МИНОБРНАУКИ РФ

ФГБОУ ВО Тверской государственный технический университет

Кафедра “Программное обеспечение”

Курсовая работа

дисциплина «Объектно-ориентированное программирование»

Тема: Разработка приложения управления списком дел «Timelon» с использованием технологий C# на основе WPF

Выполнили: студенты группы

«Б.ПИН.РИС-21.06»

Латыпов Тимофей Вадимович

Симонов Александр

Громова Полина Александровна

Пономарёв Вадим

Черепан Матвей Дмитриевич

Проверил: Биллиг Владимир Арнольдович

Тверь,2021

**Оглавление**

1. **Введение 3**
2. **Проблемы, идеи и решения**
   1. **Идеи и функционал приложения**
   2. **Трудности реализации**
   3. **Пути реализации**
3. **Используемые технологии**
   1. **С# и ООП**
   2. **WPF и XAML**
4. **Структура проекта 9**
   1. **Библиотека классов**
      1. **Пространство имён TimelonCL**
      2. **Пространство имён TimelonCl.Data**
   2. **WPF интерфейс**
      1. **Структура интерфейса**
      2. **Стили XAML**
      3. **Паттерн MVVM**
   3. **Тестирование**
   4. **Поддержка пользователя**
      1. **Подсказки**
      2. **Обращение к пользователю**
5. **Рабочий функционал программы 27**
   1. **Демонстрация работы**
   2. **Скриншоты**
   3. **…**
   4. **…**
6. **Вывод**

# **1.Введение**

Сталкиваются с проблемой нехватки времени и ощущением что всё валится из рук. В такой ситуации на помощь приходят приложения управления списками дел, которые помогают структурировать свои задачи и отслеживать их выполнение.

ООП - ………. Приложение по управлению списками дел является хорошим учебным проектом для упражнения в ООП….

# **2.Проблемы, идеи и решения**

**2.1. Идеи и функционал приложения**

MindMap -> отсеивание идей

Основные функции:

Отображение нескольких списков, отслеживание выполнеиня, история выполнения

Дополнительные функции: ….

**2.2. Трудности реализации**

Список карт - > Словарь с id

Списки списков

Приоритеты

**2.3. Пути реализации**

Словарь, json, отдельные классы для списков и т.д.

# **3.Используемые технологии**

**3.1. C# и ООП**

ООП в C# - Class, struct, интерфейсы и использованные концепции

**3.2. WPF и XAML**

Краткое описание преимуществ WPF перед WinForms, описание языка разметки XAML, его возможностей

# **4.Структура проекта**

**4.1. Библиотека классов**

Библиотека проекта Timelon состоит из нескольких классов в двух пространствах имен:

* TimelonCl
* TimelonCl.Data

*4.1.1 Пространство имён TimelonCl*

Роль основного класса играет Manager (Менеджер), который отвечает за сохранение и загрузку данных, а также хранение объектов, полученных из этих данных. Стоит отметить, что Менеджер реализует шаблон проектирования Singleton (Класс-одиночка), который не позволит создать больше одного экземпляра этого класса. Это необходимо для корректной работы с файлами данных. Также эта реализация не позволит разработчикам других решений допустить неприятную ошибку при работе с классом Менеджер, случайно или же специально создав второй экземпляр класса со всеми загруженными в него данными, которые могут отличаться.

Следует иметь ввиду, что при использовании Менеджера в своих решениях, желательно инициализировать экземпляр класса как можно раньше, если он не был инициализирован до этого.

Менеджер содержит в себе сортированный список всех объектов CardList (Список Карт, подробнее о нем изложено ниже) и базовые методы работы с ним, такие как:

* GetList (Получение списка)
* SetList (Вставка Списка)
* RemoveList (Удаление Списка)
* ContainsList (Проверка на существование Списка)
* SearchByContent (Поиск по контенту во всех Списках)

Также Менеджер имеет метод InjectEssentials (Внедрить закрепленные Списки), который каждый раз при загрузке или сохранении данных проверяет наличие в сортированном списке закрепленных Списков Карт. Закрепленные Списки - это Списки по умолчанию, которые присутствуют в Менеджере

не зависимо от действий пользователя. Они нужны для быстрого доступа к функционалу Списков Карт без необходимости тратить время на создание нового. Они имеют закрепленные идентификаторы, которые не позволят затеряться в возможном множестве созданных Списков Карт.

За работу с данными отвечают всего 2 метода: Sync (Сохранить данные в файл) и Load (Загрузить данные из файла). Файл по-умолчанию находится в директории "Мои документы" в папке Timelon. Оба этих метода работают со встроенным функционалом сериализации объектов в XML. Подробнее об

этом изложено в TimelonCl.Data.

Класс Util в пространстве имен TimelonCl отвечает за реализацию самых разнообразных потребностей, которые возникают при активной разработке программного обеспечения. На текущей стадии он хранит в себе только лишь работу с генератором псевдо-случайных чисел. А конкретно:

* Статический доступ к глобальному генератору псевдо-случайных чисел (Random)
* Функция NextBool (Получение следующего случайного булевого значения)
* Функция NextString (Получение следующей случайной строки заданной длины)
* Функция NextCollectionIndex (Получение следующего случайного индекса заданной коллекции)
* Функции NextDateTime (Получение следующих даты и времени (DateTime))

*4.1.2 Пространство имён TimelonCl.Data*

Самое интересное происходит именно в пространстве имен TimelonCl.Data, хранящем в себе основные классы, экземпляры которых участвующит в обороте данных между постоянной и оперативной памятью. А именно:

* Абстрактный класс Unique (Уникальный Класс)
* Абстрактный класс DataContainer (Контейнер Данных)
* Класс DateTimeContainerData (Данные Контейнера Дат)
* Класс CardData (Данные Карты)
* Класс CardListData (Данные Списка Карт)
* Класс DateTimeContainer (Контейнер Дат)
* Класс Card (Карта)
* Класс CardList (Список Карт)

Уникальный Класс отвечает в первую очередь за автоматическую генерацию уникальныхидентификаторов в текущей сессии. Также он хранит в себе поля и доступ к ним для идентификаторови названий объектов, наследующих эту абстракцию. Генерация происходит при помощи статическогополя инкрементора, который хранит в себе текущее значение идентификатора для наследуемого типа.Функция Register (Регистрация идентификатора) регистрирует указанный идентификатор в текущей сессии для предотвращения дублирования при, например, загрузке объекта из файла. Функция UniqueId (Получение следующего идентификатора) служит для генерирования идентификатора для новых объектов-наследников. Разработчики других решений ограничены в доступе к этим функциям во избежание создания ошибок. Уникальный Класс всю работу берет на себя, достаточно только унаследовать его.

Контейнер Данных служит только для того, чтобы наследники могли быть беспрепятственно

сериализованы в XML формат. Он имеет атрибут Serializable (Сериализируемый) и хранит в себе

пустой конструктор и метод ToXmlString (Сериализация в XML).

Классы Данные Контейнера Дат, Данные Карты и Данные Списка Карт наследуют абстракцию Контейнер Данных. Они существуют обособленно от связанных с ними объектов и хранят в себе пустые публичные поля с полным доступом для хранения данных основных объектов и их сериализации стандартными методами. Таким образом, они позволяют сохранить в основных объектах необходимую инкапсуляцию.

Контейнер Дат, Карта и Список Карт содержат в себе функции FromData и методы ToData для безопасной конвертации из соответствующих контейнеров данных в экземпляр класса и обратно. Таким образом, контейнеры данных участвуют в качестве посредников при сериализации и десериализации в Менеджере в методах Sync и Load.

Контейнер Дат служит для хранения в себе нескольких экземпляров класса DateTime:

* Created (дата создания)
* Updated (дата обновления или null)
* Planned (запланированная дата или null)

Все эти даты относятся к определенному экземпляру класса Карта. Присутствует возможность получения и изменения дат, а также проверка на статус обновленности и запланирования. Стоит обратить внимание на то, что дата обновления и запланированная дата напрямую зависят от даты создания - они не могут превышать ее. Соответственно, при изменении даты создания зависимые от нее даты будут перепроверены автоматически.

Карта наследует Уникальный Класс и представляет собой карточку дела. По большей части этот класс содержит в себе основные поля, характерные для карточки.

Список Карт, также как и Карта, наследует Уникальный Класс и представляет собой список карточек дел. В качестве способа хранения экземпляров Карт в классе используется Dictionary (Библиотека) из-за отличной скорости поиска и возможности использования идентификаторов в качестве ключа. Присутствуют как базовые методы, такие как Get (Получить Карту), Set (Сохранить Карту), Remove (Удалить Карту), Contains (Проверить наличие Карты), так и методы отвечающие за сортировку и поиск. Так как Библиотека позволяет мгновенно получить Карту через ее идентификатор вне зависимости от количества хранимых Карт, достаточно произвести сортировку один раз и занести идентификаторы в отдельный список для быстрого доступа. Таких списков в классе три:

* idListDefault (идентификаторы по дате обновления)
* idListImportant (идентификаторы по статусу важности)
* idListCompleted (идентификаторы по статусу выполнения)

Этим спискам соответствуют методы:

* GetListDefault (Получение Карт, отсортированных по дате обновления)
* GetListImportant (Получение Карт, отсортированных по статусу важности)
* GetListCompleted (Получение Карт, отсортированных по статусу выполнения)

Однако, при обновлении Библиотеки (например, сохранении новой Карты), сортировку необходимо произвести заново. Для управления статусом сортировки применяется перечисление SortOrder (Направление Сортировки):

* Initial (Карты необходимо отсортировать (используется только внутри класса))
* Unsorted (В произвольном порядке)
* Ascending (По возрастанию)
* Descending (По убыванию)

Соответственно, с помощью этого перечисления при использовании методов получения отсортированных Карт существует возможность указать необходимый порядок.

**4.2. WPF и XAML**

При разработке пользовательского интерфейса нами было принято решение использовать технологию Windows Presentation Foundation вместо Windows Forms, так как она имеет несколько преимуществ:

* Гибкая настройка визуального и функционального представления элементов управления
* Более прозрачная разработка, что позволяет работать в команде и поддерживать проект на протяжении долгого времени
* Возможность изучить новые технологии и методы разработки интерфейса

В разработке нашей командой был применён паттерн MVVM, позволяющий разбить приложение на 3 функциональных модуля: Model (в нашем случае это библиотека классов); View (MainWindow.xaml); ViewModel(ApplicationViewModel. cs)

*4.2.1 Структура интерфейса*

Интерфейс приложения представляет собой несколько функциональных зон:

1. Зона списков. В ней располагается список всех списков карт, заданных по умолчанию или созданных пользователем
2. Зона карт. В ней располагается список всех карт выбранного списка, заданных пользователем.
3. Зона карты. В ней располагаются поля, содержащие информацию о выбранной карте, и кнопки, позволяющие манипулировать этой картой.

Все зоны связаны с ViewModel (далее VM) с помощью Binding, что позволяет им отображать актуальные данные в реальном времени при их изменении. Так, например, добавив в список новую карту, эта информация будет сначала передана в VM, а она, обработав этот запрос с помощью Binding вернёт актуальный список в зону карт в элемент управления ListBox.

Большая часть функционала приложения строится на классе RelayCommand, являющимся наследником интерфейса ICommand. То есть все кнопки на интерфейсе действуют благодаря описанным в VM командам, которые представляют из себя делегатов, осуществляющих соответсвующие действия с полями VM (списками списков, карт и т.д.)

*4.2.2 Стили XAML*

Неотъемлемой частью интерфейса являются стили. Они позволяют задать универсальное представление элемента управления, позволяющее во-первых, не использовать повторно один и тот же код, а во-вторых, создать специальный вид состояний различных элементов, а так же улучшить внешний вид программы, делая её более элегантной, приятной к использованию и привлекательной для пользователя.

Стили могут быть представлены как в виде отдельных файлов xaml, содержащих код для описания указанного элемента управления, так и как часть кода в MainWindow.xaml. В нашем проекте подавляющее больниство элементов управления описывается с помощью отдельного файла – темы, хранящегося в папке Themes.

*4.2.3 Паттерн MVVM.ViewModel*

В рамках паттерна MVVM, View Model (VM) – связующее звено между View (интерфейсом) и Model (библиотекой классов). В ней описана логика приложения и поведение элементов управления. Для этого используются команды и свойства.

Например, для отображения выбранного списка используется свойство SelectedList, которое обновляет выбранный списко

**4.3. Тестирование**

Консольное приложение и класс Tests.cs

Юнит-тестирование

# **5.Структура проекта**

**5.1. Демонстрация работы**

Показать как работает приложение, как создавать карточки, списки, редактировать их, задавать какието параметры

**5.2. Скриншоты**

Прикрепить скриншоты проги. Можно не только тут

# **6.Вывод**

Благодаря приложению Timelon ….

Разработка оказалась тяжелой задачей… Мы столкнулись с трудностями…

Но смогли их решить…. К сожалению, не вышло реализовать \_\_\_ ☹

Освоили принципы ООП и качественного программирования….

Узнали много нового и получили ценный опыт разработки