in safeItemView_ItemPanel.Controls)
            {
                if ((item as Panel).Controls["Mark"].Text == "folder")
                    i++;
            }
            return i;
        }

        /// <summary>
        /// Показывает панель создания папки
        /// </summary>
        public void ShowAddFolderForm()
        {
            var newFolder = new CustomControls.AddFolder()
            {
                Owner = this
            }.ShowDialog();
        }

        /// <summary>
        /// Заполняет список папок в активном сейфе
        /// </summary>
        public void FillFolderList()
        {
            // Заполняем список папок
            // Передаем в параметры запроса объект активного сейфа
            this.folderCollectionInActiveSafe =
WorkingScripts.DateBase.GetFolderList(activeSafe);

            // TODO: Переделать, тк будут очищаться и итемы
            safeItemView_ItemPanel.Controls.Clear();
```

```
        // Создаем контролы на панели
        foreach (var item in folderCollectionInActiveSafe)
        {
            safeItemView_ItemPanel.Controls.Add(
                NewFolder.CreateNewFolder(safeItemView_ItemPanel,
                GetFolderCount(), GetFontForFolder(), item, GetLocationForFolder())
            );
        }
    }

    /// <summary>
    /// Удаляет папку
    /// </summary>
    public void DeleteFolder(int safeID, int folderID)
    {
        DateBase.DeleteFolder(safeID, folderID);
    }

    /// <summary>
    /// Создает папку
    /// </summary>
    /// <param name="folder"></param>
    public void CreateFolder(Folder folder)
    {
        safeItemView_ItemPanel.Controls.Add(
            NewFolder.CreateNewFolder(safeItemView_ItemPanel,
            GetFolderCount(), GetFontForFolder(), folder, GetLocationForFolder())
        );

        // Собираем и добавляем данные в базу
        var values = new string[]
        {
            folder.FolderName,
            DateBase.GetUserID(Properties.Settings.Default.userEmail).ToString()
        };
```

```
                DateBase.InsertData("Folder", new string[] { "FolderName", "User_ID" }, values);
                values = new string[]
                {
                    DateBase.GetLastIDFromFolder().ToString(),
                    activeSafe.SafeID.ToString()
                };
                DateBase.InsertData("Folder-Safe", new string[] { "Folder_ID", "Safe_ID" },
values);

                // Добавляем папку в список
                folderCollectionInActiveSafe.Add(folder);
            }

            /// <summary>
            /// Показывает меню сейфа
            /// </summary>
            public void OpenSafeMenu()
            {
                // Выдаем права на форму
                safeMenu.Owner = this;
                safeMenu.BringToFront();
                // Показываем меню
                safeMenu.Show();
                // Меняем позицию меню
                safeMenu.Location = new Point(this.Location.X +
                    (safeItemView.Location.X + safeItemView_Act.Location.X +
safeItemView_Act.Width - safeMenu.Width),
                    this.Location.Y + (safeItemView.Location.Y + safeItemView_Act.Location.Y +
30));
            }

            /// <summary>
            /// Скрывает меню сейфа
            /// </summary>
            public void CloseSafeMenu()
```

```
{
    safeMenu.Hide();
    safeMenuIsOpen = false;
    countOpenSafeMenu = 0;
}

/// <summary>
/// Возвращает идентификатор сейфа
/// </summary>
public int GetSafeID() => this.activeSafe.SafeID;

/// <summary>
/// Удаляет сейф
/// </summary>
/// <param name="safe"></param>
public void DeleteSafe(Safe safe)
{
    // Удаляем из БД
    DateBase.DeleteSafe(safe);

    // Скрываем панель сейфа
    foreach (var item in safeList.Controls)
    {
        (item as Label).ForeColor = Color.White;
    }
    safeItemView.Visible = false;
    // Заполняем заново
    FillSafeList();
}

/// <summary>
/// Открывает панель редактирования сейфа
/// </summary>
/// <param name="safe"></param>
public void OpenEditSafeMenu(Safe safe)
```

```
        {
            // Скрываем меню сейфа
            CloseSafeMenu();

            // Показываем панель редактирования сейфа
            editSafe = new EditSafe(safe);
            editSafe.Owner = this;
            editSafe.ShowDialog();
        }

        /// <summary>
        /// Показывает меню участников сейфа
        /// </summary>
        public void OpenMembersMenu()
        {
            members.Owner = this;
            members.BringToFront();
            // Показываем меню
            members.Show();
            //// Меняем позицию меню
            members.Location = new Point(this.Location.X +
                (safeItemView.Location.X + safeItemView_Members.Location.X),
                this.Location.Y + (safeItemView.Location.Y +
safeItemView_Members.Location.Y + 30));
        }

        /// <summary>
        /// Скрывает меню участников сейфа
        /// </summary>
        public void CloseMembersMenu()
        {
            members.Hide();
            membersMenuIsOpen = false;
            countOpenSafeMembers = 0;
        }
```

```
        /// <summary>
        /// Открывает форму приглашения
        /// </summary>
        public void OpenInviteForm()
        {
            var invite = new InviteForm();
            invite.Owner = this;
            invite.ShowDialog();
        }

        /// <summary>
        /// Получает количество элементов
        /// </summary>
        private int GetItemCount()
        {
            // REF: Аналогия для элементов
            int i = 0;
            foreach (var item in safeItemView_ItemPanel.Controls)
            {
                if ((item as Panel).Controls["Mark"].Text == "item")
                    i++;
            }
            return i;
        }


        public void CreateItem()
        {
            safeItemView_ItemPanel.Controls.Add(
                NewItem.CreateNewItem(safeItemView_ItemPanel,
                GetItemCount(), GetFontForFolder(), new string[] {
Properties.Settings.Default.userEmail, "Отчет от 21.02.2012"}, GetLocationForFolder())
            );
```

```
                //// Собираем и добавляем данные в базу
                //var values = new string[]
                //{
                //    folder.FolderName,
                //    DateBase.GetUserID(Properties.Settings.Default.userEmail).ToString()
                //};
                //DateBase.InsertData("Folder", new string[] { "FolderName", "User_ID" }, values);
                //values = new string[]
                //{
                //    DateBase.GetLastIDFromFolder().ToString(),
                //    activeSafe.SafeID.ToString()
                //};
                //DateBase.InsertData("Folder-Safe", new string[] { "Folder_ID", "Safe_ID" },
values);

                //// Добавляем папку в список
                //folderCollectionInActiveSafe.Add(folder);
            }

            #endregion

            #endregion

            #region Построение интефейса

            /// <summary>
            /// Строит верхнее меню
            /// </summary>
            private void CreateTopMenu()
            {
                // Ставим плейсхолдер
                Animation.Placeholder.addPlaceholder(searchText, "Поиск", Color.FromArgb(210,
210, 210), Color.FromArgb(120, 120, 120));
                // Событие при изменении текста в поле поиска
                searchText.TextChanged += (f, a) =>
```

```
        {
            // Если нет слова запроса, то скрываем кнопочку очистки текста
            if (searchText.Text == "Поиск" || searchText.Text == string.Empty)
            {
                textLogoMain.Focus();
                searchTextBreak.Visible = false;
            }
            // Если нет слова запроса, то показываем кнопочку очистки текста
            else
                searchTextBreak.Visible = true;
        };
        // Событие при клике на поле поиска и передаем на него фокус
        searchText.Click += (f, a) =>
        {
            // Закрываем все открытые окна
            ClearScreenOfWindow();

            searchText.Focus();
        };
        // Событие при клике на кнопочку очитски поля поиска
        searchTextBreak.Click += (f, a) =>
        {
            // Передаем фокус на поле
            searchText.Focus();
            // Скрываем кнопку
            searchTextBreak.Visible = false;
            // Обнуляем поле текста
            searchText.Text = string.Empty;
        };
        // Показываем email
        if (Properties.Settings.Default.userEmail != string.Empty &&
Properties.Settings.Default.userPassword != string.Empty)
            // Записываем email
            WriteEmail();
        // Заполняем строку уведомлений
```

```
//WriteNotification();

// Событие при клике по иконке и надписи email'a
emailText.Click += (f, a) =>
{
    // Закрываем все открытые окна
    ClearScreenOfWindow();
    // Передаем фокус
    emailText.Focus();

    // Если меню активно
    if (!emailRightSideMenu)
    {
        // Меню активно
        emailRightSideMenu = true;
        // Меняем иконку у email'a
        emailIcon.Image = Properties.Resources.emailIconGreenUp;
        // Меняем цвет у email'a
        emailText.ForeColor = Color.FromArgb(29, 185, 84);
        // Хук на количество экземпляров меню
        countOpenMenu++;

        // Показываем меню
        if (countOpenMenu == 1)
            // Показываем меню
            OpenRightSideMenu();
    }
    // Если меню не активно
    else
    {
        // Меню не активно
        emailRightSideMenu = false;
        // Меняем иконку у email'a
        emailIcon.Image = Properties.Resources.emailIconGrayDown;
        // Меняем цвет у email'a
```

```
            emailText.ForeColor = Color.FromArgb(120, 120, 120);
            // Хук на количество экземпляров меню
            countOpenMenu--;
            // Скрываем меню
            CloseRightSideMenu();
        }
    };
    emailIcon.Click += (f, a) =>
    {
        // Закрываем все открытые окна
        ClearScreenOfWindow();
        // Передаем фокус
        emailText.Focus();

        if (!emailRightSideMenu)
        {
            // Меню активно
            emailRightSideMenu = true;
            // Меняем иконку у email'a
            emailIcon.Image = Properties.Resources.emailIconGreenUp;
            // Меняем цвет у email'a
            emailText.ForeColor = Color.FromArgb(29, 185, 84);
            // Хук на количество экземпляров меню
            countOpenMenu++;

            // Показываем меню
            if (countOpenMenu < 2)
                // Показываем меню
                OpenRightSideMenu();
        }
        else
        {
            // Меню не активно
            emailRightSideMenu = false;
            // Меняем иконку у email'a
```

```
            emailIcon.Image = Properties.Resources.emailIconGrayDown;
            // Меняем цвет у email'a
            emailText.ForeColor = Color.FromArgb(120, 120, 120);
            // Хук на количество экземпляров меню
            countOpenMenu--;
            // Скрываем меню
            CloseRightSideMenu();
        }
    };
    // Событие при наведении на иконку и надпись email'a
    emailText.MouseEnter += (f, a) =>
    {
        // Если меню не активно
        if (!emailRightSideMenu)
        {
            // Меняем цвет текста
            emailText.ForeColor = Color.FromArgb(29, 185, 84);
            // Меняем иконку
            emailIcon.Image = Properties.Resources.emailIconGreenDown;
        }
    };
    emailText.MouseLeave += (f, a) =>
    {
        // Если меню не активно
        if (!emailRightSideMenu)
        {
            // Меняем цвет текста
            emailText.ForeColor = Color.FromArgb(120, 120, 120);
            // Меняем иконку
            emailIcon.Image = Properties.Resources.emailIconGrayDown;
        }
    };
    emailText.MouseDown += (f, a) =>
    {
        emailText.ForeColor = Color.FromArgb(70, 70, 70);
```

```
    if (emailRightSideMenu)
        emailIcon.Image = Properties.Resources.emailIconDarkGrayUp;
    else
        emailIcon.Image = Properties.Resources.emailIconDarkGrayDown;
};
emailText.MouseUp += (f, a) =>
{
    if (emailRightSideMenu)
        emailText.ForeColor = Color.FromArgb(29, 185, 84);
    else
        emailText.ForeColor = Color.FromArgb(120, 120, 120);
};

emailIcon.MouseEnter += (f, a) =>
{
    // Если меню не активно
    if (!emailRightSideMenu)
    {
        // Меняем цвет текста
        emailText.ForeColor = Color.FromArgb(29, 185, 84);
        // Меняем иконку
        emailIcon.Image = Properties.Resources.emailIconGreenDown;
    }
};
emailIcon.MouseLeave += (f, a) =>
{
    // Если меню не активно
    if (!emailRightSideMenu)
    {
        // Меняем цвет текста
        emailText.ForeColor = Color.FromArgb(120, 120, 120);
        // Меняем иконку
        emailIcon.Image = Properties.Resources.emailIconGrayDown;
    }
};
```

```
emailIcon.MouseDown += (f, a) =>
{
    emailText.ForeColor = Color.FromArgb(70, 70, 70);
    if (emailRightSideMenu)
        emailIcon.Image = Properties.Resources.emailIconDarkGrayUp;
    else
        emailIcon.Image = Properties.Resources.emailIconDarkGrayDown;
};
emailIcon.MouseUp += (f, a) =>
{
    if (emailRightSideMenu)
        emailText.ForeColor = Color.FromArgb(29, 185, 84);
    else
        emailText.ForeColor = Color.FromArgb(120, 120, 120);
};

notificationImage.MouseEnter += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationGreen;
    notificationCount.Back = Color.FromArgb(29, 185, 84);
    notificationCount.Invalidate();
};
notificationImage.MouseLeave += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationGray;
    notificationCount.Back = Color.FromArgb(120, 120, 120);
    notificationCount.Invalidate();
};
notificationImage.MouseDown += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationDarkGray;
    notificationCount.Back = Color.FromArgb(70, 70, 70);
    notificationCount.Invalidate();
};
notificationImage.MouseUp += (f, a) =>
```

```
{
    notificationImage.Image = Properties.Resources.notificationGreen;
    notificationCount.Back = Color.FromArgb(29, 185, 84);
    notificationCount.Invalidate();
};
notificationCount.MouseEnter += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationGreen;
    notificationCount.Back = Color.FromArgb(29, 185, 84);
    notificationCount.Invalidate();
};
notificationCount.MouseLeave += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationGray;
    notificationCount.Back = Color.FromArgb(120, 120, 120);
    notificationCount.Invalidate();
};
notificationCount.MouseDown += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationDarkGray;
    notificationCount.Back = Color.FromArgb(70, 70, 70);
    notificationCount.Invalidate();
};
notificationCount.MouseUp += (f, a) =>
{
    notificationImage.Image = Properties.Resources.notificationGreen;
    notificationCount.Back = Color.FromArgb(29, 185, 84);
    notificationCount.Invalidate();
};

notificationCount.Click += (f, a) =>
{
    // Если меню активно
    if (!notificationMenuIsOpen)
    {
```

```
            // Меню активно
            notificationMenuIsOpen = true;
            // Хук на количество экземпляров меню
            countOpenNotificationMenu++;

            // Показываем меню
            if (countOpenNotificationMenu == 1)
                // Показываем меню
                OpenNotificationForm();
        }
        // Если меню не активно
        else
        {
            // Меню не активно
            notificationMenuIsOpen = false;
            // Хук на количество экземпляров меню
            countOpenNotificationMenu--;
            // Скрываем меню
            CloseNotificationForm();
        }
    };
    notificationImage.Click += (f, a) =>
    {
        // Если меню активно
        if (!notificationMenuIsOpen)
        {
            // Меню активно
            notificationMenuIsOpen = true;
            // Хук на количество экземпляров меню
            countOpenNotificationMenu++;

            // Показываем меню
            if (countOpenNotificationMenu == 1)
                // Показываем меню
                OpenNotificationForm();
```

```
            }
            // Если меню не активно
            else
            {
                // Меню не активно
                notificationMenuIsOpen = false;
                // Хук на количество экземпляров меню
                countOpenNotificationMenu--;
                // Скрываем меню
                CloseNotificationForm();
            }
        };
    }

    /// <summary>
    /// Строим боковое левое меню
    /// </summary>
    private void CreateLeftSide()
    {
        // Если список сейфов скрыт
        if (Properties.Settings.Default.hideSafeList)
        {
            // Меняем иконку
            hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGrayRight;
            // Скрываем
            safeList.Visible = false;
        }
        // Делаем КАПС
        leftSideLogoLabel.Text = leftSideLogoLabel.Text.ToUpper();

        // Событие при наведении на лого-лейбл на левой панели
        leftSideLogoLabel.MouseEnter += (f, a) =>
        {
            // Меняем цвет текста
            leftSideLogoLabel.ForeColor = Color.FromArgb(29, 185, 84);
```

```
    // Если мы скрыли список сейфов
    if (Properties.Settings.Default.hideSafeList)
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenRight;
    else
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenDown;

};
leftSideLogoLabel.MouseLeave += (f, a) =>
{
    // Меняем цвет текста
    leftSideLogoLabel.ForeColor = Color.FromArgb(120, 120, 120);
    // Если мы скрыли список сейфов
    if (Properties.Settings.Default.hideSafeList)
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGrayRight;
    else
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGrayDown;
};

hideSafeListIcon.MouseEnter += (f, a) =>
{
    // Меняем цвет текста
    leftSideLogoLabel.ForeColor = Color.FromArgb(29, 185, 84);
    // Если мы скрыли список сейфов
    if (Properties.Settings.Default.hideSafeList)
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenRight;
    else
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenDown;
};
hideSafeListIcon.MouseLeave += (f, a) =>
{
    // Меняем цвет текста
    leftSideLogoLabel.ForeColor = Color.FromArgb(120, 120, 120);
    // Если мы скрыли список сейфов
    if (Properties.Settings.Default.hideSafeList)
        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGrayRight;
```

```
                else
                    hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGrayDown;
                };

                leftSideLogoLabel.MouseDown += (f, a) =>
                {
                    leftSideLogoLabel.ForeColor = Color.FromArgb(70, 70, 70);

                    if (Properties.Settings.Default.hideSafeList)
                        hideSafeListIcon.Image =
Properties.Resources.hideSafeListIconDarkGrayRight;
                    else
                        hideSafeListIcon.Image =
Properties.Resources.hideSafeListIconDarkGrayDown;
                };
                leftSideLogoLabel.MouseUp += (f, a) =>
                {
                    leftSideLogoLabel.ForeColor = Color.FromArgb(29, 185, 84);

                    if (Properties.Settings.Default.hideSafeList)
                        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenRight;
                    else
                        hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenDown;
                };

                // Событие при клике на лого-лейбл на левой панели
                leftSideLogoLabel.Click += (f, a) =>
                {
                    // Закрываем все открытые окна
                    ClearScreenOfWindow();

                    leftSideLogoLabel.Focus();

                    // Если мы скрыли список сейфов
                    if (Properties.Settings.Default.hideSafeList)
```

```
            {
                // Сохраняем параметр
                Properties.Settings.Default.hideSafeList = false;
                Properties.Settings.Default.Save();

                // Показываем список сейфов
                safeList.Visible = true;
                hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenDown;
            }
            else
            {
                // Сохраняем параметр
                Properties.Settings.Default.hideSafeList = true;
                Properties.Settings.Default.Save();

                // Скрываем список сейфов
                safeList.Visible = false;
                hideSafeListIcon.Image = Properties.Resources.hideSafeListIconGreenRight;
            }
        };

        // Событие при наведении на иконку создания сейфа
        createSafeIcon.MouseEnter += (f, a) => createSafeIcon.Image =
Properties.Resources.iconPlusGreen;
        createSafeIcon.MouseLeave += (f, a) => createSafeIcon.Image =
Properties.Resources.iconPlusGray;
        createSafeIcon.MouseDown += (f, a) => createSafeIcon.Image =
Properties.Resources.iconPlusDarkGray;
        createSafeIcon.MouseUp += (f, a) => createSafeIcon.Image =
Properties.Resources.iconPlusGreen;

        // Событие при клике на иконку создания сейфа
        createSafeIcon.Click += (f, a) =>
        {
            createSafeIcon.Focus();
```

```
    OpenSafePanel();
};

// Событие при наведении на кнопку создания сейфа
createSafeButton.MouseEnter += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(29, 185, 84);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGreen;
};
createSafeButton.MouseLeave += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(120, 120, 120);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGray;
    leftSideSeparator2.BackColor = Color.FromArgb(40, 40, 40);
};

createSafeButtonLabel.MouseEnter += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(29, 185, 84);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGreen;
};
createSafeButtonLabel.MouseLeave += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(120, 120, 120);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGray;
};

createSafeButtonIcon.MouseEnter += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(29, 185, 84);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGreen;
};
createSafeButtonIcon.MouseLeave += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(120, 120, 120);
```

```
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGray;
};

// Событие при нажатии на кнопку создания сейфа
createSafeButton.MouseDown += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(70, 70, 70);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusDarkGray;
    leftSideSeparator2.BackColor = Color.FromArgb(30, 30, 30);
};
createSafeButtonLabel.MouseDown += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(70, 70, 70);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusDarkGray;
    leftSideSeparator2.BackColor = Color.FromArgb(30, 30, 30);
};
createSafeButtonIcon.MouseDown += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(70, 70, 70);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusDarkGray;
    leftSideSeparator2.BackColor = Color.FromArgb(30, 30, 30);
};
createSafeButton.MouseUp += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(29, 185, 84);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGreen;
    leftSideSeparator2.BackColor = Color.FromArgb(40, 40, 40);
};
createSafeButtonLabel.MouseUp += (f, a) =>
{
    createSafeButtonLabel.ForeColor = Color.FromArgb(29, 185, 84);
    createSafeButtonIcon.Image = Properties.Resources.iconPlusGreen;
    leftSideSeparator2.BackColor = Color.FromArgb(40, 40, 40);
};
createSafeButtonIcon.MouseUp += (f, a) =>
```

```
    {
        createSafeButtonLabel.ForeColor = Color.FromArgb(29, 185, 84);
        createSafeButtonIcon.Image = Properties.Resources.iconPlusGreen;
        leftSideSeparator2.BackColor = Color.FromArgb(40, 40, 40);
    };

    // Событие при клике на снопку создания сейфа
    createSafeButton.Click += (f, a) =>
    {
        createSafeButton.Focus();
        OpenSafePanel();
    };
    createSafeButtonLabel.Click += (f, a) =>
    {
        createSafeButton.Focus();
        OpenSafePanel();
    };
    createSafeButtonIcon.Click += (f, a) =>
    {
        createSafeButton.Focus();
        OpenSafePanel();
    };
}

/// <summary>
////Строим центр
/// </summary>
private void CreateCenter()
{
    // Событие наведения и нажатия кнопки создания папки
    safeItemView_AddFolder.MouseEnter += (f, a) =>
        safeItemView_AddFolder.Image = Properties.Resources.addFolderGreen;
    safeItemView_AddFolder.MouseLeave += (f, a) =>
        safeItemView_AddFolder.Image = Properties.Resources.addFolderGray;
    safeItemView_AddFolder.MouseDown += (f, a) =>
```

```
    safeItemView_AddFolder.Image = Properties.Resources.addFolderDarkGray;
//safeItemView_AddFolder.MouseUp += (f, a) =>
//    safeItemView_AddFolder.Image = Properties.Resources.addFolderGreen;

// Событие наведения и нажатия кнопки действия (Act)
safeItemView_Act.MouseEnter += (f, a) =>
    safeItemView_Act.Image = Properties.Resources.actGreen;
safeItemView_Act.MouseLeave += (f, a) =>
    safeItemView_Act.Image = Properties.Resources.actGray;
safeItemView_Act.MouseDown += (f, a) =>
    safeItemView_Act.Image = Properties.Resources.actDarkGreen;
safeItemView_Act.MouseUp += (f, a) =>
    safeItemView_Act.Image = Properties.Resources.actGreen;

// Наведение и нажатие на кнопку "Добавить" на панели элементов сейфа
safeItemView_AddItem.MouseEnter += (f, a) =>
{
    safeItemView_AddItem.Back = Color.FromArgb(30, 215, 96);
};
safeItemView_AddItem.MouseLeave += (f, a) =>
{
    safeItemView_AddItem.Back = Color.FromArgb(29, 185, 84);
};
safeItemView_AddItem.MouseDown += (f, a) =>
{
    safeItemView_AddItem.Back = Color.FromArgb(20, 131, 59);
};
safeItemView_AddItem.MouseUp += (f, a) =>
{
    safeItemView_AddItem.Back = Color.FromArgb(30, 215, 96);
};

safeItemView_Hide.MouseEnter += (f, a) =>
{
    safeItemView_Hide.Image = Properties.Resources.closePanelViewGreen;
```

```
};
safeItemView_Hide.MouseLeave += (f, a) =>
{
   safeItemView_Hide.Image = Properties.Resources.closePanelViewGray;
};
safeItemView_Hide.MouseDown += (f, a) =>
{
   safeItemView_Hide.Image = Properties.Resources.closePanelViewDarkGray;
};
safeItemView_Hide.MouseUp += (f, a) =>
{
   safeItemView_Hide.Image = Properties.Resources.closePanelViewGreen;
};

safeItemView_Members.MouseEnter += (f, a) =>
{
   safeItemView_Members.Image = Properties.Resources.membersGreen;
   safeItemView_MembersCount.ForeColor = Color.FromArgb(30, 215, 96);
};
safeItemView_Members.MouseLeave += (f, a) =>
{
   safeItemView_Members.Image = Properties.Resources.membersGray;
   safeItemView_MembersCount.ForeColor = Color.FromArgb(120, 120, 120);
};
safeItemView_Members.MouseDown += (f, a) =>
{
   safeItemView_Members.Image = Properties.Resources.membersDarkGray;
   safeItemView_MembersCount.ForeColor = Color.FromArgb(70, 70, 70);
};
safeItemView_Members.MouseUp += (f, a) =>
{
   safeItemView_Members.Image = Properties.Resources.membersGreen;
   safeItemView_MembersCount.ForeColor = Color.FromArgb(30, 215, 96);
};
safeItemView_MembersCount.MouseEnter += (f, a) =>
```

```
{
    safeItemView_Members.Image = Properties.Resources.membersGreen;
    safeItemView_MembersCount.ForeColor = Color.FromArgb(30, 215, 96);
};
safeItemView_MembersCount.MouseLeave += (f, a) =>
{
    safeItemView_Members.Image = Properties.Resources.membersGray;
    safeItemView_MembersCount.ForeColor = Color.FromArgb(120, 120, 120);
};
safeItemView_MembersCount.MouseDown += (f, a) =>
{
    safeItemView_Members.Image = Properties.Resources.membersDarkGray;
    safeItemView_MembersCount.ForeColor = Color.FromArgb(70, 70, 70);
};
safeItemView_MembersCount.MouseUp += (f, a) =>
{
    safeItemView_Members.Image = Properties.Resources.membersGreen;
    safeItemView_MembersCount.ForeColor = Color.FromArgb(30, 215, 96);
};

safeItemView_AddFolder.Click += (f, a) => ShowAddFolderForm();
safeItemView_Hide.Click += (f, a) =>
{
    foreach (var item in safeList.Controls)
    {
        (item as Label).ForeColor = Color.White;
    }

    countOpenSafeMenu = 0;
    safeMenuIsOpen = false;
    CloseSafeMenu();

    safeItemView.Visible = false;
};
safeItemView_Act.Click += (f, a) =>
```

```
{
    // Если меню активно
    if (!safeMenuIsOpen)
    {
        // Меню активно
        safeMenuIsOpen = true;
        // Хук на количество экземпляров меню
        countOpenSafeMenu++;

        // Показываем меню
        if (countOpenSafeMenu == 1)
            // Показываем меню
            OpenSafeMenu();
    }
    // Если меню не активно
    else
    {
        // Меню не активно
        safeMenuIsOpen = false;
        // Хук на количество экземпляров меню
        countOpenSafeMenu--;
        // Скрываем меню
        CloseSafeMenu();
    }
};
safeItemView_MembersCount.Click += (f, a) =>
{

    // Если меню активно
    if (!membersMenuIsOpen)
    {
        // Меню активно
        membersMenuIsOpen = true;
        // Хук на количество экземпляров меню
        countOpenSafeMembers++;
```

```
            // Показываем меню
            if (countOpenSafeMembers == 1)
                // Показываем меню
                OpenMembersMenu();
        }
        // Если меню не активно
        else
        {
            // Меню не активно
            membersMenuIsOpen = false;
            // Хук на количество экземпляров меню
            countOpenSafeMembers--;
            // Скрываем меню
            CloseMembersMenu();
        }
    };
    safeItemView_Members.Click += (f, a) =>
    {

        // Если меню активно
        if (!membersMenuIsOpen)
        {
            // Меню активно
            membersMenuIsOpen = true;
            // Хук на количество экземпляров меню
            countOpenSafeMembers++;

            // Показываем меню
            if (countOpenSafeMembers == 1)
                // Показываем меню
                OpenMembersMenu();
        }
        // Если меню не активно
        else
```

```
                {
                    // Меню не активно
                    membersMenuIsOpen = false;
                    // Хук на количество экземпляров меню
                    countOpenSafeMembers--;
                    // Скрываем меню
                    CloseMembersMenu();
                }
            };
            safeItemView_AddItem.Click += (f, a) =>
            {
                CreateItem();
                new AddItem()
                {
                    Owner = this
                }.ShowDialog();
            };
        }
#endregion
```