МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка программного средства планировщик дел и задач – QuickShedle

Выполнил студент Бондарчик Евгений Николаевич

(Ф.И.О.)

Руководитель проекта асс. Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой д.к.т.н., Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

ст. преп. Блинова Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтроллер асс. Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2017

**Реферат**

Пояснительная записка курсового проекта содержит 28 страниц, 25 рисунков, 5 источников литературы, 1 приложение.

ЗАДАЧА, БАЗА ДАННЫХ, УВЕДОМЛЕНИЕ, ТАЙМЕР, ООП, C#, АВТОРИЗАЦИЯ.

Основной целью курсового проекта является: разработка программного средства планировщик дел и задач – «QuickShedle».

Пояснительная записка состоит из введения, четырёх разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается алгоритм решения, постановка задачи и обзор прототипов.

Во втором разделе рассматривается архитектура базы данных и программного средства.

В третьем разделе представлен процесс и результаты разработки итогового приложения.

В четвёртом разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе разработки программного средства.

**Содержание**

[Введение](#page3) 6

[1.Постановка задачи](#page4) 7

[1.1 Алгоритмы решения](#page4) 7

[1.2 Обзор прототипов](#page5) 8

[2.Разработка архитектуры проекта](#page8) 10

[2.1 Обобщенная структура](#page8) 10

[2.1 Диаграммы UML, взаимосвязь всех компонентов](#page8) 11

[3.Разработка функциональной модели и модели данных ПС](#page10) 12

[3.1 Модель базы данных](#page10) 12

[3.2 Выполняемые функции](#page11) 13

[3.2.1 Регистрация и авторизация](#регист) 13

[3.2.2 Добавление заметок и задач 1](#задачи)4

[3.2.3 Уведомление о запланированных задачах](#уведомл) 14

[3.2.4 Просмотр списка задач, отображение статистики и настройка приложения](#настр)

15

[4. Руководство пользователя](#page15) [1](#page15)7

[Заключение](#page23) 21

[Список используемых источников](#page24) 22

[Приложение А](#page25) 23

[Приложение Б 30](#page30)

**Введение**

Планировщик задач – служебный инструмент пользовательского интерфейса для автоматизации повторяющихся задач. Возможности планировщика задач шире, чем подготовка расписания для запуска сценариев, программ и документов в определенное время. С помощью данного инструмента можно отображать окна сообщений, отправлять сообщения по электронной почте, предоставлять статистику по проделанной работе, управлять личным расписанием и планировать задачи. Кроме того, можно назначить график запуска задач в зависимости от некоторых событий, в том числе запуска компьютера, регистрации в системе и даже пользовательских событий.

Сейчас на первый план выходит умение эффективно распоряжаться своим временем. Уметь распределять его так, чтобы в первую очередь выполнялись приоритетные дела.

Прогресс ушел далеко вперед, и сегодня многие документы стали цифровыми, а необходимость в бумажных ежедневниках отпала. Поэтому люди прибегают к использованию электронных планировщиков задач.

Электронные планировщики задач располагают удобными и интуитивно-понятными для пользователей интерфейсами, объектами сценариев для разработчиков. Таким образом достигается непревзойденная гибкость как при создании, так и при настройке задач.

Темой курсового проекта является разработка программного средства «Планировщик дел и задач – QuickShedle». Данное ПО должно позволить пользователям надежно хранить свои учетные записи. Пользователю будет предложена возможность хранения быстрых заметок и задач в таблицах, которые доступны только ему. Программа автоматически, при наличии не выполненных задач или не прочитанных заметок, уведомляет пользователя с возможностью повтора.

Целью курсового проекта является проектирование и реализация программного средства «QuickShedle».

**1.Постановка задачи**

Основной задачей курсового проекта является разработка программного средства, позволяющего пользователю хранить информацию в централизованной базе данных, а так же уведомлять о запланированных задачах.

Функционально ПС должно выполнять следующие задачи:

* составлять рабочие планы;
* сохранять рабочую информацию в централизованной базе данных;
* уведомлять и напоминать о запланированных задачах;
* предоставлять статистику о проделанной работе;
* настраивать параметры работы приложения.
  1. **Алгоритмы решения**

В разработке приложения были использованы нижеперечисленные технологии:

* ADO.NET;
* Microsoft SQL Server;
* WPF.

ADO.NET (ActiveX Data Object для .NET) – технология, предоставляющая доступ к данным для приложений, основанных на Microsoft .NET. Является не развитием более ранней технологии ADO, а самостоятельной технологией, частью фреймворка .NET. В отличие от классической ADO, которая была в основном предназначена для тесно связанных клиент-серверных систем, ADO.NET больше нацелена на автономную работу с помощью объектов DataSet. Эти типы представляют локальные копии любого количества взаимосвязанных таблиц данных, каждая из которых содержит набор строк и столбцов. Объекты DataSet позволяют вызывающей сборке (наподобие веб-страницы или программы, выполняющейся на настольном компьютере) работать с содержимым DataSet, изменять его, не требуя подключения к источнику данных, и отправлять обратно блоки измененных данных для обработки с помощью соответствующего адаптера данных. Фундаментальное различие между классической ADO и ADO.NET состоит в том, что ADO.NET является управляемой кодовой библиотекой, и, значит, подчиняется тем же правилам, что и любая управляемая библиотека. Типы, составляющие ADO.NET, используют протокол управления памятью CLR, принадлежат к той же системе типов (классы, интерфейсы, перечисления, структуры и делегаты), и доступ к ним возможен с помощью любого языка .NET. Классы ADO.NET находятся в сборке System.Data.dll

Microsoft SQL Server – система управления реляционными базами данных (РСУБД), 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 базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

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.

Windows Presentation Foundation (WPF)[2] – система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

**1.2 Обзор прототипов**

Программное средство *«TickTick».*

TickTick позволяет создавать множество списков задач. Это базовая функция, которая доступна во всех подобных приложениях. Синхронизирует работу между устройствами и имеет веб-версию.

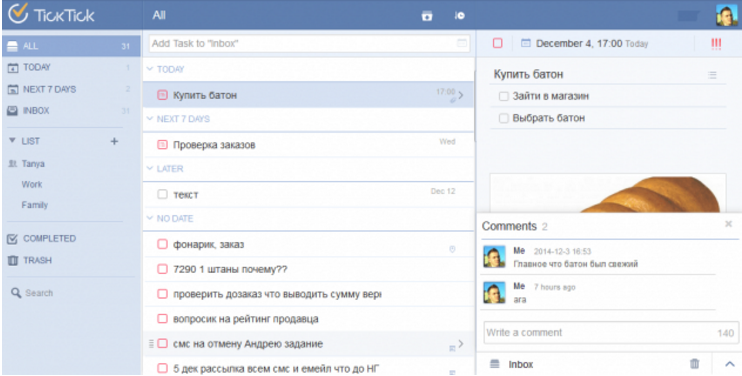


Рисунок 1.1 – Главный экран ПС «TickTick»

Также программное средство поддерживает следующий функционал:

– напоминать по времени и местоположению;

– осуществлять интеграцию с календарем;

– классифицировать задачи метками;

* различные параметры сортировки (по порядку, дате, имени, приоритету);
* добавление заметок или комментариев к задачам;
* пакетное редактирование задач;
* быстрый поиск задач.

Весомым недостатком программы является то, что она платная, а это не всегда удобно пользователю.

Программное средство *«Todoist»*

Его особенность в том, что оно позволяет планировать целые проекты, добавлять туда подзадачи и дополнительные списки. Также имеется синхронизация с социальными сетями.

Возможности программы:

– командная работа;

– возможность прикрепления файлов к задачам;

– поиск по задачам;

– мощные фильтры;

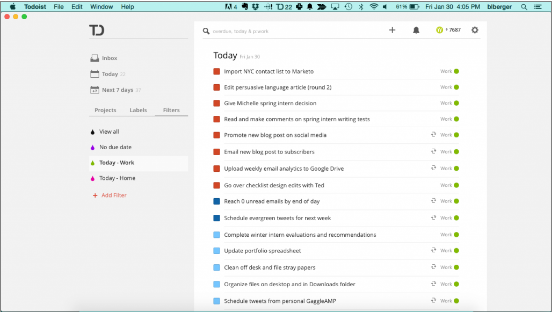


Рисунок 1.2 – Главная экран ПС «Todoist»

Данная программа платная, хоть и предоставляет бесплатный пробный период. Недостатком также является неудобные горячие клавиши и удобство использования.

**2.Разработка архитектуры проекта**

**2.1 Обобщенная структура**

Решение WpfApp1 представлено одним проектом QuickShedle, имеющем структуру, представленную на рисунке 2.1.

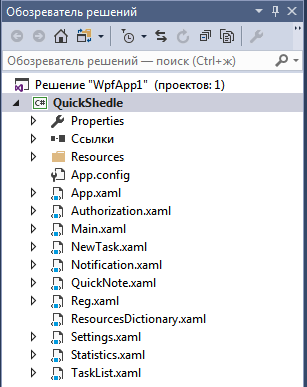


Рисунок 2.1– Структура проекта

|  |  |
| --- | --- |
| Properties | Свойства проекта, содержит информацию о сборке, используемых ресурсах и настройках |
| Ссылки | Перечень сборок, используемых в проекте |
| Resources | Папка с ресурсами проекта |
| App.config | Файл с параметрами проекта |
| App.xaml | Класс Application |
| Authorization | Окно для авторизации |
| Main | Окно «Главное меню» |
| Notification | Окно для создания новой задачи |
| QuickNote | Окно для создания быстрой заметки |
| Reg | Окно для регистрации |
| ResourcesDictionary | Словарь ресурсов |
| Settings | Окно для настроек приложения |
| Statistics | Окно для представления статистики |
| TaskList | Окно списка задач |

**2.1 Диаграммы UML, взаимосвязь всех компонентов**

ДиаграммаUML–это графическое представление набора элементов,изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации. Основная цель диаграмм – визуализация разрабатываемой системы с разных точек зрения.

Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена на рисунке 2.2 .

Классы Main, NewTask, Reg и Notification зависимы от класса App. Также класс Notification зависит от класса Main, так как в классе Main установлен таймер на создание экземпляра класса Notification.

Ассоциации установлены между классами Main и NewTask, Main Notification, Reg и Authorization, App и Authorization.

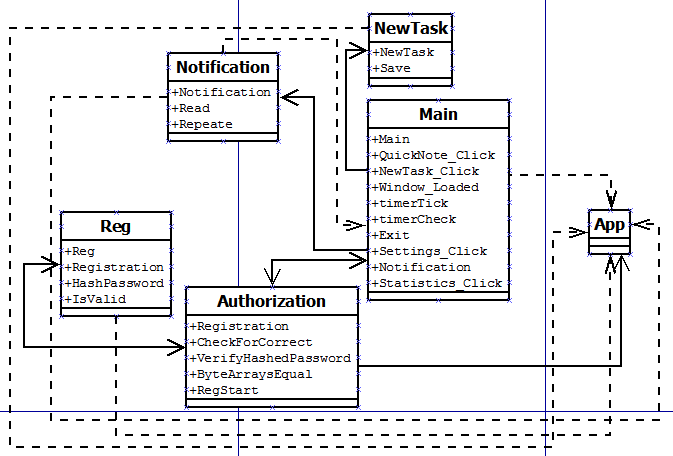


Рисунок 2.2 – Диаграмма классов

**3.Разработка функциональной модели и модели данных ПС**

**3.1 Модель базы данных**

Для реализации поставленной задачи была создана база данных QuickShedle. Для её создания использовалась система управления реляционными базами данных Microsoft SQL Server 2012.

База данных состоит из 3 таблиц, перечисленных ниже (рисунок 3.1). Скрипты для работы с базой данных представлены в Приложении Б.

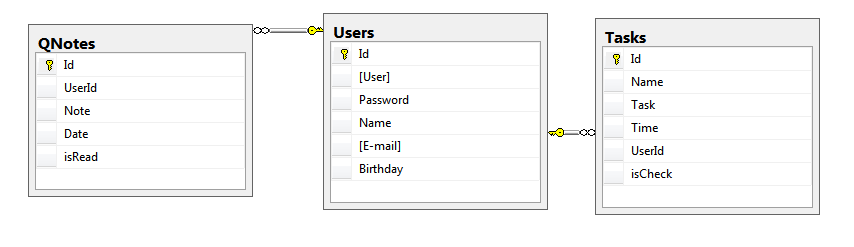


Рисунок 3.1 – Взаимосвязь таблиц базы данных

Диаграмма на рисунке 3.2 иллюстрирует структуру таблицы «Users», которая содержит информацию о пользователе.

Для полей User и E-mail установлено ограничение unique, благодаря которому невозможно добавить в таблицу повторяющееся значение имени пользователя и адреса электронной почты.

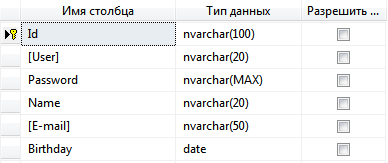


Рисунок 3.2 – Структура таблицы «Users»

Для хранения быстрых заметок создана таблица QNotes, представленная на рисунке 3.3.

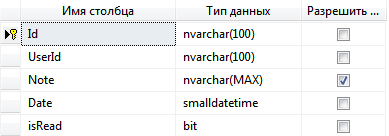


Рисунок 3.3 – Структура таблицы «QNotes»

Сведения о задачах хранятся в таблице Tasks (рисунок 3.3).

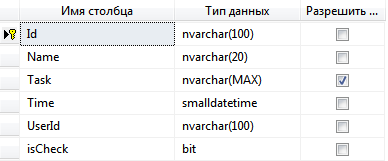


Рисунок 3.4 – Структура таблицы «Tasks»

**3.2 Выполняемые функции**

Данное приложение предоставляет минимальные требования к планировщикам задач и выполняет следующие функции:

* регистрация с последующей авторизацией пользователя;
* добавление заметок и задач;
* уведомление о запланированных задачах;
* просмотр списка задач;
* отображение статистики;
* настройка параметров работы приложения.

**3.2.1 Регистрация и авторизация**

Регистрация пользователя происходит в окне Reg, имеющем следующую структуру, отображенную на рисунке 3.5.

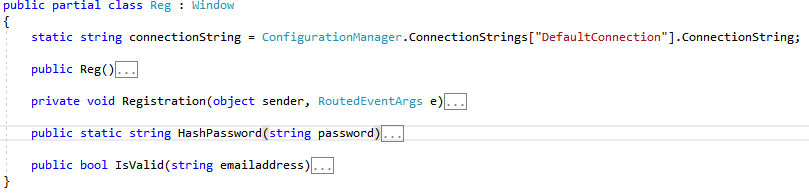


Рисунок 3.5 – Структура класса Reg

Класс Reg содержит метод Registration, который является главным в процессе регистрации.

Листинг метода Registration представлен в приложении А.

Авторизация пользователя происходит в окне Authorization, имеющем следующую структуру, отображенную на рисунке 3.6.

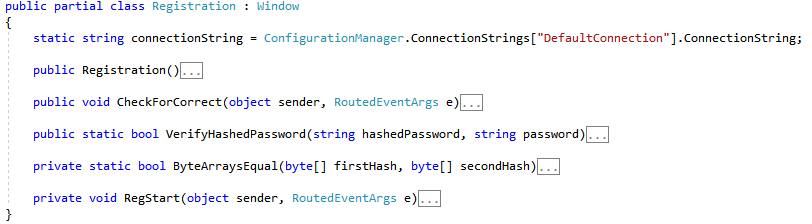


Рисунок 3.6 – Структура класса Authorization

Листинг метода CheckForCorrect представлен в приложении А.

**3.2.2 Добавление заметок и задач.**

Добавление заметок и задач происходит при открытии соответствующих окон с последующим вводом необходимой информации.

На рисунке 3.7 представлена структура класса окна QuickNote:

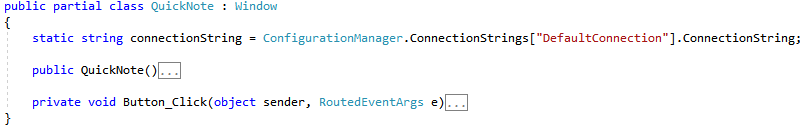


Рисунок 3.7 – Структура класса QuickNote

На рисунке 3.8 представлена структура класса окна NewTask:

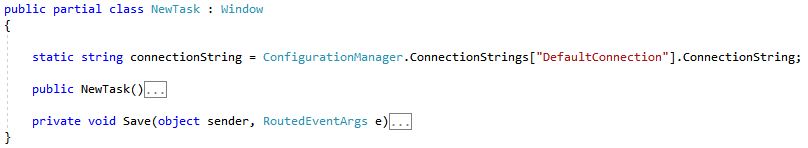


Рисунок 3.8 – Структура класса QuickNote

Класс NewTask содержит метод Save, листинг которого представлен в приложении А.

**3.2.3 Уведомление о запланированных задачах**

Уведомление пользователя о запланированных задачах реализовано с помощью класса DispatcherTimer[1], позволяющего устанавливать временные интервалы для обработки событий.

Запуск Таймера происходит при загрузке главного окна приложения Main, со следующей структурой на рисунке 3.9.

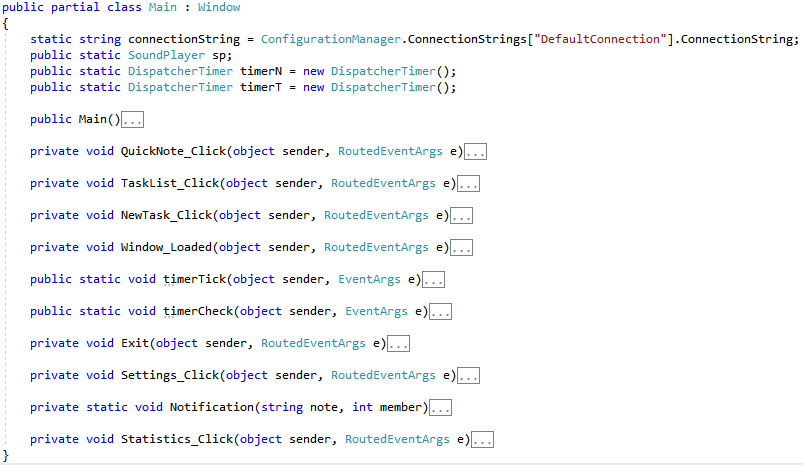


Рисунок 3.9 – Структура класса Main

Метод Window\_Loaded инициализирует счетчик, а метод timerCheck реализует уведомление. Листинг методов представлены в приложении А.

**3.2.4 Просмотр списка задач, отображение статистики и настройка приложения**

Реализация просмотра списка задач осуществлена с помощью элемента DataGrid[4] без возможности редактирования записей. Чтение из базы данных и последующий вывод их в DataGrid реализуется в классе TaskList, листинг которого представлен в приложении А.

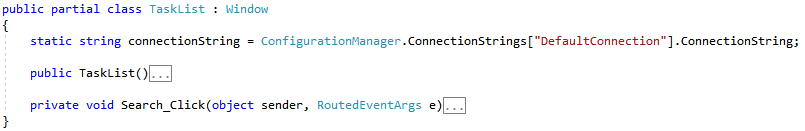


Рисунок 3.10 – Структура класса TaskList

Отображение статистики осуществляется с помощью чтения из базы данных информации о задачах, а также с помощью подключения библиотеки WpfToolkit.dll[1] реализуется построение графика по выполненным задачам на основе Chart-элемента.

Настройки работы с приложением реализуются в классе Settings и представлены на рисунке 3.11.

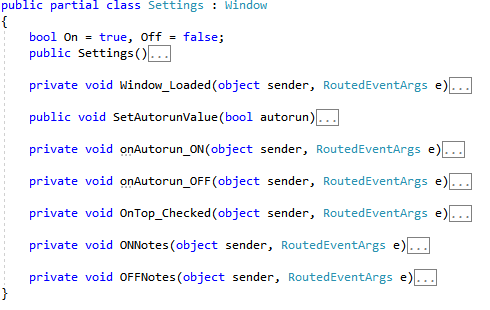


Рисунок 3.11 – Структура класса Settings

**4. Руководство пользователя**

При запуске приложения открывается окно авторизации (рисунок 4.1), на котором имеются поля для инициализации пользователя и кнопка для регистрации.

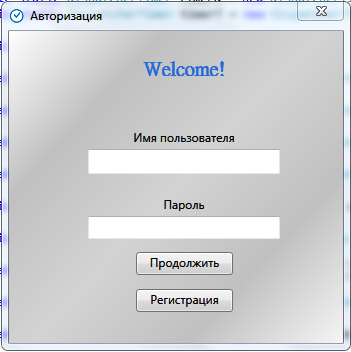


Рисунок 4.1 – Окно авторизации

После успешного прохождения авторизации пользователю будет представлено главное окно приложения (рисунок 4.2).

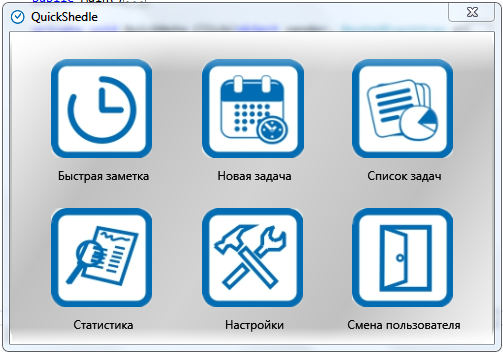


Рисунок 4.2 – Главное окно приложения

На главном окне расположено 6 кнопок для вызова соответствующих методов.

При нажатии на кнопку «Быстрая заметка» появится окно для ввода заметки (рисунок 4.3).

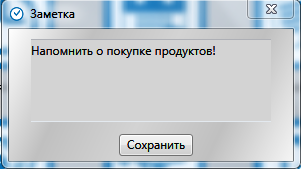


Рисунок 4.3 – Окно для добавления заметки

При нажатии на кнопку «Сохранить» пользователь будет уведомлен об успешном добавлении (рисунок 4.4).

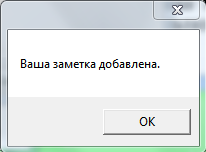


Рисунок 4.4 – Уведомление об добавлении

По истечению 1 минуты с момента добавления заметки пользователь будет оповещён звуковым сигналом и новым окном, содержащим текст его заметки (рисунок 4.5).

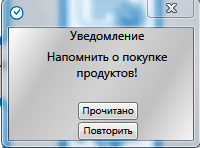


Рисунок 4.5 – Уведомление

Если пользователь захочет повторить уведомление, то необходимо нажать «Повторить», в противном случае «Прочитано».

Добавление задачи происходит аналогичным образом за исключением установления конкретного времени оповещения (рисунок 4.6).

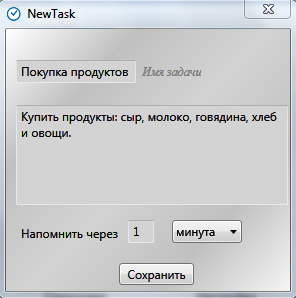


Рисунок 4.6 – Добавление задачи

По истечению указанного промежутка времени пользователь будет оповещён звуковым сигналом и новым окном с текстом задачи (рисунок 4.7).

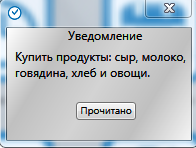


Рисунок 4.7 – Уведомление

Для отображения статистики пользователь должен нажать на одноименную кнопку на главном окне и в открывшемся окне ввести диапазон дат для отображения с последующим нажатием на кнопку «Поиск» (рисунок 4.8).

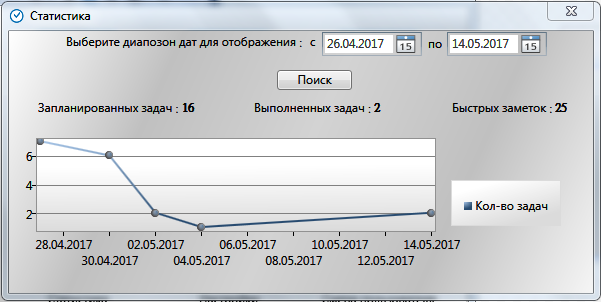


Рисунок 4.8 – Статистика

Для просмотра списка задач пользователю необходимо открыть окно «Список задач», указать диапазон дат для отображения и нажать клавишу «Поиск» (рисунок 4.9).

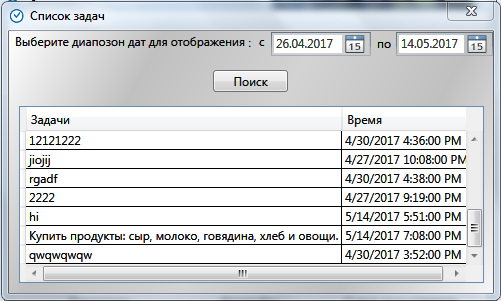


Рисунок 4.9 – Список задач

Кнопка «Настройки» вызовет одноименное окно для настроек пользователя (рисунок 4.10).

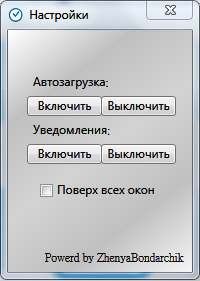


Рисунок 4.10 – Настройки

Нажатие на кнопку «Смена пользователя» вызовет окно Авторизации.

**Заключение**

В курсовом проекте описана и реализована программа, написанная на языке C#, с использованием технологий ADO.NET, WPF.

В ходе выполнения курсового проекта было создано приложение-планировщик, которое выполняет следующие функции:

* составление рабочих планов;
* сохранение рабочей информации в централизованной базе данных;
* уведомление и напоминание о запланированных задачах;
* предоставление статистики о проделанной работе;
* настройка параметров работы приложения.

Данное программное средство имеет удобный и понятный интерфейс, реагирует на ошибочный ввод данных (например, ввод данных не того типа, проверка на пустоту вводимых значений).

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

**Список используемых источников**

1. MSDN сеть разработчиков в Microsoft [Электронный ресурс] – <http://msdn.microsoft.com/library/rus/>. Дата доступа: 10.05.2017
2. Язык C# и .NET Framework [Электронный ресурс] – <http://professorweb.ru/my/csharp/charp_theory/level1/infonet.php>. Дата доступа: 10.05.2017
3. Пацей, Н.В. Курс лекций по языку программирования C# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
4. Мэтью Макдональд – WPF: Windows Presentation Foundation в .NET 4.0 с примерами на C# / М. Макдональд. – Вильямс, 2011. – 1024 с.
5. Руководство по ADO.NET и работе с базами данных [Электронный ресурс] – <https://metanit.com/sharp/adonet/>. Дата доступа: 10.05.2017

**Приложение А**

Листинг методов Регистрации:

private void Registration(object sender, RoutedEventArgs e)

{

if (Login.Text.ToString() != string.Empty ||

Password.Text.ToString() != string.Empty||

Name.Text.ToString() != string.Empty ||

Email.Text.ToString() != string.Empty ||

Birthday.Text.ToString() != string.Empty) // проверка на заполненность строк

{

Guid id = Guid.NewGuid();

string password = HashPassword(this.Password.Text); //генерация хешированного пароль

try

{

if (IsValid(this.Email.Text.ToString())) // проверка на валидность E-mail

{

string sqlExpression = string.Format("INSERT into Users (Id, [User], Password, Name, [E-mail], Birthday) values ('{0}','{1}','{2}','{3}','{4}','{5}')", id.ToString(), this.Login.Text, password, this.Name.Text.ToString(), this.Email.Text.ToString(), this.Birthday.ToString());

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();// открытие подключения к БД

SqlCommand command = new SqlCommand(sqlExpression, connection); // выполнение команды

int number = command.ExecuteNonQuery();

if (number > 0)

{

MessageBox.Show("Регистрация прошла успешно.");

Registration window = new Registration();

window.Show();

this.Close();

}

connection.Close();

}

}

else

{

MessageBox.Show("Неверный E-mail!");

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

else { MessageBox.Show("Заполните все поля!"); }

}

public static string HashPassword(string password)

{

byte[] salt;

byte[] buffer2;

if (password == null)

{

throw new ArgumentNullException("password");

}

// генерация ключа из password с размером соли 16 байт и 1000 итераций

using (Rfc2898DeriveBytes bytes = new Rfc2898DeriveBytes(password, 0x10, 0x3e8))

{

salt = bytes.Salt; //генерация соли

buffer2 = bytes.GetBytes(0x20); // возвращает значение ключа с числом генерируемых псевдослучайных байтов ключа

}

byte[] dst = new byte[0x31]; // формируем массив для хешированного пароля

Buffer.BlockCopy(salt, 0, dst, 1, 0x10); // копируем соль в хешированный пароль с 0 по 15 байт

Buffer.BlockCopy(buffer2, 0, dst, 0x11, 0x20); // копируем ключ в хешированный пароль с 16 по 49 байт

App.Hashed = Convert.ToBase64String(dst); // конвертируем хешированный пароль в строку с кодировкой Base64

return App.Hashed;

}

public bool IsValid(string emailaddress)

{

String theEmailPattern = @"^[\w!#$%&'\*+\-/=?\^\_`{|}~]+(\.[\w!#$%&'\*+\-/=?\^\_`{|}~]+)\*"

+ "@"

+ @"((([\-\w]+\.)+[a-zA-Z]{2,4})|(([0-9]{1,3}\.){3}[0-9]{1,3}))$"; // регулярное выражение

try

{

return (Regex.IsMatch(emailaddress, theEmailPattern));

}

catch (FormatException ex)

{

MessageBox.Show(ex.Message);

return false;

}

}

Листинг методов Авторизации

public void CheckForCorrect(object sender, RoutedEventArgs e)

{

bool chk = false;

string[] login = new string[1];

string[] password = new string[1];

string[] id = new string[1];

string sqlExpression = "SELECT [User], Password, Id FROM Users";

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sqlExpression, connection);

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows) // если есть данные

{

int i = 0;

while (reader.Read()) // построчное считывание

{

login[i] = reader.GetString(0);

password[i] = reader.GetString(1);

id[i] = reader.GetString(2);

i++;

Array.Resize<string>(ref login, i + 1);

Array.Resize<string>(ref password, i + 1);

Array.Resize<string>(ref id, i + 1);

}

}

reader.Close();

connection.Close();// закрытие соединения

}

if (Login.Text == string.Empty || Password.Password == string.Empty)

{

MessageBox.Show("Неверный логин или пароль!");

}

else

{

for (int c = 0; c < login.Length - 1; c++)

{

if (login[c] == Login.Text && VerifyHashedPassword(password[c], Password.Password))// проверка на совпадение пароля

{

chk = true;

App.CurrentUser = login[c];

App.CurrentId = id[c];

Main main = new Main();

main.Show();

this.Close();

}

}

if(!chk)

{

MessageBox.Show("Пользователь не найден. Зарегистрируйтесь в системе.");

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public static bool VerifyHashedPassword(string hashedPassword, string password)

{

byte[] buffer4;

if (hashedPassword == null)

{

return false;

}

if (password == null)

{

throw new ArgumentNullException("password");

}

byte[] src = Convert.FromBase64String(hashedPassword);

if ((src.Length != 0x31) || (src[0] != 0))

{

return false;

}

byte[] dst = new byte[0x10];

Buffer.BlockCopy(src, 1, dst, 0, 0x10);// копирует 0х10 сивмолов из src начиная с 1 байта в dst с 0 байта

byte[] buffer3 = new byte[0x20];

Buffer.BlockCopy(src, 0x11, buffer3, 0, 0x20);

using (Rfc2898DeriveBytes bytes = new Rfc2898DeriveBytes(password, dst, 0x3e8))

{

buffer4 = bytes.GetBytes(0x20);

}

return ByteArraysEqual(buffer3, buffer4);

}

private static bool ByteArraysEqual(byte[] firstHash, byte[] secondHash)

{

int \_minHashLength = firstHash.Length <= secondHash.Length ? firstHash.Length : secondHash.Length;

var xor = firstHash.Length ^ secondHash.Length;

for (int i = 0; i < \_minHashLength; i++)

xor |= firstHash[i] ^ secondHash[i];

return 0 == xor;

}

Листинг метода Save(…)

private void Save(object sender, RoutedEventArgs e)

{

Guid id = Guid.NewGuid();

string dateTime = string.Empty;

uint N = 0;

try

{

if (!uint.TryParse(time.Text, out N))

{

MessageBox.Show("Значение должно быть положительным и целым!", "Повторите ввод");

return;

}

if(task.Text == string.Empty)

{

MessageBox.Show("Задача должна быть описана!", "Повторите ввод");

return;

}

if (list.SelectedIndex == 0)

{

dateTime = DateTime.Now.AddHours(Convert.ToDouble(time.Text.ToString())).ToString("d-MM-yyyy H:m:s");

}

else

{

dateTime = DateTime.Now.AddMinutes(Convert.ToDouble(time.Text.ToString())).ToString("d-MM-yyyy H:m:s");

}

string sqlExpression = string.Format("INSERT into Tasks(Id, Name, Task, [Time], [UserId], isCheck) values ('{0}','{1}','{2}','{3}','{4}','{5}')", id.ToString(), name.Text, task.Text, dateTime, App.CurrentId, 0);

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sqlExpression, connection);

int number = command.ExecuteNonQuery();

if (number > 0)

{

MessageBox.Show("Ваша задача добавлена.");

this.Close();

}

connection.Close();

}

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

Листинг методов Главного окна

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

try

{

timerN.Tick += timerTick;

timerN.Interval = new TimeSpan(0, 1, 0);

timerN.Start();

timerT.Tick += timerCheck;

timerT.Interval = new TimeSpan(0, 1, 0);

timerT.Start();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public static void timerCheck(object sender, EventArgs e)

{

string[] Task = new string[1];

string sqlExpression = "SELECT Task from Tasks Where [UserId] = '" + App.CurrentId + "' and cast(Time as time(0)) = cast(cast(getdate() as smalldatetime) as time(0)) and isCheck = '0'";

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sqlExpression, connection);

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

int i = 0;

while (reader.Read()) // построчное считывание

{

Task[i] = reader.GetString(0);

i++;

Array.Resize<string>(ref Task, i + 1);

}

}

reader.Close();

connection.Close();

if (Task[0] != string.Empty)

{

for (int i = 0; i < Task.Length - 1; i++)

{

sp.Play();

Notification(Task[i],1);

System.Threading.Thread.Sleep(1000);

}

}

}

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

Листинг метода просмотра списка задач

private void Search\_Click(object sender, RoutedEventArgs e)

{

if (pick1.ToString() == string.Empty || pick2.ToString() == string.Empty)

{

MessageBox.Show("Введите диапозон по " + DateTime.Now.ToString("d-MM-yyyy") + ".");

}

else

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlDataAdapter Tasks = new SqlDataAdapter("Select Task[Задачи], Time[Время] From Tasks where [UserId] = '" + App.CurrentId + "' and cast(Time as Date) >= '"+pick1.Text.ToString()+"' and cast(Time as Date) !> '"+pick2.Text.ToString()+"'", connection);

DataSet dbo = new DataSet("dboQuickShedle");

Tasks.FillSchema(dbo, SchemaType.Source, "Tasks");

Tasks.Fill(dbo, "Tasks");

dg.ItemsSource = dbo.Tables[0].DefaultView;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

Листинг метода для отображения статистики

private void Search\_Click(object sender, RoutedEventArgs e)

{

if (pick1.ToString() == string.Empty || pick2.ToString() == string.Empty)

{

MessageBox.Show("Введите диапозон по " + DateTime.Now.ToString("d-MM-yyyy") + ".");

}

else

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string sqlExpression = string.Empty;

int i = 0;

sqlExpression = "declare @d1 Date = '" + pick1.ToString() + "', @d2 Date = '"+pick2.ToString()+"', @i varchar(100) = '" + App.CurrentId + "'; EXEC [Statistics] @d1, @d2, @i;";

SqlCommand command1 = new SqlCommand(sqlExpression, connection);

SqlDataReader reader1 = command1.ExecuteReader();

if (reader1.HasRows) // если есть данные

{

while (reader1.Read()) // построчное считывание

{

st[i] = reader1.GetInt32(0);

i++;

Array.Resize(ref st, i + 1);

}

}

reader1.Close();

////////////////////////////////

int j = 0;

sqlExpression = "declare @d1 Date = '" + pick1.ToString() + "', @d2 Date = '" + pick2.ToString() + "', @i varchar(100) = '" + App.CurrentId + "'; EXEC Graphic @d1, @d2, @i;";

SqlCommand command2 = new SqlCommand(sqlExpression, connection);

SqlDataReader reader2 = command2.ExecuteReader();

if (reader2.HasRows) // если есть данные

{

while (reader2.Read()) // построчное считывание

{

values[j] = reader2.GetInt32(0);

dates[j] = reader2.GetDateTime(1);

j++;

Array.Resize(ref values, j + 1);

Array.Resize(ref dates, j + 1);

}

}

reader2.Close();

connection.Close();

z1.Text = st[0].ToString();

z2.Text = st[1].ToString();

z3.Text = st[2].ToString();

Dictionary<DateTime, int> x = new Dictionary<DateTime, int>();

for(int k=0;k<values.Length-1;k++)

{

x.Add(dates[k], values[k]);

}

((LineSeries)mcChart.Series[0]).ItemsSource = x;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

**Приложение Б**

use dboQuickShedle;

SELECT \* from Users;

SELECT \* from QNotes;

SELECT \* from Tasks;

Create table QNotes

(

Id nvarchar(100) primary key,

[User] nvarchar(20),

Note nvarchar(MAX) NULL,

Date smalldatetime,

isRead bit

)

-------------------------------

Create table Users

(

Id nvarchar(100) primary key,

[User] nvarchar(20) unique,

Password nvarchar(MAX),

Name nvarchar(20),

[E-mail] nvarchar(50),

Birthday date

)

-------------------------------------------------------------------

select \* from Tasks order by Time;

use dboQuickShedle

go

CREATE PROCEDURE ExecTasks

@date Date,

@id varchar(100)

as begin

select count(\*) from Tasks where cast(Time as Date) = @date and isCheck = 1 and UserId = @id;

end;

-------------------------------------------------------------------

use dboQuickShedle

go

CREATE PROCEDURE [Statistics]

@date1 Date,

@date2 Date,

@id varchar(100)

as begin

Select count(\*) From Tasks where [UserId] = @id and cast(Time as Date) >= @date1 and cast(Time as Date) !> @date2 and isCheck = 1

union all

Select count(\*) From Tasks where [UserId] = @id and cast(Time as Date) >= @date1 and cast(Time as Date) !> @date2 and isCheck = 0

union all

Select count(\*) From Qnotes where [UserId] = @id and cast([Date] as Date) >= @date1 and cast([Date] as Date) !> @date2

end;

---------------------------------------------------------------

use dboQuickShedle

go

CREATE PROCEDURE Graphic @date1 Date, @date2 Date, @id varchar(100)

as begin

SELECT count(\*), cast(Time as Date) FROM Tasks where [UserId] = @id and cast(Time as Date) >= @date1 and cast(Time as Date) !> @date2 group by cast(Time as Date)

end;