МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ, МОЛОДЕЖИ РЕСПУБЛИКИ КРЫМ

ОТДЕЛ ОБРАЗОВАНИЯ, МОЛОДЕЖИ И СПОРТА АДМИНИСТРАЦИИ ЧЕРНОМОРСКОГО РАЙОНА

**МУНИЦИПАЛЬНОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ «ЧЕРНОМОРСКАЯ СРЕДНЯЯ ШКОЛА №1 ИМ. НИКОЛАЯ КУДРИ» МУНИЦИПАЛЬНОГО ОБРАЗОВАНИЯ ЧЕРНОМОРСКИЙ РАЙОН РЕСПУБЛИКИ КРЫМ**

**Разработка приложения с графическим пользовательским интерфейсом для настольных компьютеров**

Работа обучающегося 11-А класса

Мельника Павла Юрьевича

Руководитель проекта

Ященко Николай Григорьевич

Работа допущена к защите «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 год

Подпись руководителя проекта \_\_\_\_\_\_ (\_\_\_\_\_\_)

Черноморское, 2023г.

Оглавление

[Введение 3](#__RefHeading___Toc270_1140880220)

[1 Теоретическая часть 5](#__RefHeading___Toc747_1140880220)

[1.1 План 5](#__RefHeading___Toc2348_1930883748)

[1.2 Определение технологий кроссплатформенной разработки приложений для ПК 6](#__RefHeading___Toc301_1306291715)

[1.3 Концепция реактивного программирования 12](#__RefHeading___Toc222_2009997258)

[1.4 Фреймворк ReactiveUI 13](#__RefHeading___Toc224_2009997258)

[1.5 Система контроля версий 14](#__RefHeading___Toc226_2009997258)

[2 Практическая часть 16](#__RefHeading___Toc2346_1930883748)

[2.1 Установка среды разработки 16](#__RefHeading___Toc228_2009997258)

Введение

Темой индивидуального проекта является разработка приложения с графическим пользовательским интерфейсом. GUI (Graphical User Interface) или ГИП (графический интерфейс пользователя) — программная оболочка, которая предоставляет пользователю удобный интерфейс для работы с операционной системой. Она визуализирует многие компоненты в виде графических объектов, например, кнопки, меню, стрелки и т. д.

К преимуществам графического интерфейса относится наличие более дружелюбной (с англ. User-Friendly) системы управления программным обеспечением по сравнению с интерфейсом командной строки.

К недостаткам GUI относят повышенное потребление системных ресурсов, особенно оперативной памяти. Это связано с тем, что все графические объекты, используемые в графическом интерфейсе загружены в оперативную память на постоянной основе.

Разрабатываемым приложением будет «список задач». Реализация данной темы важна в наше время, так как сейчас у людей часто много дел и планов, которые нужно запомнить или записать, а также организовать эту информацию. Бумажные носители можно смело назвать устаревшими, так как у них имеется множество недостатков:

* Расход конечных природных ресурсов на бумагу и принадлежности для письма
* Необходимость красиво писать от руки
* Сложность быстрого обмена информацией
* Громоздкость при наличии больших объемов данных
* Сложность поиска среди доступной информации
* Сложность сортировки, организации и анализа данных

В то же время у хранения данных в электронном виде множество преимуществ:

* Информацию можно легко и быстро передавать в любую точку мира через интернет
* Можно осуществлять поиск нужной информации с помощью компьютерных программ, удобно структурировать данные и выполнять их анализ
* Хранение большого количества информации не занимает много места в реальном мире, не требует расхода природных ресурсов.
* ременную карту памяти размером с ноготь можно поместить содержимое всех библиотек мира
* Данные в электронном виде легко изменять и обновлять, в то время как данные на бумаге после первой же ошибки нужно переписывать заново, если нужно созранить приемлемый вид документа

Поэтому данное приложение будет востребованным. По сравнению с другими похожими проектами можно выделить следующие отличительные черты: очень простой, понятный и не надоедливый дизайн, выделение прошедших и будущих задач, сбор статистики о выполнении задач в срок

Цель работы – изучение технологий разработки программ для персональных компьютеров, а также разработка приложения "Список задач"

Задачи:

1. Определить актуальность темы
2. Изучить и сравнить способы и технологий разработки компьютерных приложений
3. Научиться разрабатывать приложения для ПК
4. Разработать приложение "Список задач"

# Теоретическая часть

## План

План работы над проектом:

1. Спецификация.

На этом этапе происходит определение требований к результату проекта т.е. к разрабатываемому приложению

1. Определение технологии выполнения работы.

На этом этапе будет проведено сравнение технологий и средств, доступных для использовании при работе над проектом.

1. Обучение.

На этом этапе будет проводиться изучение обучающих материалов в области разработки компьютерных приложений.

1. Программирование.

После проведения спецификации и обучения программированию, будет создан готовый продукт при помощи языков программирования.

1. Отладка.

На данном этапе программист занимается отладкой программы, то есть поиском и устранением ошибок.

1. Тестирование.

Тестирование программы очень важно, поскольку в большинстве случаев программисты создают их не для личного применения, а для того, чтобы их программой пользовались другие. На этапе тестирования разработчик проверяет поведение разрабатываемого продукта на различных устройствах и операционных системах при различных наборах данных.

1. Сопровождение, внесение изменений, оптимизация.

После запуска программы в эксплуатацию осуществляется её сопровождение, т.е. внесение изменений на основе выявленных недочетов в процессе эксплуатации продукта, а также проводится оптимизация функционала или добавление нового

## Определение технологий кроссплатформенной разработки приложений для ПК

Что вообще из себя представляет кроссплатформенная разработка? Если вам нужно, чтобы ваше приложение работало сразу на нескольких операционных системах с минимальными затратами человеко-ресурса, то скорее всего вы прибегнете именно к этому. Вы разрабатываете одну программу, пишете один код, а он запускается на всех поддерживаемых платформах. Удобно! Но стоит различать кроссплатформенность и нативность.

Если вкратце, то кроссплатформенность — это способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами. Нативные программы в свою очередь пишутся для работы на определенной аппаратной или программной платформе. Из этого можно сделать вывод: если ваша программа будет работать с железом, то скорее всего даже в кроссплатформенной программе вам придется прописывать нативный функционал для конкретной платформы.

В чем же преимущество кроссплатформенной разработки? Во-первых, вам нужно задействовать меньше людей (не нужно содержать .Net разработчика, Swift/objective C разработчика и всех к ним прилагающихся), во вторых вы охватываете большее количество устройств, а значит больше людей смогут пользоваться вашей программой.

На помощь приходят уже созданные Фреймворки и технологии для разработки кроссплатформенных GUI. Ниже представлены некоторые из них.

**Electron JS**

До сих пор очень многие разработчики ведут споры относительно того, стоит ли использовать Electron в своих проектах или нет. Electron JS — это JavaScript фреймворк, позволяющий вам написать полноценную программу используя Web Технологии — HTML & CSS, JS. Electron JS представляет собой движок Chromium в котором и исполняется весь ваш код. У Electron'a есть один, но достаточно серьезный недостаток — большое потребление памяти, как физической, так и оперативной. Всем давно известно, какой прожорливый Chrome, а мы на него еще своего кода хотим навесить. Но если посмотреть с другой стороны: сегодня многие популярные приложения написаны на Electron'e — Slack, Skype, Discord, VSCode, Atom, Postman, Insomnia и т.д. А с учетом непрекращающегося роста мощности компьютеров все реже приходится слышать от пользователей, что «ваш хром съел всю мою память». Высокое потребление памяти не будет играть большой роли если продукт будет хорош в своей сфере, будет грамотно написан код и распределены процессы.

Плюсы Electron JS:

* Использование наработок из Web
* Просто найти (или "воспитать") специалиста
* Качественная (и русифицированная) документация
* Поддержка сообщества и GitHub

Недостатки:

* Высокое потребление памяти (физическая и ОЗУ)
* Легко написать плохой код
* Плохая нативность

**NW.JS**

Так же как и Electron, NW.JS позволяет вам создавать кроссплатформенные приложения с использованием Web технологий. Сегодня NW.JS Спонсируется компанией Intel, и разрабатывается сообществом. NW.JS не может похвастаться таким же богатым списком проектов как Electron, но это все равно очень хороший фреймворк для создания кроссплатформенных приложений. Так же как и Electron, NW.JS тащит за собой движок Chromium и все вытекающие из него проблемы. NW.JS очень похож по своему принципу на Electron, и у него похожий список преимуществ и недостатков но некоторые различия есть. По большему счету, единственное существенное различие между двумя проектами сводится к одному признаку — способности (или отсутствию таковой) «обезопасить» ваш исходный код. На мой взгляд, это единственная причина, почему разработчик может выбрать NW.js.

**QT/QML**

Qt — очень мощный набор инструментов для создания кроссплатформенных приложений на языке С++ и Python (а также Ruby, PHP, C# и др. но они в отличии поддерживаются сообществом). Qt уже достаточно старый фреймворк, но он продолжает активно развиваться и на нем написаны такие программы как: 2ГИС для Android, Kaspersky Internet Security, Virtual Box, Skype, VLC Media Player, Opera и другие. Известное многим линукс-пользователям окружение рабочего стола KDE Plasma тоже написано с использованием Qt. Qt имеет среду разработки — Qt Creator, которая включает в себя Qt Designer с помощью которого можно создавать графический интерфейс. Визуальное создание интерфейса позволяет легко и просто создавать интерфейс, перетаскивая различные виджеты (выпадающие списки, кнопки, переключатели) на форму. Qt имеет обширную документацию, активную поддержку сообщества и множество других преимуществ.

* Визуальный редактор интерфейса
* Огромное количество модулей в составе фреймворка
* Поддержка большого количества языков программирования (официально только Python и C++, остальные поддерживаются сообществом) -> хорошая нативность при грамотном коде
* Низкий порог вхождения

Недостатки QT:

* Большой вес приложения
* Высокая стоимость коммерческой лицензии
* Сложность написания программы на C++, возможная нестабильность при использовании других языков программирования
* Слабая поддержка Microsoft Windows
* Не самый привлекательный строгий «офисный» дизайн пользовательского интерфейса

**GTK**

GTK на ряду с Qt является одной из самых популярных библиотек для кроссплатформенной разработки элементов интерфейса для X Window Systems. Разработан Gnome Foundation и GNU. С использованием этой библиотеки написаны многие окружения рабочего стола и оконные менеджеры (Gnome, xfce MATE, Cinnamon, AfterStep, Marco и многие другие), и программы, например: Chromium, FireFox, MonoDevelop, Gimp и другие. GTK написан на языке C, но существуют обертки для многих языков программирование (C, C++, Python, Java и другие). Gtk имеет конструктор для визуальной разработки интерфейсов — Glade.

Преимущества GTK:

* Поддержка большого количества языков
* Большое и активное сообщество
* Много популярных проектов и примеров
* Визуальный редактор

Недостатки:

* Слабая совместимость с Microsoft Windows, требующая значительных усилий от разработчика при распространении приложения.

**AvaloniaUI**

AvaloniaUI — кроссплатформенный фреймворк для разработки пользовательского интерфейса на основе языка разметки XAML. Достаточно молодой и активно развивающийся фреймворк для кроссплатформенной разработки, разработанный Стивеном Кирком и поддерживаемый .NET Foundation и сообществом. Он поддерживается на Windows через .NET, на Linux через Xorg и на macOS. Сама Avalonia основана на WPF/UWP. Код пишется на языке C#. Благодаря XAML Avalonia позволяет создавать гибкие и стилизованные интерфейсы. Имеет хорошую документацию, хоть и не большое, но очень приветливое рускоязычное сообщество. Для Visual Studio существует расширение, которое содержит шаблоны проектов и элементов управления. С ними вы легко вникнете в суть и сразу сможете приступить к разработке интерфейсов.

AvaloniaUI является open-source проектом, который активно развивается и совершенствуется. Однако это не мешает уже использовать его в каких-либо крупных проектах.

Исходный код Avalonia полностью открыт — вы можете найти исходный код на официальном сайте Avalonia, выпущенной под лицензией MIT.

Avalonia написана на .NET Standard и на 100\% совместима со всеми версиями .NET Core и .NET Framework. Авалония очень быстро развивается чрезвычайно умными людьми по всему миру, которые работают с клиентами и знают, что им нужно. В Avalonia есть поддержка Visual Studio и Rider для создания файлов XAML и XAML intellisense, хотя Avalonia XAML intellisense все еще отстает от того, что может предложить WPF XAML intellisense. Avalonia позволяет создавать приложения, которые выглядят и ведут себя одинаково на разных платформах, но также поддерживает настройку для конкретной платформы. Несколько чрезвычайно популярных проектов с открытым исходным кодом .NET перешли с WPF на Avalonia, чтобы сделать свои продукты мультиплатформенными.

Преимущества AvaloniaUI:

* Хорошая нативность (+поддержка системных декораций)
* Активное и приветливое сообщество, как русскоязычное, так и англоязычное (чаты, форумы)
* Заимствование и сходство с WPF (Если вы работали с WPF вам будет легко разобраться с Авалонией, и при возникновении трудностей, код написанный для WPF скорее всего заработает у вас на авалонии)
* Стили (стили в Авалонии имеют сходство с CSS, что упрощает написание этих самых стилей). Это позволяет легко изменить дизайн программы, чтобы он соответствовал времени.

**WxWidgets/WxPython**

wxWidgets (ранее известная как wxWindows) — это кросс-платформенная библиотека инструментов с открытым исходным кодом для разработки кроссплатформенных на уровне исходного кода приложений, в частности для построения графического интерфейса пользователя (GUI).Она разработана не только для того, чтобы создавать GUI. Она также имеет набор классов для работы с графическими изображениями, HTML, XML документами, архивами, файловыми системами, процессами, подсистемами печати, мультимедиа, сетями, классы для организации многопоточности, отладки, отправки дампов и множество других инструментов. Имеет обертку для языка Python -WxPython. С использованием этой библиотеки написанны такие программы как: FileZilla, AudaCity, BitTorrent и другие.

Преимущества WxWidgets:

* Простое написание интерфесов
* поддержка разных языков программирования

Недостатки:

* Мало современной информации в интернете
* Не очень активное сообщество

**Tkinter**

Tkinter - кросс-платформенная событийно-ориентированная графическая библиотека разработанная Гвидо ван Россумом (создатель языка Python), позволяет написать вам простой, но функциональный интерфейс для вашего проекта на Python. По сути Tkinter представляет из себя пакет для Python, предназначенный для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя (graphical user interface – GUI), написанные на языке программирования Tcl и существуют реализации для разных языков (Tkinter приведен для примера).

Преимущества:

* Простое написание интерфесов
* Позволяет вам быстро написать простой GUI для вашей Python программы

Недостатки:

* Примитивные интерфейсы
* Для одного языка (но непосредственно Tk существует для разных языков)

Для разработки программы был выбран фреймворк AvaloniaUI, платформа .NET и язык С# - читается как «Си шарп». Это язык программирования от компании Microsoft. Вообще его решили разработать для проектов под Windows, но не так давно данный язык стал универсальным. Сейчас на этом языке пишут различные игры, десктопные программы, веб-сервисы, нейросети и графику для метавселенных. Разрабатывал язык знаменитый Андерс Хейлсберг.

Преимущества языка С#:

* Независим от железа.

Разработчику не потребуется адаптировать приложение под различные платформы и системы. За человека это будет выполнять специальная виртуальная машина, которая уже вшита NET Framework. Получается, что один и тот же код удастся запускать на различных гаджетах. Например, на телефонах, серверах, компьютерах и т.д.

* Хорошая совместимость с Windows.
* Управление памятью.

Для того чтобы приложение работало хорошо и без лагов, её требуется время от времени очищать от различных ссылок, кэша, ненужных объектов и т.д. В С\# этот процесс осуществляется автоматическим путем. Специалисту не нужно будет следить за тем, как расходуется память, бороться с её утечками или убирать мертвые куски кода.

* Строжайшая типизация.

Как только специалист объявит переменную в С\#, потребуется сначала указать, что именно в ней лежит – число, массив или строчка. Таким способом придется немного дольше разрабатывать, но ваш код будет работать правильно и предсказуемо. Числа будут взаимодействовать с числами, строки со строками и т.д. В языках, у которых слабая типизация свободы и драйва будет больше, но специалист может случайно пропустить ошибку. Она потом всплывет в готовом приложении и приведет к серьезным последствиям.

* Огромное сообщество.

На данном языке пишут множество специалистов во всем мире. В социальных сетях и различных сообществах можно задавать разные вопросы, обсуждать сложные темы или искать готовые решения. Некоторым программистам удается находить ментора, который может поделиться собственными знаниями и поможет намного быстрее выучить язык С\#.

* Синтаксический сахар.

В программе С\# существует множество различных методов, как сократить код, чтобы не нарушить логику приложения. Опытные специалисты называют подобные приемы «синтаксическим сахаром» - они позволяют выполнить код намного понятнее, проще и красивее.

## Концепция реактивного программирования

Реактивное программирование — это подход к разработке ПО, который строится на реагировании на события и на распространении событий. При этом модель реакции на события предполагает возможность простого распространения этих или трансформированных событий далее по системе. Ярким примером реализации реактивного подхода может служить таблица Excel. В ней существует цепочка вычислений, разделённая на несколько ячеек: при изменении значения одной из ячеек в цепочке значения в зависимых ячейках пересчитываются автоматически.

При разработке программы были использованы принципы реактивного программирования: приложение должно быть быстрым, отказоустойчивым и хорошо масштабироваться.

Выглядит как «мы за всё хорошее против всего плохого», верно?

Что подразумевается под этими словами:

* Отзывчивость. Приложение должно отдавать пользователю результат за полсекунды. Сюда же можно отнести и принцип fail fast — то есть, когда что-то идёт не так, лучше вернуть пользователю сообщение об ошибке типа «Извините, возникла проблема. Попробуйте позже», чем заставлять ждать у моря погоды. Если операция долгая, показываем пользователю прогресс-бар. Если очень долгая — «ваш запрос будет выполнен ориентировочно 18 марта 2042 года. Мы пришлём Вам уведомление на почту».
* Масштабируемость — это способ обеспечить отзывчивость под нагрузкой. Представим жизненный цикл какого-либо относительно успешного сервиса: Запуск — поток запросов маленький, сервис крутится на виртуалке с одним ядром. Поток запросов увеличивается — виртуалке добавили ядер и запросы обрабатываются в несколько потоков. Ещё больше нагрузка — подключаем batching — запросы к базе и жёсткому диску группируются. Ещё больше нагрузка — нужно поднимать ещё сервера и обеспечивать работу в кластере. В идеале система должна сама масштабироваться в большую или меньшую сторону в зависимости от нагрузки.
* За отказоустойчивость мы принимаем то, что живём в несовершенном мире и случается всякое. На случай, если в нашей системе что-то пойдёт не так, мы должны предусмотреть обработку ошибок и способы восстановления работоспособности.
* И наконец, нам предлагается всего этого добиться при помощи системы, архитектура которой основана на обмене сообщениями (message-driven).

## Фреймворк ReactiveUI

## Система контроля версий

Система контроля версий является прежде всего инструментом, а инструмент призван решать некоторый класс задач. Итак, система контроля версий – это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определенной версии. Мы хотим гибко управлять некоторым набором файлов, откатываться до определенных версий в случае необходимости. Можно отменить те или иные изменения файла, откатить его удаление, посмотреть кто что-то поменял. Как правило системы контроля версий применяются для хранения исходного кода, но это необязательно. Они могут применяться для хранения файлов совершенно любого типа.

Как хранить различные версии файлов? Люди пришли к такому инструменту как системы контроля версий не сразу, да и они сами бывают очень разные. Предложенную задачу можно решить с применением старого доброго copy-paste, локальных, централизованных или распределенных систем контроля версий.

**Copy-paste**

Известный метод при применении к данной задаче может выглядеть следующим образом: будем называть файлы по шаблону filename\_{version}, возможно с добавлением времени создания или изменения.

Данный способ является очень простым, но он подвержен различным ошибкам: можно случайно изменить не тот файл, можно скопировать не из той директории (ведь именно так переносятся файлы в этой модели).

**Локальная система контроля версий**

Следующим шагом в развитии систем контроля версий было создание локальных систем контроля версий. Они представляли из себя простейшую базу данных, которая хранит записи обо всех изменениях в файлах.

Одним из примеров таких систем является система контроля версий RCS, которая была разработана в 1985 году (последний патч был написан в 2015 году) и хранит изменений в файлах (патчи), осуществляя контроль версий. Набор этих изменений позволяет восстановить любое состояние файла. RCS поставляется с Linux'ом.

Локальная система контроля версий хорошо решает поставленную перед ней задачу, однако ее проблемой является основное свойство — локальность. Она совершенно не преднезначена для коллективного использования.

**Централизованная система контроля версий**

Централизованная система контроля версий предназначена для решения основной проблемы локальной системы контроля версий.

Для организации такой системы контроля версий используется единственный сервер, который содержит все версии файлов. Клиенты, обращаясь к этому серверу, получают из этого централизованного хранилища. Применение централизованных систем контроля версий на протяжении многих лет являлась стандартом. К ним относятся CVS, Subversion, Perforce.

Такими системами легко управлять из-за наличия единственного сервера. Но при этом наличие централизованного сервера приводит к возникновению единой точки отказа в виде этого самого сервера. В случае отключения этого сервера разработчики не смогут выкачивать файлы. Самым худшим сценарием является физическое уничтожение сервера (или вылет жесткого диска), он приводит к потерю кодовой базы.

Несмотря на то, что мода на SVN прошла, иногда наблюдается обратный ход — переход от Git'а к SVN'у. Дело в том, что SVN позволяет осуществлять селективный чекаут, который подразумевает выкачку лишь некоторых файлов с сервера. Такой подход приобретает популярность при использовании монорепозиториях, о которых можно будет поговорить позже.

**Распределенная система контроля версий**

Для устранения единой точки отказа используются распределенные системы контроля версий. Они подразумевают, что клиент выкачает себе весь репозиторий целиком заместо выкачки конкретных интересующих клиента файлов. Если умрет любая копия репозитория, то это не приведет к потере кодовой базы, поскольку она может быть восстановлена с компьютера любого разработчика. Каждая копия является полным бэкапом данных.

Все копии являются равноправными и могут синхронизироваться между собой. Подобный подход очень напоминает (да и является) репликацией вида master-master.

К данному виду систем контроля версий относятся Mercurial, Bazaar, Darcs и Git. Последняя система контроля версий и будет рассмотрена нами далее более детально.

**История Git**

В 2005 году компания, разрабатывающая систему контроля версий BitKeeper, порвала отношения с сообществом разработчиков ядра Linux. После этого сообщество приняло решение о разработке своей собственной системы контроля версий. Основными ценностями новой системы стали: полная децентрализация, скорость, простая архитектура, хорошая поддержка нелинейной разработки.

## Среда разработки

Интегрированная среда разработки (IDE) — это многофункциональная программа, которая поддерживает многие аспекты разработки программного обеспечения. Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки.

**Средства повышения производительности**

Вот несколько популярных возможностей Visual Studio, которые повышают производительность при разработке программного обеспечения:

* Волнистые линии и быстрые действия

Волнистые линии обозначают ошибки или потенциальные проблемы кода прямо во время ввода. Эти визуальные подсказки помогают немедленно устранить проблемы, не дожидаясь появления ошибок во время сборки или выполнения. Если навести указатель мыши на волнистую линию, на экран будут выведены дополнительные сведения об ошибке. Также в поле слева может отображаться лампочка, указывающая на наличие сведений о быстрых действиях для устранения ошибки.

* Очистка кода

Вы можете одним нажатием кнопки отформатировать код и применить к нему исправления, предложенные параметрами стиля кода, соглашениями в файле .editorconfig и (или) анализаторами Roslyn. Очистка кода, которая сейчас доступна только для кода C#, помогает устранять проблемы в коде перед переходом к его проверке.

* Рефакторинг

Рефакторинг включает в себя такие операции, как интеллектуальное переименование переменных, извлечение одной или нескольких строк кода в новый метод и изменение порядка параметров методов.

* IntelliSense

IntelliSense — это набор возможностей, отображающих сведения о коде непосредственно в редакторе и в некоторых случаях автоматически создающих небольшие отрывки кода. По сути, это встроенная в редактор базовая документация, которая избавляет от необходимости искать информацию в других источниках. Функции IntelliSense зависят от языка. Дополнительные сведения см. в руководствах по IntelliSense для C# , IntelliSense для Visual C++, IntelliSense для JavaScript и IntelliSense для Visual Basic.

* Поиск в Visual Studio

Иногда вам будет казаться, что в Visual Studio слишком много меню, действий и свойств. Чтобы быстро находить функции интегрированной среды разработки или элементы кода, в Visual Studio представлен единый компонент поиска (CTRL+Q).

* Live Share

Предоставляет возможности совместного редактирования и отладки в реальном времени независимо от типа приложения или языка. Вы можете мгновенно предоставлять общий доступ к своему проекту с поддержкой высокого уровня безопасности. Кроме того, вы можете предоставлять общий доступ к сеансам, экземплярам терминала, веб-приложениям на локальном компьютере, голосовым звонкам и т. п.

* Иерархия вызовов

В окне Иерархия вызовов показаны методы, вызывающие выбранный метод. Это может быть полезно, если вы собираетесь изменить либо удалить метод или хотите отследить ошибку.

* CodeLens

CodeLens помогает находить ссылки на код, изменения кода, связанные с кодом ошибки, рабочие элементы, проверки кода и модульные тесты — не выходя из редактора.

* Перейти к определению

Функция Перейти к определению позволяет перейти к расположению, где определена выбранная функция или тип.

* Показать определение

В окне Показать определение можно отобразить метод или определение типа, не открывая отдельный файл.

* Горячая перезагрузка

Горячая перезагрузка позволяет редактировать файлы кода приложения и сразу же применять изменения кода к работающему приложению.

# Практическая часть

## Установка среды разработки

**Начало работы. Visual Studio**

Чтобы облегчить написание, а также тестирование и отладку программного кода нередко используют специальные среды разработки, в частности, Visual Studio. Рассмотрим создание приложений на C# с помощью бесплатной и полнофункциональной среды Visual Studio Community 2022, которую можно загрузить с официального сайта Microsoft.

После загрузки запустим программу установщика. В открывшемся окне нам будет предложено выбрать те компоненты, которые мы хотим установить вместе с Visual Studio. Стоит отметить, что Visual Studio - очень функциональная среда разработки и позволяет разрабатывать приложения с помощью множества языков и платформ. В нашем случае нам будет интересовать прежде всего C# и .NET.

Чтобы добавить в Visual Studio поддержку проектов для C# и .NET 7, в программе установки среди рабочих нагрузок можно выбрать только пункт "Разработка классических приложений .NET" . Можно выбрать и больше опций или вообще все опции, однако стоит учитывать свободный размер на жестком диске - чем больше опций будет выбрано, соответственно тем больше места на диске будет занято.