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**
   1. **Предисловие**
   2. **Описание и цель проекта**
2. **Идеи, проблемы и решения**
   1. **Идеи и функционал приложения**
   2. **Примеры задач, решаемых проектом**
   3. **Трудности реализации**
   4. **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**

**1.1. Предисловие**

Перед нашей командой стояла задача - необходимо выработать интересную идею для реализации как курсовой работы. При этом она должна быть не слишком сложна и не слишком проста, так как было желание научиться чему-то новому. Так же мы хотели учесть практическое применение нашего приложения, хотели сделать так, чтобы мы могли пользоваться им сами и рекомендовать другим.

Мы устроили несколько сессий мозговых штурмов, на которых составили так называемую MindMap (интеллект-карту) наших идей о том, какие проекты разрабатывать и об отдельных решениях в этих проектах. Мы не боялись задавать быстрые темпы и давать громкие обещания самим себе, дав волю фантазии. Команда вошла в азарт и смело предлагала всё более и более интересные, сложные и оригинальные решения, пока добавлять уже было нечего.

Через время после мозговых штурмов мы сели с чистой головой, чтобы обработать нашу интеллект-карту и выбрать именно то, чем мы не только хотим заняться, но и имеем достаточно навыков для этого. Отбросив большое количество вариантов от игр до гороскопов, мы остановились на списке задач.

Мы решили, что это будет отличный проект для того, чтобы и научиться чему-то новому, и в то же время оставаться заинтересованными в разработке. Этот проект имеет четкое практическое применение, его легко показать на оценку другим людям и самим оценивать результат своей работы. На начальных этапах разработки он не так сложен, чтобы отбивать желание преодолевать трудности изучения чего-то нового, но в то же время имеет большой потенциал для развития как приложения, а значит и для нас самих как разработчиков. И даже, если мы не захотим делать список задач нашим opus magnum, то он всё равно станет хорошим опытом как программирования, так и работы в команде над большим проектом.

**1.2. Описание и цель проекта**

Многие люди сталкиваются с проблемой нехватки времени и ощущением того, что всё валится из рук. В такой ситуации на помощь приходят приложения для организации жизни, например, списки задач (или таск-трекеры), которые помогают структурировать свои задачи и отслеживать их выполнение.

Список задач — программа для управления проектами. Её функционал позволяет пользователю следить за рабочими процессами и контролировать выполнение поставленных задач.

*Цель проекта*: создать удобный список задач, в котором можно создавать уникальные задачи и группировать их по спискам с возможностью отслеживать статус их выполнения.

# **2.Идеи, проблемы и решения**

**2.1. Идеи и функционал приложения**

Наш проект оперирует несколькими понятиями сущностей и обладает списком возможностей. Они описаны ниже:

*Сущности:*

* Карточка дела (карточка, карта) - набор характеристик карточки, который включает в себя краткую постановку задачи (название), описание задачи, дату последнего обновления, отметку о выполнении.

- Отметка о важности. Выделяет задачу среди всех других. Позволяет пользователю определить приоритетные для себя дела.

- Отметка о выполнении. Определяет статус задачи: В работе или выполнена

* Список карт (список, список задач) - карты, сгруппированные пользователем в отдельный список. Позволяет хранить несколько карт, сортировать их по различным параметрам, осуществлять поиск карт по части названия или описания.
* Список списков карт (менеджер списков) - список, содержащий все списки карт. Позволяет хранить списки карт, осуществлять поиск по части названия или описания по всем содержащимся в нём спискам.

*Возможности:*

* Добавление/Удаление карт, списков
* Выполнение задачи(карты). Отметить задачу как выполненную. Есть возможность просмотреть все выполненные задачи в выбранном списке.
* Отметка о важности задачи(карты). Отметить задачу как важную. Такая задача отображается в списке выше всех остальных
* Сортировка карт. Есть возможность сортировать карты в разном порядке и по разным параметрам: по дате, по степени выполнения и по важности.
* Просмотр выполненных задач. Возможность посмотреть, какие задачи были выполнены ранее в выбранном списке.
* Поиск задачи(карты) по части названия или дате.

**2.2. Примеры задач, решаемых проектом**

1. Отслеживание дел по дому (уборка, покупки, подарки близким и т.п.)
2. Отслеживание рабочих проектов (Формирование списка чётких задач, алгоритма, которому можно следовать и поэтапно выполнять поставленную задачу)
3. Постановка глобальных целей (научиться программировать, закончить университет, изучить французский язык ) и отслеживание пути к их выполнению (добавление тематических задач: записаться на языковые курсы и т.п.)

**2.3. Трудности реализации**

Во время разработки проекта мы столкнулись с несколькими трудностями, решение для которых нам предстояло найти:

* Задачи могут называться одинаково, однако должны существовать и адекватно восприниматься системой
* В программе существуют списки по умолчанию
* Организовать быстрый поиск и сортировку карт в списке
* Отображать список-родитель для найденной карты
* Отображать все важные карты в одном списке – «Важное»
* Сохранение изменений
* Необходимо уместить в элементе списка карт кнопки, отвечающие за выполнение карты, к которой прикреплены кнопки, и отметка её как важной
* Когда выбранный список пуст, пользователю должно выводится сообщение-подсказка о том, как работать с программой

**2.4. Пути реализации**

Для решения этих проблем мы ввели:

* систему ID для карт и списков
* специальное свойство для списка, которое отражает является ли он списком по умолчанию
* Класс-наследник карты, в котором хранится информация о списке-родителе
* Технологию XML, сериализацию и десериализацию данных для хранения их на компьютере в постоянной памяти

К сожалению, реализовать отображение всех важных карт в списке «Важное» и кнопки на элементе в списке карт у нас не вышло.

# **3.Используемые технологии**

**3.1. C# и ООП**

ООП (Объектно-ориентированное программирование) – методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса.

Использование ООП решает задачу структурирования информации. Такой подход существенно улучшает управляемость самим процессом моделирования.

Основными принципами ООП являются:

* Абстракция для выделения в моделируемом предмете важного для решения конкретной задачи по предмету. Представление с достаточной точностью объекта минимальным набором полей и методов
* Инкапсуляция для быстрой и безопасной организации иерархической управляемости. Предназначена для изоляции контрактных обязательств абстракции (протокол/интерфейс) от их реализации.
* Наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах. Абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.
* Полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

Язык программирования C# обладает широкими возможностями для реализации объектно-ориентированного подхода.

Уровни абстракции обеспечиваются с помощью пространств имён, интерфейсов и абстрактных классов.

Инкапсуляция возможна благодаря различным параметрам уровня доступа: private, public, protected и т.д.

**3.2. WPF и XAML**

Для разработки интерфейса приложения нами была выбрана технология Windows Presentation Foundation, основанная на языке разметки XAML. Основным преимуществом этой технологии является гибкость разработки, позволяющая не только создавать уникальный внешний вид элементов управления, но и легко поддерживать связи данных и чистоту кода.

Неотъемлемой частью XAML 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 также улучшить внешний вид программы, делая её более элегантной, приятной к использованию и привлекательной для пользователя.

# **4.Структура проекта**

**4.1. Библиотека классов**

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

* Timelon
* Timelon.Data

*4.1.1 Пространство имён Timelon*

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)

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: SaveData (Сохранить данные в файл) и LoadData (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 Timelon.Data.

*4.1.2 Пространство имён Timelon.Data*

Самое интересное происходит именно в пространстве имен TimelonCl.Data, хранящем в себе основные классы, экземпляры которых участвуют 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 уникальных идентификаторов в текущей сессии. Также он хранит в себе поля и доступ к ним для идентификаторов и названий объектов, наследующих эту абстракцию. Генерация происходит при помощи статического поля 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 ToString (Представление в виде 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 SaveData и LoadData.

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*

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 большинство 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 список

**4.3. Тестирование**

**4.3.1 Требования**

*4.3.2.Тест-план:*

Содержание

1. Введение

1.1. Основная информация

1.2. Цель

2. Область тестирования программы

3. Тест-план и стратегия тестирования

3.1. Функциональное тестирование

3.2 Нефункциональное тестирование

3.3. Процедура тестирования

4. Критерии качества

5. Результаты

1. Введение

1.1. Основная информация

Документ описывает методы и подходы к тестированию, которые будут

использоваться для тестирования приложения.

Объект тестирования — это деятельность, направленная на проверку работоспособности

функций приложения Timelon в операционных системе Windows.

1.2. Цель

Тест-план проекта преследует следующие цели:

* Определить существующую информацию о проекте и программных компонентах,
* подлежащих тестированию.
* Описать стратегии тестирования, которые будут использоваться.
* Привести результаты тестирования.

Результаты будут представлены в виде отчетов.

2. Область тестирования приложения

В объем работ по тестированию входит

* Тестирование библиотеки классов
* Тестирование реализованных в интерфейсе компонентов и функций:

1. Создание новой задачи
2. Timelon должен обладать функцией создания карточек задач.
3. Timelon должен хранить карточки задач.
4. Timelon должен обладать функцией редактирования карточек задач.
5. Timelon должен обладать функцией удаления карточек задач.
6. Timelon должен обладать функцией сортировки по алфавиту карточек задач.
7. Должна быть реализована функция приоритетности задач.
8. Timelon должен отображать название задачи.
9. Timelon должен отображать индикатор выполненности задачи.
10. Timelon должен отображать подробности задачи.
11. Timelon должен отображать время создания карточки задачи.
12. Интерфейс Timelon должен иметь читаемые шрифты.
13. Интерфейс Timelon должен обладать работающими кнопками.
14. Интерфейс Timelon должен обладать возможность исправно добавлять карточки задач.
15. На интерфейсе Timelon должна быть реализована справка для пользователя с объяснением о том, как правильно пользоваться программой.
16. На интерфейсе Timelon должна присутствовать первая карточка с сообщением для пользователя " создайте первую задачу".
17. При выходе из приложения Timelon должно отображаться предупреждение о том, что пользователь не сохранился.

3.1. Процедура тестирования

Процедура тестирования предполагает следующие пункты:

• Сообщения об обнаруженных ошибках программного обеспечения.

Необходимо проверить различные аспекты тестируемого программного обеспечения, для этого требуется выполнение различных типов тестирования.

Основные типы тестирования, которые должны быть выполнены:

• Функциональное Тестирование.

• Тестирование пользовательского интерфейса.

• Юзабилити-тестирование.

3.2. Функциональное тестирование

Цель функционального тестирования состоит в том, чтобы убедиться, что весь программный продукт работает в соответствии с требованиями, и в приложении не появляется существенных ошибок. Функциональное тестирование является наиболее существенной частью тестирования программного обеспечения, включающее в себя проверку различных аспектов системы. Программный продукт должен пройти все

запланированные тесты. Только в этом случае можно быть уверенным в его качестве.

|  |  |
| --- | --- |
| Цель тестирования: | Обеспечение надлежащего качества целевой функциональности |
| Техника: | Выполнение каждого сценария, используя допустимые и  недопустимые данные:  • Ожидаемые результаты возникают при использовании достоверных  данных.  • Соответствующие сообщения об ошибках или предупреждения  отображаются, когда используются неверные данные.  • Каждый пункт проверен. |
| Входные критерии: | Подготовлено тестовое окружение, тестируемая задача задеплоена на  тестовый стенд |
| Критерий приемки: | • Все дымовые тесты пройдены.  • Нет блокирующих багов.  • Все баги с высоким приоритетов поправлены.  • Ручные тесты пройдены.  • Автоматические тесты пройдены.  вместо таблицы ЮНИТ ТЕСТЫ? И описание блоков юнит тестов? |

3.3. Нефункциональное тестирование

Кейс-тесты!?!?!?!??!?!

4. Критерии качества

Продукт должен работать в соответствии с требованиями и техническим заданием

(если таковые имеются).

Продукт не должен содержать критических и блокирующих дефектов в

окончательной версии проекта.

5. Результаты

Результаты тестирования

Консольное приложение и класс Tests.cs

Юнит-тестирование

4.3.3 Отчёт о тестировании

# **5.Работа с пользователем**

**5.1. Внешний вид программы**

**5.2. Фоновый режим**

**5.1. Подсказки**

**5.1. Руководство пользователя**

# **6.Структура проекта**

**6.1. Демонстрация работы**

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

**6.2. Скриншоты**

Прикрепить скриншоты проги. Можно не только тут

# **7.Вывод**

Благодаря приложению Timelon пользователь может удобно и эффективно управлять своими проектами, отслеживать выполнение целей и задач.

Разработка приложения Timelon стала для нас хорошим опытом разработки относительно большого проекта в команде. Мы столкнулись с некоторыми трудностями. Например, было необходимо в достаточно короткий срок изучить новые технологии и постараться грамотно применить их в проекте. Также нам было важно смоделировать работу реальной команды разработки, поэтому мы с ответственностью подошли к созданию библиотеки классов и интерфейса, постаравшись обеспечить чистоту их кода и простоту дальнейшей поддержки приложения. Не остались без внимания и область тестирования, в которой мы постарались формализовать требования и процесс тестирования, а также область работы с пользователем, в которой мы старались привлечь и удержать пользователя в приложении, для чего было важно обеспечить максимальное удобство использования для пользователя и привлекательный внешний вид программы.

Мы справились с поставленной задачей и успешно разработали корректно работающее приложение, которое отвечает запрошенным требованиям и готово к использованию.

К сожалению, нам не удалось реализовать некоторые идеи, однако проект открыт к нововведениям и может быть дополнен нами новым функционалом в будущем.

При разработке приложения Timelon мы узнали много нового, изучили новые технологии, принципы и методологии программирования, научились создавать качественные проекты, отвечающие требованиям пользователя, а также получили ценный опыт разработки большого приложения в команде.