МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра современных технологий программирования

ПАПРОЦКАЯ АНАСТАСИЯ СЕРГЕЕВНА

**Разработка приложения для планирования дел**

Курсовая работа

по дисциплине «Алгоритмизация и программирование»

студентки 2 курса специальности

1– 26 03 01 «[Управление информационными ресурсами](http://mf.grsu.by/abit/docs/uprav)»

дневной формы получения образования

|  |  |
| --- | --- |
|  | Научный руководитель  Гуща Юлия Вальдемаровна,  старший преподаватель  кафедры современных технологий программирования |

Гродно 2019

**РЕЗЮМЕ**

**Курсовая работа**: 20 с., 8 рис., 2 прил., 5 источников.

ПРИЛОЖЕНИЕ, ПЛАНИРОВАНИЕ, ЗАДАЧИ, WPF, C#.

**Цель работы:** разработать просто и удобное в использовании приложение для планирования дел.

**Методы исследования:** объектно-ориентированный анализ и проектирование, сравнительный, моделирования, обобщения.

**Элементы научной новизны:** разработано приложения для планирования дел, которое позволит использовать его в практических целях.

**Область возможного практического применения:** применение в личных целях.

Автор работы подтверждает, что приведённый в курсовой работе теоретический материал правильно и объективно отражает все необходимые для разработки приложения аспекты, а все заимствованные из литературных и других источников теоретические, методологические и методические положения и концепции сопровождаются ссылками на их авторов.

# **СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc532028438)

[1. Анализ предметной области 5](#_Toc532028439)

[1.1 Основные аспекты 5](#_Toc532028440)

[1.2 Обзор существующих решений 5](#_Toc532028440)

[1.2.1 Приложение «Microsoft To-Do: List, Task & Reminder» 6](#_Toc532028440)

[1.2.2 Сервис управления делами «ЛидерТаск» 7](#_Toc532028440)

[2. проектирование системы 9](#_Toc532028444)

[2.1 Выбор инструмента для создания интерфейса 9](#_Toc532028445)

[2.2 Требования к разоаботке 9](#_Toc532028446)

[2.3 Проектирование пользовательского интерфейса 10](#_Toc532028446)

[3. реализация системы 12](#_Toc532028439)

[3.1 Программная реализация приложения 12](#_Toc532028440)

[3.2 Графическая реализация приложения 15](#_Toc532028440)

[ЗАКЛЮЧЕНИЕ 19](#_Toc532028450)

[Список использованных источников 20](#_Toc532028439)

ПРИЛОЖЕНИЕ 1 21

ПРИЛОЖЕНИЕ 2 22

# **ВВЕДЕНИЕ**

В современном мире темп жизни настолько увеличился, что стало намного сложнее удержать всю нужную информацию в голове. Из-за большого разнообразия различных программных продуктов связанных с планированием дел, доступных сегодня в продаже и открытом доступе, пользователь может столкнуться с проблемой выбора наилучшего для него варианта. Многие приложения могут быть сложны в использовании или вовсе не соответствовать необходимым потребностям. В связи с этим вопрос создания наиболее удобного приложения, с помощью которого можно эффективно решать поставленные задачи, стоит на первом месте как у разработчиков программного обеспечения, так и у пользователей, которые имеют необходимость в данном программном продукте.

Цель курсовой работы ­­− разработать простое приложение для ведения всех возможных списков дел и задач, в котором можно будет записывать и планировать всё, начиная от списка покупок и заканчивая любым объёмным проектом. При этом у пользователя будет возможность контролировать свою продуктивность в выполнении любой задачи с помощью различных списков задач – текущие задачи, завершённые и просроченные.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Выполнить обзор и анализ похожих продуктов категории «Планировщик дел», выделить сущности предметной области, сформулировать требования к функциям и техническим характеристикам разрабатываемого приложения.

2. Спроектировать архитектуру приложения «Melman's Assistant», выделить основные программные модули.

3. Реализовать приложение «Melman's Assistant», предоставляющее пользователю следующие возможности:

* организовывать всю важную информацию в заметки и списки задач;
* планировать всевозможные мероприятия, проекты, персональные задачи и т.д.;
* использовать разные списки (ближайшие, просроченные и т.д.) для анализа своей деятельности и повышения продуктивности в дальнейшем достижении поставленных задачах.

# **1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## 1.1 Основные аспекты

Планирование − оптимальное распределение ресурсов для достижения поставленных целей, деятельность (совокупность процессов), связанная с постановкой целей (задач) и действий в будущем. [1]

Необходимость в планировании задач на день является актуальной для каждого человека, так как именно с помощью планирования можно всегда заботится о том, что должно произойти. Смысл заключается в том, чтобы предугадывать события и действовать, потому что именно это поможет избежать ненужных ошибок и использовать все возможности по максимуму.

Планирование – ежедневный ритуал, который полезно использовать для улучшения своей личностной эффективности. Самое лучшее, что каждый может сделать для себя – это работа над автоматизацией такой важной привычки как планирование.

В наше время планирование в блокнотах и на листиках ушло на второй план, так как только часть людей считает такой способ действенным и полезным. Другая часть активно использует различные программные продукты, созданные как для смартфонов, так и для компьютеров/ноутбуков. И этот вариант считается наиболее эффективным, потому что данные девайсы всегда находятся под рукой, они надёжны, удобны и поддерживают синхронизацию данных.

Основными целями планировщика задач являются:

* создание списка задач,
* установка даты, до которой необходимо выполнить задачу,
* отслеживание процесса выполнения задач,
* напоминание о существующих задачах.

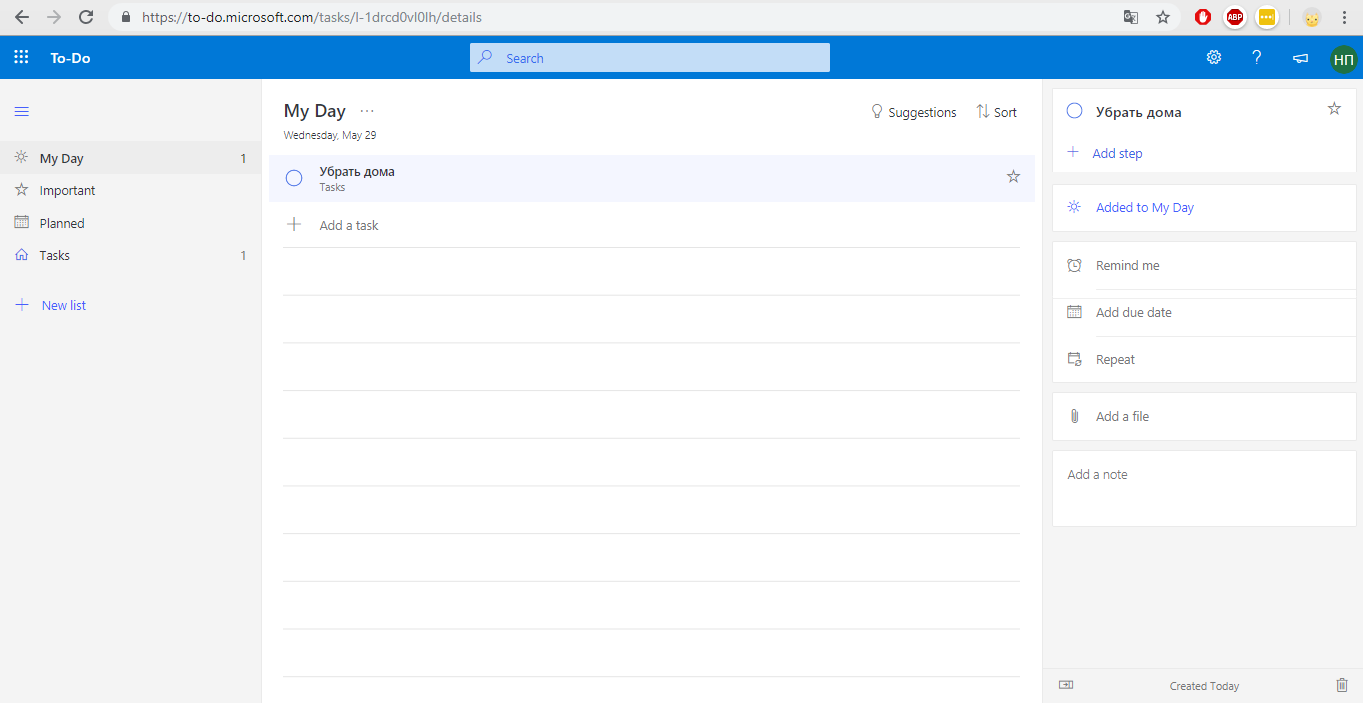
## 1.2 Обзор существующих решений

Функционал большинства планировщиков достаточно схож:

* добавить/удалить/изменить задачу,
* задать дату, до которой необходимо выполнить данную задачу,
* добавить к задаче описание, тег, тип, приоритет,
* установить напоминание о необходимости выполнить задачу.

На рассмотрение и краткий обзор подобных существующих решений были выбраны такие планировщики как «Microsoft To-Do: List, Task & Reminder» и сервис управления делами «ЛидерТаск».

**1.2.1 Приложение «Microsoft To-Do: List, Task & Reminder»**



**Рисунок 1** – Интерфейс приложения Microsoft To-Do

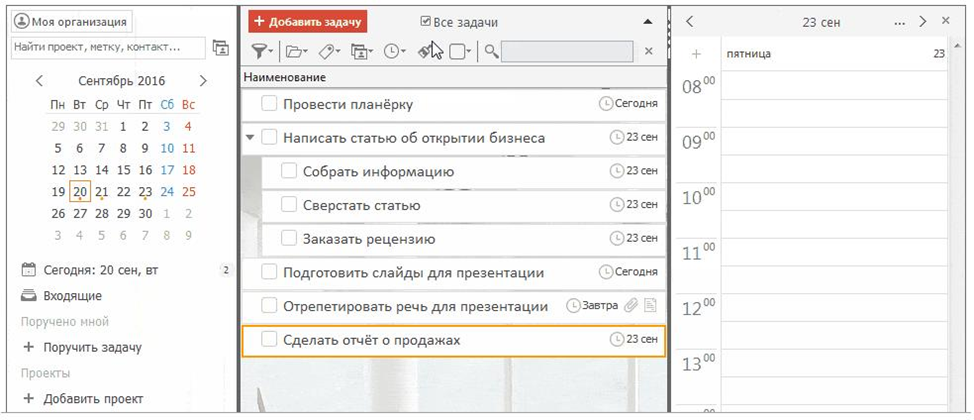
Примечание – Источник: собственная разработка

Простое и удобное приложение для ведения списка дел Microsoft To-Do поможет вам правильно планировать день и успевать все рабочие, учебные и домашние дела без лишних переживаний и хлопот. Приложение отличается красивым дизайном и обладает набором интеллектуальных функций. [2]

Данное приложение имеет мобильную и веб-версию, доступно на iOS, Windows и Android. С его помощью вы можете:

* добавить задачу,
* отметить её как выполненную или отменить это действие,
* пользоваться списками задач, такими как “мой день”, “важное”, “запланированное”,
* возможность редактирования задачи (напоминание, прикрепить файл, до какого числа назначить выполнение) и т.д.

Помимо этого, он также предоставляет достаточное количество дополнительных возможностей. Функционал данного решения достаточно гибкий и позволяет максимально точно описать суть и цель задачи. Но действительно небольшое количество людей используют эти возможности данного приложения: кто-то не знает (используют более популярные приложения), для кого-то данный способ просто не удобен.

**1.2.2 Сервис управления делами «ЛидерТаск»**

**Рисунок 2** – Интерфейс приложения ЛидерТаск

Примечание – Источник: собственная разработка

LeaderTask (ЛидерТаск) – сервис для управления делами и поручениями. Представляет собой программы для Windows, Mac, Android, iPhone, iPad. Позволяет составлять списки дел на день/неделю/месяц, раздавать поручения и контролировать исполнение, а также управлять проектами. [3]

Данное решение предоставляет мощный функционал для работы с задачами.

В его дополнительные возможности входят:

* прикрепление файлов к задачам,
* создание напоминаний,
* разделение задач и раздача распоряжений другим пользователям и т.д.

Данный сервис является действительно очень функциональным, но для использования дополнительных функций необходимо каждый месяц оформлять платную подписку (около 10$/мес.).

Данный факт делает сервис не таким доступным и менее привлекательным.

# **2. ПРОЕКТИРОНИВАЕ СИСТЕМЫ**

## 2.1 Выбор инструмента для создания интерфейса

До начала разработки приложения нужно было сделать выбор между Windows Forms или Window Presentation Foundation(WPF) для создания графического интерфейса. Выбор был сделан в сторону WPF, так как эта технология новее, чем Windows Forms и для рендеринга графики используется графический процессор на видеокарте компьютера. Это позволяет использовать аппаратное ускорение графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: он даёт возможность создавать графический интерфейс, используя или декларативное объявление интерфейса, или код на языке C#, либо совмещать и то, и другое.

Также у WPF имеется широкий спектр возможностей по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое. [4]

Ещё одним ключевым фактором являлось то, что в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.

В качестве среды разработки приложения будет использована Visual Studio 2017 Community. Используемый язык программирования – С#.

## 2.2 Требования к разработке

В данной курсовой работе описывается приложение для планирования дел «Melman's Assistant», которое реализовано в соответствии со следующими требованиями, которые являются основой постановки задачи:

1. Программа позволяет просматривать ближайшие задачи при загрузке приложения (если таковые имеются).
2. Программа должна давать возможность добавить/редактировать/ завершить(отметить как выполненную) задачу.
3. Программа должна позволять просматривать все задачи.
4. Программа должна предоставлять возможность очистить историю завершённых задач.
5. Программа должна позволять отмечать просроченные задачи как “выполнена/не выполнена”.
6. Програма должна предоставлять возможность сортировать задачи по определенным критериям в колонках.
7. Программа должна предоставлять возможность просмотра рейтинга в зависимости от количества выполненных и не выполненных задач.
8. Запись и чтение данных с помощью xml файлов.

Разработанное в процессе исследования приложение ориентированно на широкий круг пользователей и предоставляет базовый функционал простого и удобного планировщика дел.

## 2.2 Проектирование пользовательского интерфейса

Графический интерфейс будет представлять собой несколько окон, содержащих в себе информацию, введенную пользователем, сгенерированную автоматически и элементы, расположенные по умолчанию при разметке этих окон.

Для отображения списка задач будет использоваться элемент DataGrid, с привязкой заголовков столбцов к полям объектов классов. Для навигации по окнам проекта будут использоваться элементы Button.

Данный проект подразумевает создание задач, хранение различной информации о задачах и обработку этих самых задач. Для реализации этого создано три класса: Task, TaskManager и Serialization.

В классе Task должны быть следующие поля:

* Номер задачи,
* Тег задачи,
* Тип задачи,
* Описание задачи,
* Последний день выполнения,
* Статус выполнения задачи,
* Приоритет задачи,
* Дата успешного выполнения.

В классе TaskManager будут храниться различные объекты класса Task. Поэтому необходимо реализовать следующие поля:

* Списки задач (текущие, ближайшие, завершённые, просроченные и временные, которые будут использоваться для поиска задач по дате).

И методы:

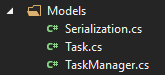
* Добавление задачи,
* Завершение задачи,
* Поиск задач,
* Проверка и сортировка задач по спискам,
* Загрузка/сохранение ресурсов (списков задач),
* Проверка наличия тегов и типов.

Для создания списков задач будет использоваться коллекция ObservableCollection, поскольку она позволяет извещать элементы, использующие ее в качестве источника данных, о том, что коллекция была изменена - чего не позволяет делать коллекция типа List. Данный подход избавит от необходимости вручную обновлять элементы интерфейса, отображающие данные о задачах.

# **3. РЕАЛИЗАЦИЯ СИСТЕМЫ**

## 3.1 Программная реализация приложения

В основе проекта лежат три класса: Task, TaskManager, Serialization.



**Рисунок 3** – Структура моделей

Примечание – Источник: собственная разработка

Класс Task содержит в себе все спроектированные поля, которые в дальнейшем заполняются пользователем.

[Serializable]

public class Task

{

public int ID { get; set; }

public string Type { get; set; }

public string Tag { get; set; }

public string Priority { get; set; }

public string Description { get; set; }

public string StringDate { get; set; }

public bool Status { get; set; }

public DateTime Date { get; set; }

public string FinishDate { get; set; }

public Task()

{

Status = false;

}

public Task(string type, string tag, string priority, string description, DateTime date)

{

Type = type;

Tag = tag;

Priority = priority;

Description = description;

StringDate = date.ToLongDateString();

Date = date.Date;

Status = false;

}

public void ChangeStatus()

{

if (!Status)

{

Status = true;

}

}

}

**Листинг 3.1** – Класс Task

Примечание – Источник: собственная разработка

В классе TaskManager хранятся списки тегов, типов, задач, и логика работы методов, для обработки этих списков. А также методы для загрузки и сохранения ресурсов (информации, хранящейся в списках). В листинге в основном перечислены методы данного класса.

[Serializable]

[XmlInclude(typeof(Task))]

public class TaskManager

{

private static TaskManager instance;

public static TaskManager GetInstance()

{

if (instance == null)

{

instance = new TaskManager();

}

return instance;

}

public TaskManager(){ }

public ObservableCollection<Task> tasks = new ObservableCollection<Task>();

public ObservableCollection<Task> completedTasks = new ObservableCollection<Task>();

public ObservableCollection<Task> approachingTasks = new ObservableCollection<Task>();

public ObservableCollection<Task> tempTasks = new ObservableCollection<Task>();

public ObservableCollection<Task> overdueTasks = new ObservableCollection<Task>();

public ObservableCollection<string> tags = new ObservableCollection<string>();

public ObservableCollection<string> types = new ObservableCollection<string>();

public void AddTask(Task newTask)

public void CompleteTask(Task task, bool status)

public void FindTasks(SelectedDatesCollection dates)

public bool IsTagExist(string tag)

public bool IsTypeExist(string type)

public void CheckTasks()

public void LoadResources()

public void SaveResources()

**Листинг 3.2** – Класс TaskManager со всеми методами

Примечание – Источник: собственная разработка

Для гарантии, что будет создан только один объект класса TaskManager а также предоставления к этому объекту точки доступа, использовался порождающий паттерн Singleton.

Класс Serialization предоставляет доступ к механизму сериализации полей объекта класса TaskManager, для того, чтобы состояние созданного экземпляра сохранялось в том виде, в котором он находился, и в последующем могло быть изменено.

public class Serialization <T>

{

public static void Serialize(T tempVar, string filename)

{

XmlSerializer serializer = new XmlSerializer(typeof(T));

using (FileStream fs = new FileStream(filename, FileMode.Create))

{

serializer.Serialize(fs, tempVar);

}

}

public static T Deserialize(string filename)

{

T tempVar;

using (FileStream fs = new FileStream(filename, FileMode.Open))

{

XmlSerializer deserializer = new XmlSerializer(typeof(T));

tempVar = (T)deserializer.Deserialize(fs);

return tempVar;

}

}

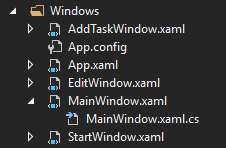
}

**Листинг 3.3** – Класс Serialization

Примечание – Источник: собственная разработка

## 3.2 Графическая реализация приложения

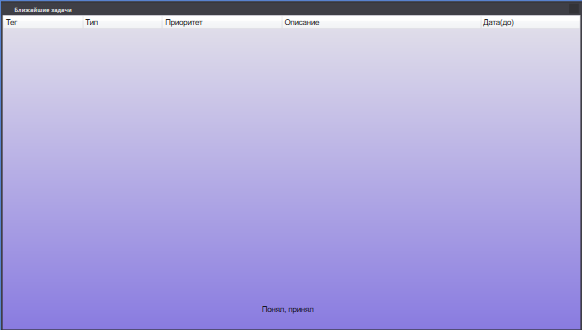
За графическую часть отвечают окна StartWindow, MainWindow, AddTaskWindow и EditWindow.



**Рисунок 4** – Структура окон

Примечание – Источник: собственная разработка

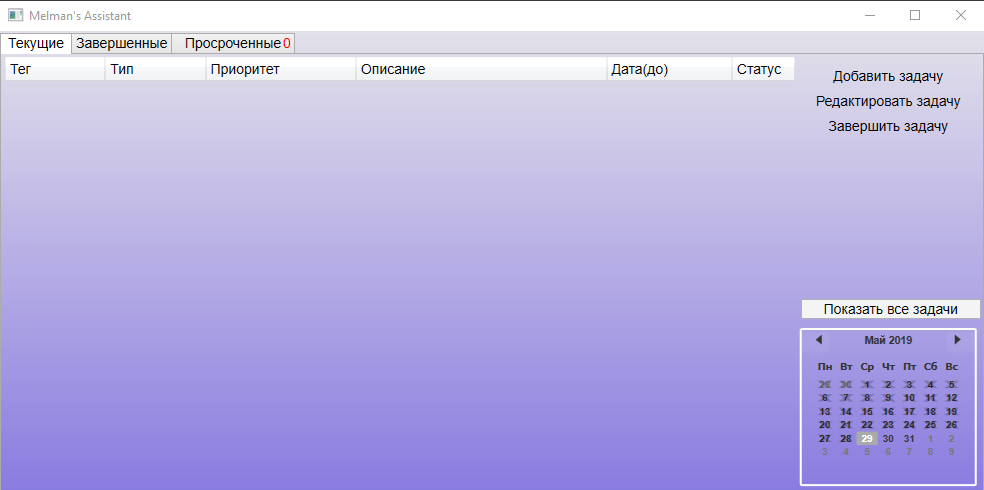
Окно StartWindow является точкой входа в приложение. В нем осуществляется загрузка всех ресурсов для экземпляра класса TaskManager, проверка и сортировка задач, нахождение задач, ближайших к исполнению задач, если таковых нет, то выдаётся соответственное сообщение и запускается окно MainWindow.



**Рисунок 5** – Вариация окна StarWindow

Примечание – Источник: собственная разработка

В окне MainWindow содержится вся основная информация о существующих задачах пользователя, возможность создания, изменения, удаления задачи, просмотр выполненных или просроченных задач. Также поддерживается сортировка задач по их данным и поиск задач по необходимой дате выполнения с помощью выделения ее календаре.

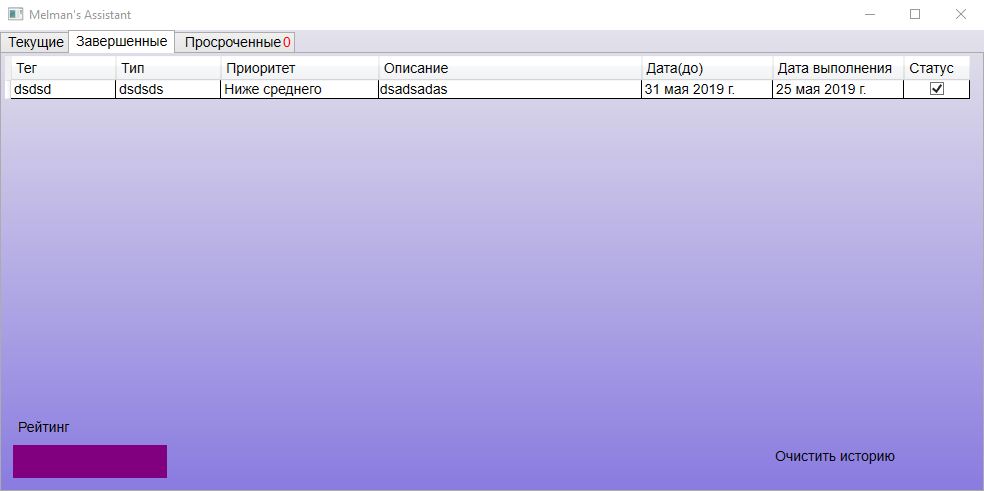


**Рисунок 6** – Окно MainWindow, вкладка «Текущие»

Примечание – Источник: собственная разработка

Данное окно содержит в себе элемент TabItem, который позволяет переключаться между тремя вкладками «Текущие», «Завершённые» и «Просроченные» задачи. В каждой вкладке находится элемент DataGrid, который хранит список тех или иных задач.

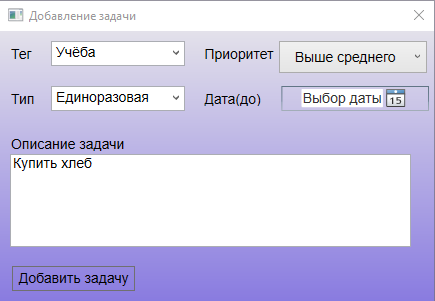
На вкладке «Завершённые» присутствует элемент ProgressBar, который отображает рейтинг пользователя. Данный рейтинг рассчитывается на основе соотношения завершённых и успешно выполненных задач. Результат отображается через изменение цвета (если процент выполненных задач <50% – красный, <70% – зелёный, >90% – фиолетовый). Также есть возможность очистить список всех завершённых задач.



**Рисунок 7** – Окно MainWindow, вкладка «Завершённые»

Примечание – Источник: собственная разработка

Окно AddTaskWindow осуществляет механизм создания и добавления новой задачи, с проверкой корректности введённых данных. В поля «Тег» и «Тип» пользователь имеет возможность вводить информацию вручную, тем самым пополнять соответствующие коллекции данных. Окно EditWindow предоставляет доступ для изменения основной информации о задаче. Реализует те же функции, что и окно AddTaskWindow



**Рисунок 8** – Окно EditWindow

Примечание – Источник: собственная разработка

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсовой работы были изучены новые возможности языка высокого уровня C#, а также были изучены его свойства. Был разработан программный код для реализации приложения на платформе Visual Studio 2017 с использованием технологии WPF (Windows Presentation Foundation) – обширный API-интерфейс для создания настольных графических программ имеющих насыщенный дизайн и интерактивность. [5]

Также при разработке был использован язык XML для сериализации и десериализации файлов. Написание программы содействовало закреплению абстрактного использованного материала на практике. Данная курсовая работа, помогла прояснить некоторые определенные теоретические вопросы, а также практические проблемы, которые были сопряжены с визуальной средой программирования С#.

В рамках данной курсовой работы было реализовано приложение для планирования дел, которое будет понятно и удобно для любого пользователя.

В ходе написание курсовой работы были получены практические навыки разработки программных приложений и глубокие теоретические.

В заключении необходимо отметить, что поставленные задачи и цель курсовой работы были успешно достигнуты.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Википедия [Электронный ресурс] / Планирование – Режим доступа: [https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5](https://ru.wikipedia.org/wiki/Планирование). – Дата доступа: 29.05.2019.
2. Приложение в Google Play – Microsoft To-do: список дел, задачи и напоминания [Электронный ресурс] – Режим доступа: <https://play.google.com/store/apps/details?id=com.microsoft.todos&hl=ru> – Дата доступа: 29.05.2019.
3. Википедия [Электронный ресурс] / LeaderTask – Режим доступа: <https://ru.wikipedia.org/wiki/LeaderTask> – Дата доступа: 29.05.2019.
4. WPF | Введение [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/wpf/1.php> – Дата доступа: 29.05.2019.
5. WPF – Windows Presentation Foundation [Электронный ресурс] – Режим доступа: <https://professorweb.ru/my/WPF/base_WPF/level1/info_WPF.php> – Дата доступа: 29.05.2019.

# **ПРИЛОЖЕНИЕ 1**

**Листинг 3.1 Класс Task**

[Serializable]

public class Task

{

public int ID { get; set; }

public string Type { get; set; }

public string Tag { get; set; }

public string Priority { get; set; }

public string Description { get; set; }

public string StringDate { get; set; }

public bool Status { get; set; }

public DateTime Date { get; set; }

public string FinishDate { get; set; }

public Task()

{

Status = false;

}

public Task(string type, string tag, string priority, string description, DateTime date)

{

Type = type;

Tag = tag;

Priority = priority;

Description = description;

StringDate = date.ToLongDateString();

Date = date.Date;

Status = false;

}

public void ChangeStatus()

{

if (!Status)

{

Status = true;

}

}

}

# **ПРИЛОЖЕНИЕ 1**

**Листинг 3.2 Класс TaskManager**

[Serializable]

[XmlInclude(typeof(Task))]

public class TaskManager

{

private static TaskManager instance;

public static TaskManager GetInstance()

{

if (instance == null)

{

instance = new TaskManager();

}

return instance;

}

public TaskManager(){ }

public ObservableCollection<Task> tasks = new ObservableCollection<Task>();

public ObservableCollection<Task> completedTasks = new ObservableCollection<Task>();

public ObservableCollection<Task> approachingTasks = new ObservableCollection<Task>();

public ObservableCollection<Task> tempTasks = new ObservableCollection<Task>();

public ObservableCollection<Task> overdueTasks = new ObservableCollection<Task>();

public ObservableCollection<string> tags = new ObservableCollection<string>();

public ObservableCollection<string> types = new ObservableCollection<string>();

public void AddTask(Task newTask)

{

newTask.ID = tasks.Count + 1;

tasks.Add(newTask);

}

public void CompleteTask(Task task, bool status)

{

if (status)

{

task.FinishDate = DateTime.Today.Date.ToLongDateString();

task.ChangeStatus();

completedTasks.Add(task);

tasks.Remove(task);

approachingTasks.Remove(task);

}

else

{

task.FinishDate = DateTime.Today.Date.ToLongDateString();

completedTasks.Add(task);

tasks.Remove(task);

approachingTasks.Remove(task);

}

}

public void FindTasks(SelectedDatesCollection dates)

{

if (tempTasks!=null || tempTasks.Count!=0)

{

tempTasks.Clear();

}

foreach (DateTime date in dates)

{

foreach (Task task in tasks)

{

if (task.Date == date)

{

tempTasks.Add(task);

}

}

}

}

public bool IsTagExist(string tag)

{

if (tags.Contains(tag))

return true;

else

{

tags.Add(tag);

return false;

}

}

public bool IsTypeExist(string type)

{

if (types.Contains(type))

return true;

else

{

types.Add(type);

return false;

}

}

public void CheckTasks()

{

if (tasks.Count.Equals(0))

{

approachingTasks.Clear();

}

for (int i = 0; i < tasks.Count; i++)

{

if (tasks[i].Date.Date <= DateTime.Today.Date.AddDays(4) && tasks[i].Date.Date >= DateTime.Today.Date)

{

if (!approachingTasks.Contains(tasks[i]))

{

approachingTasks.Add(tasks[i]);

continue;

}

}

if (tasks[i].Date.Date < DateTime.Today && !overdueTasks.Contains(tasks[i]))

{

overdueTasks.Add(tasks[i]);

tasks.Remove(tasks[i]);

--i;

}

}

}

public void LoadResources()

{

tasks = Serialization<ObservableCollection<Task>>.Deserialize("tasks.xml");

completedTasks = Serialization<ObservableCollection<Task>>.Deserialize("completedTasks.xml");

overdueTasks = Serialization<ObservableCollection<Task>>.Deserialize("overdueTasks.xml");

tags = Serialization<ObservableCollection<string>>.Deserialize("tags.xml");

types = Serialization<ObservableCollection<string>>.Deserialize("types.xml");

}

public void SaveResources()

{ Serialization<ObservableCollection<Task>>.Serialize(tasks, "tasks.xml"); Serialization<ObservableCollection<Task>>.Serialize(completedTasks, "completedTasks.xml"); Serialization<ObservableCollection<Task>>.Serialize(overdueTasks, "overdueTasks.xml");

Serialization<ObservableCollection<string>>.Serialize(tags,"tags.xml");

Serialization<ObservableCollection<string>>.Serialize(types, "types.xml");

} }