Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**ВИДЖЕТ МОНИТОРИНГА СОСТОЯНИЯ АППАРАТНОГО ОБЕСПЕЧЕНИЯ КОМПЬЮТЕРА И ПРОГНОЗА ВРЕМЕНИ ЕГО РАБОТЫ**

БГУИР КП 1-40 04 01 01 025

Студент Филипеня А.Д.

Руководитель Марков А.Н.

Нормоконтролер Калиновская А.А.

Минск 2023

СОДЕРЖАНИЕ

[Введение 3](#_Toc147187568)

[1 Архитектура вычислительной системы 4](#_Toc147187569)

[1.1 Структура и архитектура вычислительной системы 4](#_Toc147187570)

[1.2 История, версии и достоинства 11](#_Toc147187571)

[1.3 Анализ вычислительной системы и обоснование выбора для написания программы 15](#_Toc147187572)

[2 Платформа программного обеспечения 17](#_Toc147187573)

[2.1 Структура и архитектура платформы 17](#_Toc147187574)

[2.2 История, версии и достоинства 20](#_Toc147187575)

[2.3 Анализ программного обеспечения для написания программы и обоснование выбора платформы 27](#_Toc147187576)

[Приложение А (обязательное) Листинг кода 29](#_Toc147187577)

# **ВВЕДЕНИЕ**

В современном информационном обществе компьютеры занимают центральное место, обеспечивая функционирование множества сфер человеческой деятельности. Эффективность работы компьютерных систем напрямую зависит от состояния и производительности их аппаратного обеспечения. Таким образом, мониторинг состояния аппаратной части компьютера и прогноз времени работы становятся актуальными задачами, нацеленными на обеспечение бесперебойной и стабильной работы информационных технологий.

Целью данного курсового проекта является разработка виджета, предназначенного для мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы. Этот виджет будет иметь практическое применение, такое как обеспечение пользователей важной информацией о состоянии и перспективах использования их компьютера.

Для достижения данной цели были поставлены следующие задачи:

1 Анализ существующих решений: провести обзор существующих программных средств мониторинга состояния аппаратного обеспечения компьютера и прогноза его работы с целью выявления основных достоинств и недостатков.

2 Разработка виджета мониторинга состояния аппаратного обеспечения компьютера: разработать программный виджет, способный собирать информацию о состоянии различных аппаратных компонентов компьютера, таких как процессор, оперативная память, жесткий диск и видеокарта.

3 Прогнозирование работы компьютера: разработать алгоритм прогнозирования работы компьютера на основе данных мониторинга.

4 Интерфейс пользователя: разработать пользовательский интерфейс виджета, который будет обеспечивать удобство и информативность его использования.

Данная курсовая работа направлена на повышения уровня информированности пользователей о состоянии своего компьютера, что, в свою очередь, способствует предотвращению возможных проблем и сбоев в его работе.

# **АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ**

## **1.1 Структура и архитектура вычислительной системы**

Вычислительная система (ВС) – результат интеграции аппаратных и программных средства, функционирующих в единой системе и предназначенных для решения задач определенного класса. Структура вычислительной системы состоит из пяти уровней:

1 Аппаратный уровень вычислительной системы: определяется набором аппаратных компонентов и их характеристиками, используемых вышестоящими уровнями иерархии и определяющими воздействие на них. К физическим ресурсам этого уровня относятся: процессор, оперативная память, внешние устройства. [1, 15 c.]

2 Уровень управления физическими ресурсами вычислительной системы: уровень управления физическими ресурсами ВС является первым уровнем системного программного обеспечения вычислительной системы, его назначение – систематизация и стандартизация правил программного использования физических ресурсов. На этом уровне обеспечивается создание программ управления физическими ресурсами. Для обеспечения управления физическими ресурсами используются программы, которые называются драйверами физического ресурса (устройства).

Драйвер физического устройства – программа, основанная на использовании команд управления конкретного физического устройства и предназначенная для организации работы с данным устройством.

3 Уровень управления логическими/виртуальными ресурсами вычислительной системы:

Логическое/виртуальное устройство (ресурс) – устройство, некоторые эксплуатационные характеристики которого (возможно, все) реализованы программным образом.

Драйвер логического/виртуального ресурса – программа, обеспечивающая существование и использование соответствующего ресурса.

Этот уровень ориентирован на пользователя. Команды данного уровня не зависят от физических устройств, они обращены к предыдущему уровню. На базе этого уровня могут создаваться новые логические ресурсы. При организации драйвера логического устройства могут использоваться драйверы физических или логических/виртуальных устройств. Система поддерживает иерархию драйверов. Многоуровневая унификация интерфейса.

Ресурсы вычислительной системы – совокупность всех физических и виртуальных ресурсов. Одной из характеристик ресурсов является их конечность, следовательно, возникает конкуренция за обладание ресурсом между его программными потребителями.

Операционная система – это комплекс программ, обеспечивающий управление ресурсами вычислительной системы.

Разветвленная иерархия виртуальных и физических устройств. Драйверы можно разделить на 3 группы:

– драйверы физических устройств;

– драйверы виртуальных устройств, обобщающих характеристики соответствующих физических устройств;

– драйверы виртуальных устройств, не имеющих аппаратной реализации (т.е. "полностью" виртуальных, например, драйвер файловой системы).

4 Уровень систем программирования:

Система программирования – это комплекс программ, обеспечивающий поддержание жизненного цикла программы в вычислительной системе.

Жизненный цикл программы в вычислительной системе состоит из четырех основных этапов:

– проектирование программного продукта;

– кодирование (программная реализация);

– тестирование и отладка.

Тестирование – проверка программы на тестовых нагрузках. Тест – заранее определенные входные данные. Тестовое покрытие – минимальный набор тестов, наиболее полно проверяющих программу.

Отладка – процесс поиска, локализации и исправления зафиксированных при тестировании ошибок.

Внедрение и сопровождение. Внедрение – установка программного комплекса на объектную вычислительную систему и его первичная настройка. Сопровождение – исправление недочетов разработки и внедрения программного комплекса (например, выпуск патчей).

5 Уровень прикладных систем

Прикладная система – программная система, ориентированная на решение или автоматизацию решения задач из конкретной предметной области. [1, 15-17 c.]

Цифровой компьютер состоит из связанных между собой процессоров, модулей памяти и устройств ввода-вывода.

**1.1.1** Выбранная вычислительная система

Вычислительная система, используемая для разработки данного курсового проекта, базируется на операционной системе *Windows 10* и аппаратном обеспечении ноутбука *HP* из серии *Pavilion*. Эта система включает в себя следующие компоненты:

1 Процессор *Intel Core i5*: процессор (центральный процессор) – основной вычислительный движок компьютера. Обеспечивает высокую производительность и является важной частью архитектуры системы.

2 Оперативная память (*RAM*) 16 ГБ: оперативная память служит для временного хранения данных, используемых программами во время их выполнения. Большой объем оперативной памяти позволяет эффективно обрабатывать множество задач и приложений одновременно. Это важный аспект архитектуры, так как от объемы и скорости оперативной памяти зависит производительность системы.

3 Носитель данных: ноутбук оснащен твердотельным накопителем *SSD*, который служит для хранения операционной системы, приложений и данных.

4 Видеокарта: встроенная видеокарта *Intel® Iris® Xe Graphics*. Эта видеокарта часто используется в ноутбуках с процессорами *Intel* последних поколений и предоставляет приемлемую производительность для повседневных задач и развлечений.

**1.1.2** Устройство центрального процессора

Процессор – мозг компьютера. Его задача – выполнять программы, находящиеся в основной памяти. Для этого он вызывает команды из памяти, определяет их тип, а затем выполняет одну за другой. Компоненты соединены шиной, представляющей собой набор параллельно связанных проводов для передачи адресов, данных и управляющих сигналов.

Процессор состоит из нескольких частей. Блок управления отвечает за вызов команд из памяти и определение их типа. Арифметика-логическое устройство (АЛУ) выполняет арифметические операции и логические операции. [2, 77 c.]

Трак данных состоит из регистров (обычно от 1 до 32), арифметико-логического устройства (АЛУ) и нескольких соединительных шин. Содержимое регистров поступает во входные регистры АЛУ. В них находятся входные данные АЛУ, пока АЛУ производит вычисления. Тракт данных – важная составляющая часть всех компьютеров.

Процессор – это только один из компонентов компьютера. В дополнении к нему требуются по крайней мере некоторое оперативное запоминающее устройство (ОЗУ), какой-то способ, позволяющий пользователю записать информацию в компьютер (устройство ввода), а также извлечь ее из него (устройство вывода). Кроме того, необходимо еще несколько микросхем для объединения всех компонентов. Сам микропроцессор мы можем представить в качестве «черного ящика», внутреннюю работу которого нам не обязательно досконально изучать, чтобы разобраться в его функциях. Для этого достаточно ознакомиться с входными и выходными сигналами, в частности с набором команд. [3, 476 c.]

Центральный процессор выполняет каждую команду за несколько шагов:

– вызывает следующую команду из памяти и переносит ее в регистр команд;

– меняет положение указателя команд, который теперь указывает на следующую команду;

– определяет тип вызванной команды;

– если команда использует слово из памяти, определяет местонахождение этого слова;

– при необходимости переносит слово в регистр центрального процессора;

– выполняет команду;

– переходит к первому шагу, чтобы начать выполнение следующей команды.

Такая последовательность шагов (выборка – декодирование – исполнение) является основой работы всех компьютеров. [2, 79 c.]

**1.1.3** Оперативная память

Оперативная память (*Random Access Memory, RAM*) используется для временного хранения данных и кода, которые активно используются программами во время их выполнения.

Память – это тот компонент компьютера, в котором хранятся программы и данные. Также часто встречается термин «запоминающее устройство». Без памяти, откуда процессоры считывают и куда записывают информацию, не было бы современных цифровых компьютеров.

Основной единицей измерения хранения данных в памяти является двоичный разряд, который называется битом. Бит может содержать 0 или 1. Эта самая маленькая единица памяти.

Применение двоичной системы счисления в компьютерах объясняется ее «эффективностью». При этом имеется в виду, что хранение цифровой информации может быть основано на отличиях между разными величинами какой-либо физической характеристики, например напряжения или тока. Чем больше величин нужно различать, тем меньше отличий между смежными величинами и тем менее надежна память. В двоичной системе счисления требуется различать всего две величины, следовательно, это – самый надежный метод кодирований цифровой информации. [2, 94 c.]

Память состоит из ячеек, каждая из которых может хранить некоторую порцию информации. Каждая ячейка имеет номер, который называется адресом. По адресу программы могут ссылаться на определенную ячейку. [2, 95 c.]

В компьютерах, в которых используется двоичная система счисления (включая восьмеричной и шестнадцатеричное представление двоичных числе), адреса памяти также выражаются в двоичных числах.

Ячейка – минимальная адресуемая единица памяти. В последние годы практически все производители выпускают компьютеры с 8-разрядными ячейками, которые называются байтами. Байты группируются в слова. [2, 96 c.]

**1.1.4** Вспомогательная память

Твердотельные накопители (*SSD*) и жесткие диски (*HDD*) являются носителями данных. *SSD* хранят данные во флэш-памяти, а жесткие диски – на магнитных дисках.

*HDD* использует вращающиеся магнитные диски для записи и чтения данных. Данные сохраняются на магнитных пластинах с помощью магнитных головок. *HDD* имеют меньшую скорость чтения и записи данных по сравнению с *SSD*, это связано с механическими движущимися частями и временем, необходимым для доступа к данным. *HDD* обычно имеют большую емкость по сравнению с *SSD* за ту же цену, что делает их привлекательными для хранения больших объемов данных, таких как медиафайлы и архивы. В то же время из-за механических компонентов *HDD* более подвержены износу и механическим повреждениям, особенно при сильных ударах или падениях.

Устройства на базе энергонезависимой флэш-памяти, часто называют твердотельными накопителями или *SSD*-дисками (*Solid State Disk*), постепенно начинают рассматриваться как высокоскоростная альтернативна традиционным технологиям магнитных дисков. [2, 119 c.]

Так как *SSD*-диски использует флэш-память для хранения данных, то данные доступны намного быстрее. *SSD* обеспечивают значительно более высокую скорость чтения и записи данных по сравнению с *HDD*. Они способствуют более быстрой загрузке операционной системы и приложений. Поскольку устройство не имеет подвижных частей, оно особенно хорошо подходит для ноутбуков (колебания в перемещении не влияют на его способность обращаться к данным).

**1.1.5** Графический процессор

Графический процессор (*Graphics Processing Unit, GPU*) – это специализированный вычислительный процессор, разработанный для обработки графических и видео данных. Он играет ключевую роль в воспроизведении графики, отображении изображений на мониторе, выполнении вычислительных задач, связанных с графикой, и ускорении обработки видео.

Наиболее важные части *GPU* на самом низком из возможных уровней: блоки *FP32 FP* 64 и/или *SPU* (специализированные вычислительные блоки); блоки *INT32*; регистры; встроенная память: *L0$* (редко в оборудовании потребительского класса), *L1$* используется совместно в разделяемой памятью (*Shared Memory*) на каждом вычислительном блоке, *L2$* на *GPU*; внешняя память (память устройства); планировщик команд; диспетчер команд *TMU*, *ROP* – фиксированные блоки конвейера и другие. Все вышеперечисленное сгруппировано во множество иерархий, которые в итоге составляют графические карты.

## **1.2 История, версии и достоинства**

**1.2.1** Архитектура x64

Архитектура x86-64, также известная как *Intel 64*, представляет собой расширение архитектуры x86, которая была разработана компаниями *AMD* и *Intel*. Ее история связана с потребностью в поддержке больших объемов памяти и более высокой производительности. Она была представлена в 2000 году.

С тех пор архитектура x86-64 пережила несколько версий и улучшений. Наиболее значимыми версиями были:

– *AMD64*: исходная версия архитектуры, представленная *AMD* в 2000 году;

– *Intel 64*: *Intel* также внедрила свою версию архитектуры x86-64, известную как *Intel 64*, также в начале 2000-х годов.

Архитектура x86-64 имеет ряд значительных преимуществ:

1 Поддержка 64-битных приложений: позволяет запускать и обрабатывать 64-битные приложения, что повышает производительность и позволяет эффективно управлять большими объемами данных.

2 Поддержка больших объемов памяти: способность адресации более 4 ГБ оперативной памяти, что является критически важным для современных вычислительных задач.

3 Совместимость: обратная совместимость с 32-битными приложениями и операционными системами, что обеспечивает плавный переход к 64-разрядной архитектуре.

Архитектура x86-64 внесла множество инноваций и технологических решений, включая:

1 Наборы инструкций: расширенные наборы инструкций, которые обеспечивают более высокую производительность и возможности оптимизации.

2 Виртуализация: встроенная поддержка технологий виртуализации для лучшей изоляции и управления виртуальными машинами.

3 Защита данных: механизмы защиты данных и аппаратная защита от вредоносных атак.

Архитектура x86-64, в том числе *Intel 64*, широко применяется в различных областях:

1 Серверы: в крупных серверных фермах для обработки данных и виртуализации.

2 Настольные и ноутбуки: в персональных компьютерах для выполнения разнообразных задач, включая игры, мультимедиа и офисные приложения.

3 Разработка и научные исследования: для высокопроизводительных вычислений и моделирования.

Архитектура x86-64 соперничает с другими архитектурами, такими как *ARM64*, в различных областях, и каждая из них имеет свои сильные стороны и применение.

**1.2.2** Оперативная память (*RAM*) 16 ГБ

История оперативной памяти началась почти одновременно с появлением самого компьютера. Сначала это были регистры вычислительного устройства, основная задача которых заключалась в хранении исходных параметров для выполняемой процессором операции, а затем – в приёме результата этой операции. Однако регистры были небольшими, и их было мало.

Для решения таких задач появилось выделенное устройство кратковременной памяти и специальный контроллер для сопряжения процессора с этим устройством. По мере того, как развивалась электроника, выделилось два направления реализации кратковременной памяти: на транзисторах и на конденсаторах.

Память на основе конденсаторов стали называть динамической памятью с произвольным доступом *DRAM* – *Dynamic Random Access Memory*. Динамическая – потому что надо перезаряжать конденсаторы. Произвольный доступ – потому что единовременно можно получить доступ на чтение или запись к любой ячейке по ее адресу, причем всегда за одно и то же время вне зависимости от расположения ячейки. К этому надо добавить ещё одну характеристику *DRAM* – асинхронность. Память на основе транзисторов стали называть статической памятью с произвольным доступом *SRAM*– *Static Random Access Memory*. У статической памяти *SRAM* на основе транзисторов тоже были свои задержки, но их не надо перезаряжать, и работают они синхронно. Поэтому кэш в процессоре значительно более производительный.

В начале 1990-х годов появилась память *SDRAM* – *Synchronous Dynamic Random Access Memory*. Теперь можно было управлять потоком данных при чтении и сохранении информации. Более того, теперь можно было синхронизировать работу процессора и памяти, пусть даже и с большой разницей в производительности. Кэширование и буферизация данных в процессоре нивелировали эту разницу.

В последние десятилетия, с развитием многоядерных процессоров и 64-разрядных операционных систем, объем и скорость оперативной памяти продолжают расти. Современные стандарты памяти *DDR3*, *DDR4* и *DDR5* предлагают высокую производительность и эффективность.

На протяжении истории компьютерных технологий существует множество версий оперативной памяти. Однако основные различия заключаются в следующем:

1 *DDR* (*Double Data Rate*) – первые стандарты, которые существуют с 2000 года.

2 *DDR2* – следующее поколение памяти, улучшившее скорость передачи данных.

3 *DDR3* – стандарт, который сделал память еще быстрее и эффективнее.

4 *DDR4* – последнее поколение на момент написания, предлагающее высокую производительность и низкое энергопотребление.

5 *DDR5* – ожидаемый стандарт, который будет продолжать увеличивать производительность.

Оперативная память (*RAM*) является одним из ключевых компонентов компьютерной системы и играет важную роль в ее производительности и функциональности. Вот некоторые из основных преимуществ оперативной памяти:

– быстрый доступ к данным, что ускоряет выполнение задач;

– поддержка многозадачности;

– снижение времени загрузки операционной системы и приложений;

– надежность и стабильность в работе системы;

– легкость расширения.

**1.2.3** *SSD*-диск *Micron 2210* 512 ГБ

Микрон (*Micron Technology*) – это американская компания, специализирующаяся на производстве полупроводников и твердотельных накопителей. Что касается *SSD* дисков *Micron*, их история связана с развитием технологии *NAND*-флэш памяти и ростом рынка твердотельных накопителей. Микрон начала производство *SSD* дисков в начале 2010-х годов. С течением времени *SSD* диски *Micron* стали обладать большей емкостью, более высокой скоростью чтения и записи, а также улучшенной надежностью благодаря развитию *NAND*-технологий и контроллеров. Рост объемов данных, требования к скорости и надежности, а также снижение стоимости *NAND*-флэш памяти оказали влияние на эволюцию *SSD* дисков *Micron*.

Серия *SSD* дисков *Micron 2210*, как правило, имеет несколько версий с различными объемами памяти и характеристиками скорости. Улучшения могут включать в себя увеличение емкости, повышение скорости чтения/записи и снижение энергопотребления.

К достоинствам *SSD*-дисков *Micron* можно отнести:

– высокая скорость чтения и записи данных;

– надежность и долгий срок службы;

– отсутствие подвижных деталей, что делает их устойчивыми к физическим ударам;

– относительно низкое энергопотребление;

– улучшение производительности системы благодаря быстрому доступу к данным.

Micron внедряет новые технологии *NAND*-флэш памяти, такие как *3D NAND*, и совершенствует контроллеры, чтобы улучшить производительность и надежность своих *SSD* дисков.

*SSD* диски *Micron 2210* применяются в различных областях, включая настольные компьютеры, ноутбуки, серверы и встраиваемые системы. Они эффективно работают в задачах, где требуется быстрый доступ к данным, надежность и производительность, таких как загрузка операционной системы, работа с приложениями и обработка данных.

**1.2.4** Видеокарты *Intel® Iris® Xe Graphics*

Архитектура графических процессоров (*GPU*) развивалась на протяжении многих лет. Первые *GPU* были разработаны в конце 20-го века и предназначались для ускорения графических вычислений на компьютерах. С развитием видеоигр и требований к графике, графические процессоры стали более мощными и функциональными.

*Intel® Iris® Xe Graphics* – это интегрированная графическая подсистема, разработанная компанией *Intel*. Эта видеокарта входит в состав процессоров *Intel* с архитектурой *Tiger Lake* и предназначена для обеспечения высококачественного воспроизведения видео, выполнения графически интенсивных задач и даже для небольших игр.

Некоторые ключевые характеристики *Intel® Iris® Xe Graphics*:

1 Архитектура *Xe*: графические процессоры *Iris Xe* базируются на архитектуре *Xe*, которая представляет собой современную и эффективную архитектуру с большим числом вычислительных ядер.

2 Поддержка *DirectX 12* и *Vulkan*: *Intel® Iris® Xe Graphics* обеспечивает поддержку современных графических *API*, таких как *DirectX 12* и *Vulkan*, что позволяет разработчикам создавать более реалистичные и производительные приложения.

3 *4K Ultra HD* и *HDR*: видеокарта способна воспроизводить контент в разрешении *4K Ultra HD* и поддерживает технологии *HDR* (*High Dynamic Range*) для более ярких и контрастных изображений.

4 Поддержка многозадачности: *Intel® Iris® Xe Graphics* обеспечивает поддержку многозадачности, что позволяет пользователям одновременно выполнять различные задачи, включая работу с графикой, видео и веб-браузерами.

5 Интегрированная графика: одной из особенностей *Iris Xe Graphics* является её интегрированный характер. Это означает, что видеокарта встроена непосредственно в процессор, что позволяет создавать более компактные и энергоэффективные ноутбуки и устройства.

*Intel® Iris® Xe Graphics* часто используется в ноутбуках с процессорами *Intel* последних поколений и предоставляет приемлемую производительность для повседневных задач и развлечений.

## **1.3 Анализ вычислительной системы и обоснование выбора для написания программы**

При выборе вычислительной системы важным фактором было обеспечение комфортных условий для работы, создания виджета и взаимодействия с аппаратным обеспечением. В данном случае выбранная вычислительная система отвечает этим требованиям по нескольким причинам:

1 Производительность процессора *Intel Core i5*: процессор *Intel Core i5* предоставляет хорошую производительность для выполнения широкого спектра задач. Его многозадачные способности позволяют комфортно работать с различными приложениями, включая разработку программного обеспечения, обработку данных и другие вычислительно интенсивные задачи.

2 Объем и скорость оперативной памяти (*RAM*): выбор оперативной памяти является важным аспектом архитектуры системы. 16 ГБ оперативной памяти, предоставляемой этой системой, обеспечивают достаточное количество ресурсов для эффективной работы с приложениями, включая среды разработки и виртуальные машины.

3 Скорость и емкость жесткого диска (*SSD*): присутствие *SSD* обеспечивает высокую скорость загрузки операционной системы и приложений, что улучшает общую производительность системы. Он также обеспечивает хорошую отзывчивость и быстрое открытие файлов.

4 Видеокарта: встроенная видеокарта *Intel® Iris® Xe Graphics* предоставляет приемлемую производительность для повседневных задач и развлечений.

5 Совместимость с программными средами: *Windows 10* является популярной операционной системой, обеспечивающей совместимость с большим количеством программ и сред разработки. Это важно для выполнения курсового проекта и обеспечения легкости в разработке и отладке программ.

5 Мобильность: ноутбук *HP* из серии *Pavilion* предоставляет портативность, что позволяет работать над проектом в различных местах и условиях, что может быть важно для эффективного выполнения задач.

В целом, выбранная вычислительная система обеспечивает баланс между производительностью, совместимостью, портативностью и доступностью, что делает ее максимально оптимальным решением для выполнения данного курсового проекта.

# **2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**2.1** **Структура и архитектура платформы**

**2.1.1** Выбранная платформа

Для разработки данного проекта был выбран язык программирования *C#* и среда программирования *Visual Studio*.

*Visual Studio* – это интегрированная среда разработки (*IDE*) от *Microsoft*, предоставляющая разработчикам широкий набор инструментов и ресурсов для создания, отладки и сопровождения приложений. Среда *Visual* *Studio* обладает удобным пользовательским интерфейсом, интегрированными средствами отладки, подсказками по коду и инструментами для управления версиями проектов.

Для работы с проектом была выбрана операционная система *Windows*. *Windows* – это операционная система, созданная корпорацией *Microsoft*, которая является одной из самых распространенных операционных систем в мире и обеспечивает широкую совместимость с приложениями, разработанными на языке *C#* с использованием *.NET Framework*. Это обеспечивает высокую доступность и удобство использования разработанных приложений для пользователей *Windows*.

**2.1.2** Язык программирования

Язык программирования – это формальный набор инструкций и правил, которые разработчик использует для написания программного кода, который впоследствии будет выполнен компьютером. Языки программирования предоставляют абстракции и средства для описания алгоритмов, обработки данных и выполнения задач.

Языки программирования предназначены для создания программного обеспечения, которое может выполнять разнообразные задачи и операции на компьютере. Они используются для разработки приложений для компьютеров, мобильных устройств и веб-серверов, а также для автоматизации задач, обработки данных, создания веб-приложений, разработки игр, научных и инженерных расчетов, системного программирования, исследовательской работы, работы с искусственным интеллектом и машинным обучением, создания мобильных приложений и многого другого. Язык программирования отличается от естественных языков тем, что предназначен для управления ЭВМ, когда естественные языки используются, прежде всего, для общения людей между собой.

**2.1.3** Интегрированная среда разработки

Интегрированная среда разработки (*IDE*) – это программное приложение, которое помогает программистам эффективно разрабатывать программный код. Оно повышает производительность разработчиков, объединяя такие возможности, как редактирование, создание, тестирование и упаковка программного обеспечения в простом для использования приложении. Так же как писатели используют текстовые редакторы, а бухгалтеры – электронные таблицы, разработчики программного обеспечения применяют *IDE*, чтобы упростить свою работу.

Для написания кода можно использовать любой текстовый редактор. Однако большинство интегрированных сред разработки (*IDE*) включают в себя функции, выходящие за рамки редактирования текста. Они предоставляют центральный интерфейс для общих инструментов разработчика, делая процесс разработки программного обеспечения гораздо более эффективным. Разработчики могут быстро приступить к программированию новых приложений вместо того, чтобы вручную интегрировать и настраивать различное программное обеспечение. Кроме того, нет необходимости изучать все инструменты, а можно сосредоточиться только на одном приложении.

Ниже перечислены некоторые причины, по которым разработчики используют *IDE*:

1 Автоматизация редактирования кода: в языках программирования существуют правила структурирования утверждений, поскольку *IDE* знает эти правила, она содержит множество интеллектуальных функций для автоматического написания или редактирования исходного кода.

2 Выделение синтаксиса: *IDE* может форматировать написанный текст, автоматически выделяя некоторые слова жирным или курсивом либо используя различные цвета шрифта, эти визуальные подсказки делают исходный код более читабельным и мгновенно информируют о случайных синтаксических ошибках.

3 Интеллектуальное завершение кода: *IDE* может предлагать завершить ввод кода, когда разработчик начинает набирать текст.

4 Поддержка рефакторинга: рефакторинг кода – это процесс реструктуризации исходного кода с целью сделать его более эффективным и читаемым без изменения его основной функциональности. *IDE* могут в некоторой степени использовать авторефакторинг, позволяя разработчикам быстро и легко улучшать свой код.

5 Автоматизация локальной сборки: *IDE* повышают производительность труда программистов, выполняя повторяющиеся задачи разработки, которые обычно являются частью каждого изменения кода. Ниже приведены примеры регулярных задач кодирования, которые выполняет *IDE*.

6 Компиляция: *IDE* компилирует или преобразует код в упрощенный язык, понятный операционной системе. Некоторые языки программирования реализуют компиляцию «точно в срок», при которой *IDE* преобразует понятный при прочтении код в машинный код внутри приложения.

7 Тестирование: *IDE* позволяет разработчикам автоматизировать модульные тесты локально, прежде чем программное обеспечение будет интегрировано с кодом других разработчиков и будут запущены более сложные интеграционные тесты.

8 Отладка: отладка – это процесс исправления любых ошибок или недочетов, которые выявляет тестирование. Одна из самых больших ценностей *IDE* для целей отладки заключается в том, что вы можете построчно просматривать код, по мере его выполнения и проверять поведение кода. В *IDE* также встроено несколько инструментов отладки, которые выявляют неполадки, вызванные человеческими ошибками, в режиме реального времени, даже когда разработчик набирает текст.

**2.1.4** Операционная система

С точки зрения программиста, операционная система – это программа, добавляющая ряд команд и функций к командам и функциям, предлагаемым уровнем архитектуры команд. Обычно операционная система организуется программно, но нет никаких веских причин, по которым ее, как микропрограммы, нельзя было реализовать.

Все команды уровня операционной системы доступны для прикладных программистов. Это – практически все команды более низкого уровня, а также новые команды, добавленные операционной системой. Новые команды называются системными вызовами. Они вызывают предопределенную службу операционной системы, в частности одну из ее команд. Например, типичный системный вызов может читать данные из файла. [2, 475 c.]

**2.2 История, версии и достоинства**

**2.2.1** Язык программирования *C#*

*C#* – объектно-ориентированный язык программирования, который был разработан компанией *Microsoft* в 1998 – 2001 годах как язык разработки приложений для платформы *Microsoft*.

*C#* прошёл через несколько версий и обновлений, каждая из которых добавляла новые функции и улучшения. Ключевые версии *C#*:

1 *C#* 1.0 (2000): первая версия *C#*, представленная вместе с .*NET Framework 1.0* и включающая в себя основные конструкции языка: классы, методы, свойства, события, возможность работать с объектами и наследование.

2 *C#* 2.0 (2005): вторая версия *C#* добавила в язык ряд новых функций: обобщенные типы, анонимные методы, итераторы и делегаты с поддержкой возвращаемых значений.

3 *C#* 3.0 (2007): третья версия *C#* ввела лямбда-выражения, выражения запросов *LINQ*, расширяющие методы.

4 *C#* 4.0 (2010): четвертая версия *C#* включила поддержку динамических типов, именованных и необязательных параметров.

5 *C#* 5.0 (2012): пятая версия *C#* внесла в язык поддержку асинхронного программирования с использованием ключевых слов *async* и *await*.

6 *C#* 6.0 (2015): шестая версия *C#* добавила инициализаторы свойств, вычисляемые свойства, интерполяцию строк и другие удобные функции.

7 *C#* 7.0 (2017): седьмая версия *C#* внесла следующие нововведения: сопоставление с шаблоном, выходные переменные и сокращенные лямбда-выражения.

8 *C#* 8.0 (2019): восьмая версия *C#* включила выражения *switch* с кратким синтаксисом, расширение для деконструкции, а также другие улучшения.

9 *C#* 9.0 (2020): девятая версия *C#* представила шаблоны, сопоставление с образцом для *switch*, инициализаторы и другие улучшения синтаксиса и функциональности.

10 *C#* 10.0 (2021): десятая версия *C#* представила расширенные шаблоны свойств, обработчики интерполированных строк, объявление пространства имен в пределах файла, улучшения в лямбда-выражениях.

11 *C#* 11.0 (2022): одиннадцатая версия *C#* представила необработанные строковые литералы, символы новой строки в выражениях интерполяции строк, шаблоны списков и другое.

*C#* пользуется широкой популярностью благодаря ряду выдающихся достоинств:

1 Объектно-ориентированная парадигма: *C#* поддерживает основные принципы объектно-ориентированного программирования, такие как наследование, полиморфизм и инкапсуляция. Это позволяет разработчикам создавать модульные, гибкие и расширяемые приложения. Классы являются основными строительными блоками программы, а объекты представляют экземпляры этих классов.

2 Безопасность типов: *C#* является языком со строгой типизацией, что означает, что каждая переменная и выражение имеют определенный тип данных. Это помогает предотвратить ошибки типов во время компиляции, улучшает надежность программы и повышает безопасность выполнения приложений

3 Управление памятью: В *C#* используется автоматическое управление памятью с помощью сборки мусора. Это означает, что разработчику не нужно явно освобождать память после использования объектов, так как сборщик мусора автоматически определяет, когда объекты больше не используются и освобождает память, занимаемую ими. Это упрощает разработку, уменьшает количество ошибок, связанных с управлением памятью, и повышает производительность приложений.

4 Многоязыковая поддержка: *C#* может взаимодействовать с другими языками, поддерживаемыми платформой *.NET*, такими как *Visual Basic .NET* и *F#*. Это позволяет разработчикам использовать различные языки в рамках одного приложения в зависимости от их предпочтений и требований проекта. Также это облегчает повторное использование кода и интеграцию с существующими проектами на разных языках.

5 Богатая стандартная библиотека: *C#* имеет обширную стандартную библиотеку классов, которая предоставляет различные функциональные возможности для работы с файлами, сетью, базами данных, графикой и другими аспектами разработки приложений. Благодаря этому разработчикам необходимо создавать множество функций "с нуля", а они могут использовать готовые классы и методы из стандартной библиотеки. Это существенно упрощает разработку приложений и повышает производительность разработчика.

6 Поддержка асинхронного программирования: *C#* предоставляет инструменты для работы с асинхронными операциями. Ключевые слова *async* и *await* позволяют создавать асинхронные методы, которые могут выполняться параллельно и не блокируют главный поток выполнения. Это особенно полезно для разработки реактивных и отзывчивых приложений, а также для улучшения производительности и отзывчивости пользовательского интерфейса.

7 Мощные инструменты разработки: для разработки приложений на *C#* существует множество интегрированных сред разработки (*IDE*), таких как *Microsoft Visual Studio* и *Visual Studio Code*, которые предоставляют широкий набор инструментов для отладки, автодополнения кода, управления проектами и других задач разработки. Эти инструменты значительно упрощают разработку и улучшают производительность разработчика.

8 Поддержка различных платформ: *C#* поддерживает разработку приложений для разных платформ, включая *Windows*, *macOS* и *Linux*. Благодаря использованию платформы *.NET Core*, разработчики могут создавать переносимый код, который может быть запущен на разных операционных системах без необходимости переписывания значительной части приложения.

Язык программирования *C#* является мощным инструментом для разработки разнообразных приложений, обладает удобным синтаксисом, богатыми возможностями и широкой поддержкой со стороны *Microsoft* и сообщества разработчиков.

**2.2.2** Среда разработки *Microsoft Visual Studio*

*Visual Studio* – это интегрированная среда разработки (*IDE*) от *Microsoft*, которая была представлена в 1997 году. Она была создана для облегчения разработки приложений для платформы *Windows*, а позднее стала поддерживать разработку для различных платформ и языков программирования.

Ключевые этапы её истории:

1 *Visual Studio 97*: первая версия *Visual Studio* была представлена в 1997 году, была выпущена в двух версиях: *Professional* и *Enterprise* – и предоставляла интегрированные инструменты для разработки на платформе *Windows*.

2 *Visual Studio 6.0*: это была одна из самых популярных версий *Visual Studio*, выпущенная в 1998 году. Она включала средства для разработки приложений на *Visual Basic*, *Visual C++* и других языках. Данная версия являлась основной средой разработки приложений под *Windows* от *Microsoft* до появления платформы *.NET*.

3 *Visual Studio .NET*: версия была представлена в 2002 году. Это был переломный момент, когда *Visual Studio* получила поддержку для платформы *.NET Framework*. Введены языки *C#* и *VB.NET*, которые стали основными языками для разработки *.NET*-приложений.

4 *Visual Studio 2005*: выпущена в 2005 году и вобрала в себя множество новых функций, включая поддержку *ASP.NET 2.0* и расширенные свойства отладки.

5 Visual Studio 2008: выпущена в 2008 году, представила поддержку .*NET Framework 3.5* и возможность создания приложений для *Windows Presentation Foundation* (*WPF*).

6 *Visual Studio 2012*: эта версия была выпущена в 2012 году и принесла поддержку *Windows 8*.

7 *Visual Studio 2015*: версия была выпущена в 2015 году, были введены инструменты для работы приложений для *Windows 10*, а также возможности для кроссплатформенной разработки.

8 *Visual Studio 2017*: версия была выпущена в 2017 году и представила новую установку и управление пакетами *NuGet*, а также инструменты для разработки мобильных приложений.

9 *Visual Studio 2019*: версия была выпущена в 2019 году и включает множество инноваций в области разработки, включая более интеллектуальные инструменты, расширенную поддержку *Git* и улучшенные возможности для облачной разработки.

10 *Visual Studio 2022*: версия была выпущена в 2022 году. *Visual Studio 2022* в *Windows* теперь является 64-разрядным приложением. Это означает, что вы можете открывать, редактировать, запускать и отлаживать даже самые большие и сложные решения без нехватки памяти.

*Visual Studio* имеет следующие достоинства:

1 Мощные инструменты разработки: *Visual Studio* предоставляет разработчикам широкий спектр инструментов, включая интегрированный отладчик, редактор кода, дизайнеры интерфейсов и тестовые средства.

2 Поддержка множества языков и платформ: среда поддерживает не только *C#* и *.NET*, но и множество других языков программирования, включая *C++*, *Python*, и *JavaScript*. Она также позволяет разрабатывать приложения для *Windows*, *Android*, *iOS*.

3 Богатое сообщество и экосистема: среда обладает активным сообществом разработчиков и большим количеством сторонних расширений и библиотек, что обогащает возможности разработки.

4 Поддержка *Git* и управление кодом: *Visual Studio* включает в себя инструменты для управления версиями и совместной работы над проектами с использованием системы контроля версий *Git*.

5 Интеграция с *Visual Studio Code*: для разработчиков, предпочитающих более легковесный кодовый редактор, *Visual Studio* интегрирована с *Visual Studio Code*, обеспечивая гибкость выбора среды разработки.

6 Обновления и поддержка: *Microsoft* регулярно выпускает обновления и улучшения для *Visual Studio*, что позволяет разработчикам использовать последние технологии.

Следовательно, *Visual Studio* остается одной из наиболее популярных и мощных сред разработки в мире программирования, предоставляя разработчикам инструменты и ресурсы для создания разнообразных приложений и сервисов. [сайт майкрософта)]

**2.2.3** Операционная система *Windows*

На сегодняшний день *Windows*, разработанная корпорацией *Microsoft*, является одной из самых популярных операционных систем и имеет богатую историю в несколько десятилетий:

1 *Windows* *1.0* (1985): первая версия *Windows* была выпущена в 1985 году и представляла собой графический интерфейс пользователя (*GUI*) для *MS-DOS*, включала базовые приложения и оконный интерфейс.

2 *Windows* *2.0* (1987): вторая версия *Windows* добавила поддержку приложений, работающих в разных окнах, возможность использовать сочетания клавиш, появились перекрывающиеся окна и возможность создания ярлыков на рабочем столе.

3 *Windows* *3.x* (1990-1994): с этой версией пришло множество улучшений, включая появление диспетчера программ и мультимедийных возможностей. Она стала широко распространенной и популярной версией. *Windows* 3.1 внесла некоторые улучшения в интерфейс, включая логотип *Windows*, а также поддержку сетей и широкоформатные шрифты.

4 *Windows 95* (1995): была крупным обновлением и принесла множество изменений в интерфейс, включая кнопку «Пуск» и панель задач. Она также включала поддержку *Plug* and *Play* и была первой версией *Windows*, которая не требовала запуска поверх *DOS*.

5 *Windows 98* (1998): эта версия улучшила поддержку *USB*-устройств, предоставила *Internet Explorer* как стандартный браузер и добавила поддержку *DVD*.

6 *Windows Me* (*Windows Millennium Edition*) (2000): была ориентирована на домашних пользователей и предоставила улучшенные мультимедийные функции, но она была менее стабильной по сравнению с предыдущими версиями.

7 *Windows XP* (2001): стала одной из самых популярных версий *Windows*. Она предоставила стабильность и продолжительную поддержку, а также улучшенный интерфейс. Служба поддержки *Windows XP* завершилась в 2014 году. Данная версия была разработана на основе ядра *Windows NT*, что придавало ей высокую степень стабильности и надежности.

8 *Windows Vista* (2007): были внесены изменения в интерфейс и безопасность, но данная версия столкнулась с критикой из-за производительности и совместимости.

9 *Windows 7* (2009): *Windows 7* улучшила интерфейс пользователя и внесла улучшения в производительность. Эта версия также осталась популярной на протяжении долгого времени.

10 *Windows 8.x* (2012-2013): *Windows 8* представила новый интерфейс *Metro*, который был ориентирован на сенсорные устройства. Эта версия была менее популярной из-за изменений в интерфейсе. Также были критикованы за отсутствие классической кнопки «Пуск», которая в *Windows 8.1* была возвращена и стала точкой входа на экран «Пуск».

11 *Windows 10* (2015): предоставила бесплатное обновление для пользователей *Windows 7* и *8*. Она внесла улучшения в интерфейс, включая возвращение меню «Пуск», а также интеграцию с облачными сервисами. Версия также получала регулярные обновления функций.

12 *Windows 11* (2021): последняя версия *Windows*, представлена в 2021 году. Она внесла изменения в интерфейс, включая новое меню "Пуск", улучшенные возможности многозадачности и дополнительные функции.

*Windows* имеет множество версий, включая *Home*, *Pro*, *Enterprise* и другие, предназначенные для разных категорий пользователей и компаний.

Windows обладает рядом значительных преимуществ и достоинств:

1 Популярность и широкая совместимость: *Windows* обеспечивает совместимость с большинством программ и устройств, что делает её популярной платформой для пользователей и разработчиков.

2 Удобный интерфейс: *Windows* предоставляет дружественный и интуитивно понятный интерфейс для пользователей всех уровней навыков.

3 Множество функций: *Windows* предоставляет множество функций, включая встроенные инструменты для работы с офисными приложениями, веб-браузерами и мультимедиа.

4 Широкий спектр аппаратных устройств: *Windows* поддерживает множество устройств, включая настольные ПК, ноутбуки, планшеты и смартфоны.

5 Обновления и поддержка: *Microsoft* регулярно предоставляет обновления и обеспечивает техническую поддержку для *Windows*, обеспечивая безопасность, актуальность и стабильность системы.

* 1. **Анализ программного обеспечения для написания программы и обоснование выбора платформы**

Для разработки виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы были рассмотрены несколько *IDE*:

– *Visual Studio*;

– *Rider*.

*Rider* – среда от *JetBrains* для работы с платформой *.NET*. Выпущена недавно, но уже приобрела много поклонников.

Из них была выбрана именно *Visual Studio* как основная среда разработки приложения на языке *C#* на основе следующих критериев:

1 Опыт работы: *Visual Studio* была выбрана в качестве *IDE* из-за существующего опыта работы с ней.

2 Интеграция с *.NET* и *C#*: *Visual Studio* разработана *Microsoft* и является официальной *IDE* для работы с языком *C#* и платформой *.NET*. Это обеспечивает высокую степень совместимости и интеграции, что позволяет использовать все мощные функции языка и фреймворка без ограничений.

3 Богатые инструменты разработки: в *Visual Studio* встроено множество инструментов и ресурсов, которые помогают разработчикам улучшать качество и производительность своего кода. Это включает в себя статический и динамический анализ кода, автоматическое завершение кода, систему управления версиями и многое другое.

4 Широкая поддержка сообщества: *Visual Studio* имеет огромное активное сообщество разработчиков, что облегчает доступ к информации, обмен опытом и получение ответов на вопросы. *Project Rider* имеет меньшее сообщество разработчиков по сравнению с *Visual Studio*, что усложняет поиск решений ряда проблем.

Курсовой проект разрабатывается с использованием языка программирования *C#*. Этот выбор основывается на комфорте в работе с этим языком программирования и знании инструментов, что способствует более эффективной разработке курсового проекта.

# **3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**

## **3.1 Обоснование необходимости разработки**

Доступ к данным об аппаратном обеспечение компьютера для рядового пользователя ограничен и довольно сложен. Стандартные средства диагностики и устранения неполадок, встроенные в операционную систему *Windows* дают неполную поверхностную информацию об отдельных аппаратных компонентах ПК, расположенную хаотично в разных местах системы. В связи с этим для получения более подробной и структурированной информации, возникает необходимость в использовании специализированного программного обеспечения, которое будет более эффективно решать данную задачу.

Ключевые аспекты обоснования необходимости разработки программного продукта “Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы”:

1 Повышение производительности и стабильности: системы мониторинга позволяют пользователям следить за состоянием аппаратного обеспечения и эффективно управлять его ресурсами. Это важно для поддержания высокой производительности и стабильности компьютера. В реальном времени виджет может предупреждать о перегреве, высокой загрузке процессора или других проблемах, которые могут вызвать сбой системы.

2 Предотвращение потери данных: отказы аппаратного обеспечения могут привести к потере важных данных. С помощью виджета мониторинга пользователи могут получать предупреждения о возможных проблемах и принимать меры, чтобы предотвратить потерю данных, что критически важно для пользователей.

3 Удовлетворение потребностей пользователей: разработка виджета мониторинга позволяет удовлетворить потребности пользователей в надежном и удобном инструменте для отслеживания состояния и производительности своих компьютеров.

4 Экономия времени и ресурсов: мониторинг состояния аппаратного обеспечения и прогноз времени его работы позволяют предотвращать простои и сбои в работе, что экономит время и ресурсы пользователей. Предупреждение проблем в ранней стадии делает обслуживание более эффективным и дешевым.

Таким образом, разработка программного продукта "Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы" обоснована множеством фундаментальных причин. Этот виджет обеспечивает пользователей мощным инструментом для мониторинга, оптимизации и поддержания их компьютеров, что способствует повышению производительности, стабильности и надежности вычислительных систем, а также экономии времени и ресурсов.

## **3.2 Технологии программирования, используемые для решения поставленных задач**

Для решения поставленных задач использованы следующие технологии программирования:

1 *С#*: язык программирования *C#* выбран для реализации курсового проекта из-за его широкой популярности и универсальности. *C#* является частью экосистемы *.NET* и предоставляет множество инструментов и ресурсов для создания приложений.

2 *Visual Studio*: среда разработки *Visual Studio* выбрана как основное средство разработки из-за ее мощных инструментов и возможностей интеграции. *Visual Studio* предоставляет удобное окружение для разработки, отладки и тестирования приложений на *C#*, а также обеспечивает поддержку создания приложений с использованием *.NET MAUI*.

3 *WMI* (*Windows Management Instrumentation*): *Windows Management Instrumentation* – это одна из базовых технологий, обладающая обширным набором инструментов для централизованного управления и наблюдения за работой различных компонентов оборудования компьютера как локально, так и удаленно. В основе инструментария *WMI* запросов лежит объектно-ориентированный подход к представлению компонентов системы. Инструментарий *WMI* является наиболее мощным средством для осуществления программного мониторинга за состоянием устройств компьютера при использовании платформы *.NET* *Framework*. Данная технология позволяет получать сведения об аппаратной части персонального компьютера (ПК) с помощью *WMI*-запросов. Ключевой особенностью этой технологии является то, что она даёт разработчикам программ возможность получать доступ к данным из разнообразных источников с помощью общей архитектуры. Источником сведений может быть часть аппаратного обеспечения, операционная система или программное приложение. [7, c.7]

Важной особенностью *WMI* является то, что хранящиеся в данной библиотеке объекты являются динамическими, то есть параметры этих объектов постоянно меняются, поэтому они не хранятся постоянно, а создаются по запросу. Хранилище таких свойств называется репозиторием и расположено в системной папке операционной системы. Так как инструментарий *WMI* создан по объектно-ориентированному принципу, то все данные операционной системы представлены в виде объектов их свойств и методов. Все классы группируются в пространства имен, которые иерархически упорядочены и логически связаны друг с другом по определенной технологии или области управления. Так как классы находятся в иерархической зависимости друг от друга, то классы-потомки могут наследовать или переопределять свойства классов родителей, а также добавлять свои свойства.

Для доступа к различным объектам *WMI* используются специальные запросы на языке *WQL* (*WMI Query Language*), который является одним из разновидностей *SQL*. Основное его отличие от *ANSI SQL* это невозможность изменения данных, то есть с помощью *WQL* возможна лишь выборка данных с помощью команды *SELECT*. Помимо ограничений на работу с объектами, *WQL* не поддерживает такие операторы как *DISTINCT*, *JOIN*, *ORDER*, *GROUP*, математические функции. Конструкции *IS* и *NOT* *IS* применяются только в сочетании с константой *NULL*.

4 *Open Hardware Monitor Library*: дополнительно для более подробного мониторинга аппаратного обеспечения используется библиотека *Open Hardware Monitor*. Эта библиотека предоставляет доступ к данным о состоянии аппаратного обеспечения, включая температуру процессора, скорость вентиляторов и другие параметры. Интеграция с *Open Hardware Monitor Library* расширяет возможности мониторинга и помогает предотвращать возможные проблемы с аппаратным обеспечением.

5 .*NET MAUI* (.*NET Multi-Platform App UI*): *NET MAUI* предоставляет единую платформу для создания пользовательских интерфейсов для мобильных и классических приложений с использованием языка программирования *C#* и языка разметки *XAML*.

*.NET MAUI* предоставляет коллекцию элементов управления, которые можно использовать для отображения данных, инициирования действий, указания действий, отображения коллекций, выбора данных и многого другого. В дополнение к коллекции элементов управления *.NET MAUI* также предоставляет:

1 Сложный механизм макета для проектирования страниц.

2 Несколько типов страницы для создания полнофункциональных типов навигации.

3 Предоставление прямого доступа к нативным *API* каждой платформы, в том числе к аппаратным возможностям платформ.

4 Возможность использования платформы *.NET* и языка программирования *C#*.

5 Поддержка привязки данных.

6 Горячая перезагрузка *.NET*, чтобы можно было изменить *XAML* и управляемый исходный код во время работы приложения, а затем наблюдать за результатом изменений без перестроения приложения.

В *.NET MAUI*, как и во многих других современных мобильных и кроссплатформенных фреймворках, поддерживается *MVVM* (*Model-View-ViewModel*) архитектурный паттерн. *MVVM* позволяет эффективно разделять логику приложения и пользовательский интерфейс, что упрощает разработку, тестирование и поддержку приложений.

## **3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением**

Архитектурные особенности аппаратного обеспечения, такие как процессор, оперативная память, жесткий диск и видеокарта, являются основой для разработки виджета мониторинга. Архитектура процессора и его характеристики определяют, насколько эффективно виджет сможет анализировать данные о состоянии системы. Объем и скорость оперативной памяти напрямую влияют на производительность и быстродействие виджета. *SSD* несет на себе ответственность за хранение операционной системы и данных, а видеокарта может использоваться для визуализации информации.

Операционная система *Windows* играет ключевую роль в архитектуре вычислительной системы, на которой будет работать разрабатываемый виджет. Взаимодействие с ОС предполагает не только учет её архитектурных особенностей, но и интеграцию виджета с операционной системой. Это позволяет предоставлять пользователям доступ к информации о состоянии аппаратного обеспечения и прогнозу времени его работы через удобные интерфейсные решения.

Слаженная работа виджета с аппаратным обеспечением и операционной системой обеспечивает оптимальную производительность. Это важно, так как мониторинг и прогнозирование требуют вычислительных ресурсов. Оптимизация кода и использование ресурсов компьютера максимально эффективно помогут достичь высокой производительности.

# **4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ**

## **4.1 Обоснование и описание функций программного обеспечения**

**4.1.1** Цели и задачи программного обеспечения

Программное обеспечение, разрабатываемое в рамках данного курсового проекта, ориентировано на решение конкретных задач, связанных с мониторингом состояния аппаратного обеспечения компьютера и прогнозом времени его работы. Это программное обеспечение представляет собой виджет мониторинга, который выполняет ряд важных функций:

1 Мониторинг состояния программного обеспечения: основной функцией разрабатываемого виджета является мониторинг состояния различных компонентов аппаратного обеспечения компьютера. В качестве объектов мониторинга выступают процессор, оперативная память, жесткий диск, видеокарта. Программное обеспечение будет отслеживать параметры, такие как:

– мощность, температура, частота и загрузка ЦПУ;

– количество занятой и свободной памяти, а также текущая загрузка ОЗУ;

– данные о жестком диске: объем, загрузка и т. д.

Эти функции обеспечат пользователей информацией о текущем состоянии компьютера и помогут выявить потенциальные проблемы или узкие места в системе.

2 Прогноз времени работы: разработанное программное обеспечение способно анализировать данные, собранные в процессе мониторинга, и на их основе предоставлять прогноз времени работы компьютера. В одном из окон виджета представлены текущие данные о батарее: зарядка, статус, оставшееся время до полной разрядки. Это важно для пользователей, так как позволяет предотвращать возможные сбои и неожиданные выключения, что может привести к потере данных.

3 Интерфейс пользователя: важной функцией программы является создание удобного и информативного пользовательского интерфейса. Пользователи смогут легко получать доступ к информации о состоянии своего компьютера и анализировать данные, предоставляемые виджетом.

Общая функциональность программного обеспечения тесно связана с его целью – обеспечить пользователям инструмент для контроля и мониторинга состояния и производительности их компьютеров. Это позволит им повысить надежность работы системы, предотвратить потенциальные проблемы и улучшить общую производительность аппаратного обеспечения.

**4.1.2** Полное описание функций

Мониторинг процессора: позволяет пользователю получить информацию о процессоре, включая его модель, тактовую частоту, количество ядер, а также данные по загрузке процессора, его частоте и температуре. Результат работы мониторинга процессора с помощью виджета мониторинга аппаратного обеспечения компьютера представлен в приложении Д на рисунке 1.

Для подтверждения достоверности полученных данных, к каждому снимку работы программы предоставлены изображения результатов, полученные из утилиты *Speccy*.

Speccy – это бесплатная программа, которая предоставляет подробную информацию о характеристиках компьютера. Она позволяет узнать основные технические параметры системы, такие как операционная система, процессор, память, видеокарта, жесткий диск и многое другое. При помощи Speccy можно получить детальную информацию о каждом компоненте и оценить общую производительность компьютера.

Предоставление данных из утилиты Speccy наряду с результатами разработанного виджета обеспечат подтверждение и точность отображаемой информации о состоянии аппаратного обеспечения компьютера.

Данные о процессоре, полученные из утилиты *Speccy*, представлены на рисунке 4.1.

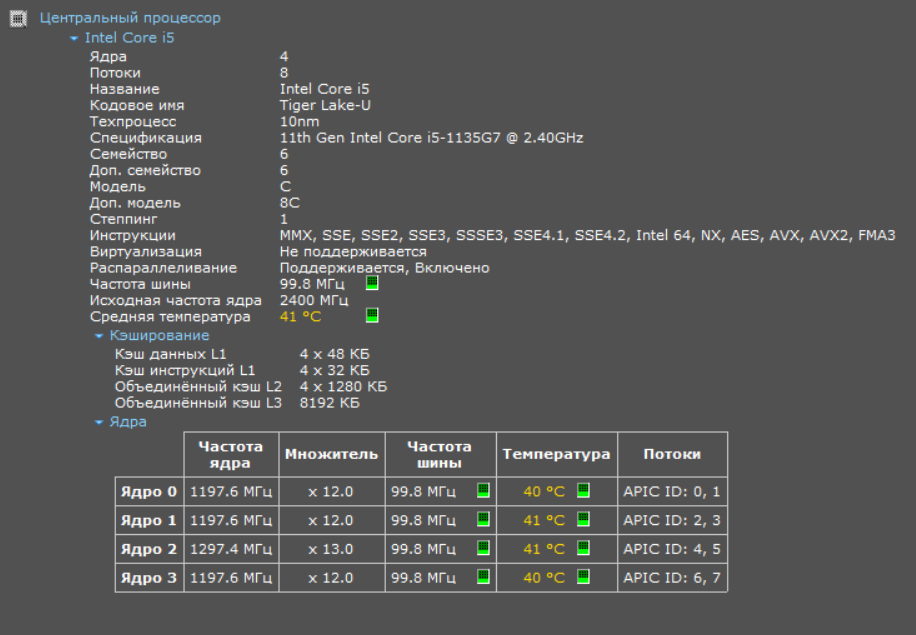


Рисунок 4.1 – Мониторинг процессора при помощи *Speccy*

Процент загрузки процессора, полученный из диспетчера задач, представлен на рисунке 4.2.

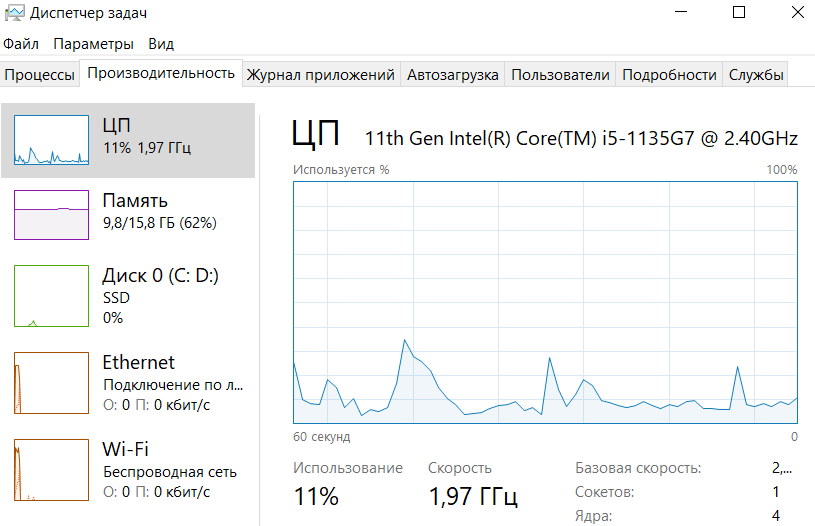


Рисунок 4.1 – Загрузка процессора

Мониторинг оперативной памяти: пользователь имеет возможность получить информацию о использовании оперативной памяти, включая общий объем и доступную память.

Результат мониторинга информации по оперативной памяти представлен в приложении Д на рисунке 2. Для сравнения на рисунке 4.3 также представлены данные, полученные из утилиты *Speccy*.

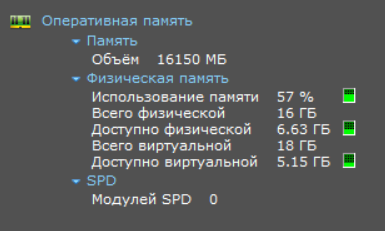


Рисунок 4.3 – Мониторинг оперативной памяти при помощи *Speccy*

Мониторинг видеокарты: позволяет пользователю получить информацию о графической карте, включая ее модель. Информация о графическом процессоре отображается в виджете, и пользователи могут следить за ее состоянием.

Информация о мониторинге графического процессора предоставлена в приложении Д на рисунке 3. Для сравнения на рисунке 4.4 представлены данные о графическом процессоре, полученные из утилиты *Speccy*.

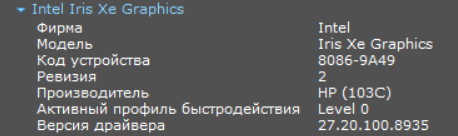


Рисунок 4.4 – Данные о видеокарте, полученные при помощи *Speccy*

Мониторинг жестких дисков и *SSD*: пользователь имеет возможность отслеживать доступное пространство на диске, использование и состояние жестких дисков. Что позволяет пользователю оперативно реагировать на уровень свободного пространства, рано выявлять возможные проблемы и, как следствие, предотвращать потери данных. Информация о мониторинге диска *C* и *D* при помощи разработанного виджетапредставлена в приложении Д на рисунке 4.

Для сравнения приведены данные о свойствах диска *C* на рисунке 4.5 и диске *D* на рисунке 4.6, данные получены из вкладки «Свойства» для этих дисков.

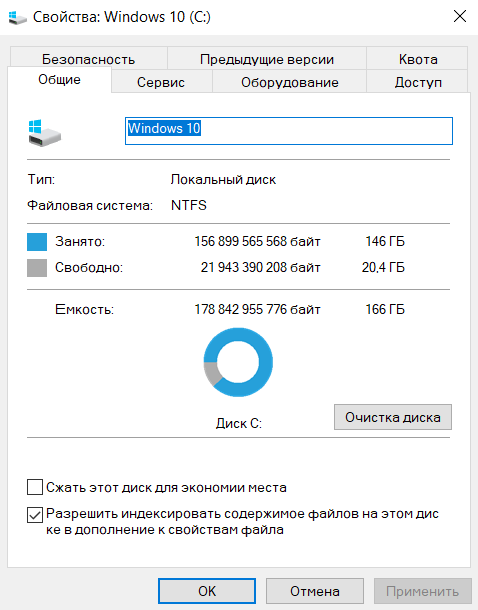


Рисунок 4.5 – Свойства диска *C*

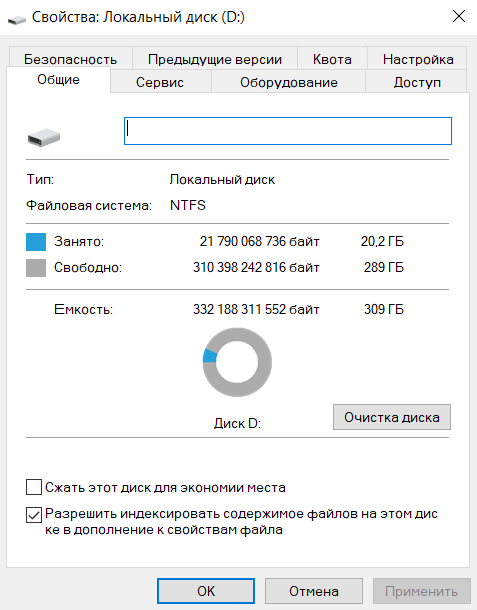


Рисунок 4.5 – Свойства диска *D*

Интерфейс пользователя: разработанный виджет предоставляет интуитивно понятный и легко воспринимаемый интерфейс. Простые элементы управления обеспечивают быстрый доступ к нужной информации. Каждая страница предоставляет ключевую информацию о соответствующем компоненте, переключение между различными страницами происходит при помощи удобного верхнего меню. Внешний вид пользовательского интерфейса можно увидеть на рисунках, предоставленных в приложении Д.

# **5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ**

**5.1 Общая структура программы**

Программа представлена в виде графического виджета, который пользователь сможет установить и запустить на своем компьютере. Графический интерфейс предоставляет информацию о текущем состоянии аппаратного обеспечения и прогнозе времени его работы.

Основные компоненты виджета включают в себя:

1 Панель мониторинга: это главная часть интерфейса, на которой отображаются данные о состоянии аппаратного обеспечения. Здесь можно увидеть информацию о процессоре, оперативной памяти, дисковом пространстве, видеокарте и других аппаратных компонентах.

2 Информация о времени работы: виджет также будет предоставлять информацию о прогнозе времени работы компьютера от аккумулятора (в случае ноутбуков). Это важно для пользователей, которые хотят знать, сколько времени осталось до разряда батареи.

3 Настройки и управление: пользователь может настраивать виджет, выбирая, какие аппаратные параметры.

Структура программы следует принципам объектно-ориентированного программирования. Код программы разделен на модули и классы, отвечающие за сбор информации, отображение данных, управление интерфейсом и другие задачи.

Следовательно, общая структура программы ориентирована на предоставление пользователю информации о состоянии аппаратного обеспечения и прогнозе времени его работы в удобном и интуитивно понятном виде.

**5.2 Описание функциональной схемы программы**

Описание функциональной схемы программы "Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноз времени его работы" включает следующие ключевые компоненты и функции:

– сбор информации о состоянии аппаратного обеспечения;

– обработка и анализ данных;

– отображение информации;

– настройка пользовательского интерфейса;

– прогноз времени работы (в случае при работе с ноутбуком).

Программа собирает информацию о состоянии аппаратного обеспечения компьютера, после чего обрабатывает и анализирует данные, для последующего вывода информации на экран пользователя. Пользователь в свою очередь может просмотреть всю необходимую информацию с помощью пользовательского интерфейса. Так же, если виджет используется на ноутбуке, пользователю будет доступна возможность просмотреть оставшееся время работы ноутбука в зависимости от заряда его батареи.

Функциональная схема разработанного проекта предоставлена в приложении Б.

**5.3 Описание блок схемы алгоритма программы**

Блок схема алгоритма программы "Виджет мониторинга состояния аппаратного обеспечения компьютера и прогноз времени его работы" представляет основные шаги и этапы алгоритма программы для виджета мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы. Блок схема алгоритма представлена в приложении В.

Программа начинается с инициализации и настройки, затем переходит в цикл мониторинга, где собирает информацию о компьютере, анализирует и обрабатывает данные о обрабатываемых данных, то есть о *CPU*, *GPU*, *RAM*, *SSD* и батарее, отображает их на виджете. Если компьютер является ноутбуком, программа также рассчитывает прогноз времени работы от батареи.

Взаимодействие с пользователем, настройки, интерфейс и взаимодействие с операционной системой дополняют основной алгоритм. Так же в блок схеме алгоритма есть цикл обновления данных, где программа проверяет, обновлены данные или нет. Если данные были обновлены, происходит выход из цикла и отображение данных на экран. Цикл по умолчанию применяется ежесекундно или в зависимости от выбранного времени обновления.

В итоге, разработанный виджет представляет собой интуитивно понятный, функциональный и настраиваемый инструмент, который удовлетворяет потребности разнообразной аудитории пользователей. Сочетание простоты использования с богатой функциональностью делает этот виджет неотъемлемой частью вычислительного опыта пользователей.

# **ЗАКЛЮЧЕНИЕ**

В рамках данного курсового проекта был разработан виджет мониторинга состояния аппаратного обеспечения компьютера, а также прогноза времени его работы на основе данных о заряде батареи при ее наличии. В процессе разработки использовались современные технологии, такие как язык программирования *C#*, среда разработки *Visual Studio*, платформа *.NET MAUI* для создания графического интерфейса пользователя и библиотеки *WMI*, *LibreHardwareMonitorLib* для сбора информации о состоянии аппаратного обеспечения, которое включает в себя *CPU*, *GPU*, *RAM*, *SSD* и аккумулятор при его наличии.

Интерфейс приложения включает в себя информацию о центральном процессоре, графическом процессоре, загрузке оперативной памяти и жесткого диска, а также о состоянии батареи, предоставляя пользователю полную картину о текущем состоянии компьютера. Также добавлены элементы визуализации, такие как полосы прогресса для более наглядного отображения уровня заполненности дисков.

Основной функционал приложения реализован с использованием объектно-ориентированного подхода, что обеспечивает удобство расширения и поддержки кода в будущем. Таймер обновления данных позволяет мгновенно получать актуальную информацию о состоянии системы каждую секунду, обеспечивая актуальность данных в интерфейсе.

Таким образом, созданный виджет предоставляет пользователю удобный и информативный инструмент для мониторинга состояния аппаратного обеспечения компьютера и прогноза времени его работы.

## **CПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Архитектура вычислительных систем [Электронный ресурс]: учебное пособие – Эл. изд. - Электрон. текстовые дан. (1 файл pdf: 77 с.). – Грейбо С.В., Новосёлова Т.Е., Пронькин Н.Н., Семёнычева И.Ф. 2019. – Режим доступа: <http://scipro.ru/conf/computerarchitecture.pdf>.

[2] Таненбаум, Э. Архитектура компьютера – 6-е изд. / Э. Таненбаум. СПб. : Питер, 2013. – 816 с.

[3] Петцольд, Ч. Код. Тайный язык информатики / Ч. Петцольд. – М. : Манн, Иванов и Фербер, 2021. – 448 с.

[4] Харрис, Д. М. Цифровая схемотехника и архитектура компьютера – 2-е изд. / Д. М. Харрис, С. Л. Харрис. – NY : Elsevier Inc, 2013. – 1662 c.

[5] Шитов, В. Н. Windows 10 : самый простой и понятный самоучитель / В. Шитов. – М. : Эксмо, 2023. – 464 с.

[6] Русинович, М. Внутреннее устройство Windows – 7-е изд. / М. Русинович [и др.]. – СПб. : Питер, 2018. – 944 с.

[7] Попов, А. Администрирование Windows с помощью WMI и WMIC. / А. В. Попов, Е. А. Шикин. СПб. : БХВ-Петербург, 2004. – 752 с.

[8] Стиллмен Эндрю, Грин Дженнифер Head First. Изучаем C#. 4-е изд. / Пер. с англ. Е. Матвеева. — СПб.: Питер, 2022. — 768 с.

[9] [Электронный ресурс]. – Режим доступа:  [https://coremission.net/gamedev/kak-rabotaet-gpu – Дата доступа 30.09.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2020.09.2023)

[10] [Электронный ресурс]. – Режим доступа: https://internet-lab.ru/windows\_os – Дата доступа: 02.09.2023

[11] [Электронный ресурс] – Режим доступа: https://gb.ru/posts/c\_sharp\_ides – Дата доступа: 15.10.2023

[12] [Электронный ресурс] – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/wmisdk/wmi-start-page – Дата доступа: 14.11.2023

[13] [Электронный ресурс] – Режим доступа: https://script-coding.com/WMI.html – Дата доступа: 14.11.2023

[14] [Электронный ресурс] – Режим доступа: http://sysengineering.ru/notes/administrirovanie-s-pomoschyu-wmi – Дата доступа: 17.11.2023

[15] [Электронный ресурс] – Режим доступа: https://www.nuget.org/packages/LibreHardwareMonitorLib/ – Дата доступа: 20.11.2023

[16] [Электронный ресурс] – Режим доступа: https://www.nix.ru/computer\_hardware\_news/hardware\_news\_viewer.html?id=211599 – Дата доступа: 22.11.2023

[17] [Электронный ресурс] – Режим доступа: https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-7.0 – Дата доступа: 25.11.2023

[18] [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/maui/ – Дата доступа: 29.11.2023

[19] Jesse Liberty .NET MAUI for C# Developers/ Jesse Liberty, Rodrigo Juares – UK: Packt Publishing, 2023. – 296 c.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код**

**CPU.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Management;

using System.Text;

using System.Threading.Tasks;

using LibreHardwareMonitor.Hardware;

namespace HardwareMonitoringLibrary

{

public class CPU

{

private static ObjectQuery query;

private static ManagementObjectSearcher searcher;

private static Computer computer;

public string Name { get; set; }

public int NumberOfCores { get; set; }

public int NumberOfThreads { get; set; }

public double Frequency { get; set; }

public float OccupancyPercentage { get; set; }

public float Temperature { get; set; }

public CPU()

{

query = new ObjectQuery("SELECT \* FROM Win32\_Processor");

searcher = new ManagementObjectSearcher(query);

computer = new Computer

{

IsCpuEnabled = true

};

GetTotalInfo();

GetOccupancyPercentage();

GetTemperature();

}

public void GetTotalInfo() {

foreach (ManagementObject info in searcher.Get())

{

Name = info["Name"].ToString();

NumberOfCores = Convert.ToInt32(info["NumberOfCores"]);

NumberOfThreads = Convert.ToInt32(info["ThreadCount"]);

Frequency = Convert.ToDouble(info["MaxClockSpeed"]);

}

}

//название

public void GetProcessorName()

{

foreach (ManagementObject info in searcher.Get())

{

Name = info["Name"].ToString();

}

}

//ядра

public void GetNumberOfCores()

{

foreach (ManagementObject info in searcher.Get())

{

NumberOfCores = Convert.ToInt32(info["NumberOfCores"]);

}

}

//потоки

public void GetNumberOfThreads()

{

foreach (ManagementObject info in searcher.Get())

{

NumberOfThreads = Convert.ToInt32(info["ThreadCount"]);

}

}

//частота

public void GetProcessorFrequency()

{

foreach (ManagementObject info in searcher.Get())

{

Frequency = Convert.ToDouble(info["MaxClockSpeed"]);

}

}

//загруженность

public static int GetProcessorLoad()

{

ObjectQuery query = new ObjectQuery("SELECT \* FROM Win32\_PerfFormattedData\_PerfOS\_Processor WHERE Name='\_Total'");

ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);

foreach (ManagementObject info in searcher.Get())

{

return Convert.ToInt32(info["PercentProcessorTime"]);

}

return -1;

}

public void GetTemperature()

{

computer.Open();

computer.Accept(new UpdateVisitor());

foreach (IHardware hardware in computer.Hardware)

{

foreach (ISensor sensor in hardware.Sensors)

{

if (sensor.SensorType == SensorType.Temperature)

{

Temperature = sensor.Value ?? 0;

}

}

}

computer.Close();

}

public void GetOccupancyPercentage()

{

computer.Open();

computer.Accept(new UpdateVisitor());

foreach (IHardware hardware in computer.Hardware)

{

foreach (ISensor sensor in hardware.Sensors)

{

if (sensor.SensorType == SensorType.Load)

{

OccupancyPercentage = sensor.Value ?? 0;

}

}

}

computer.Close();

}

}

}

**RAM.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Management;

using System.Text;

using System.Threading.Tasks;

namespace HardwareMonitoringLibrary

{

public class RAM

{

private static ObjectQuery query = new ObjectQuery("SELECT \* FROM Win32\_OperatingSystem");

private static ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);

public ulong TotalMemory { get; set; }

public ulong FreeMemory { get; set; }

public ulong LoadedMemory { get; set; }

public double MemoryLoadPercentage { get; set; }

public RAM()

{

UpdateMemoryInfo();

}

private void UpdateMemoryInfo()

{

foreach (ManagementObject info in searcher.Get())

{

TotalMemory = Convert.ToUInt64(info["TotalVisibleMemorySize"]);

FreeMemory = Convert.ToUInt64(info["FreePhysicalMemory"]);

LoadedMemory = TotalMemory - FreeMemory;

MemoryLoadPercentage = Convert.ToDouble(LoadedMemory) / Convert.ToDouble(TotalMemory) \* 100;

}

}

public ulong GetTotalMemory()

{

UpdateMemoryInfo();

return TotalMemory;

}

public ulong GetFreeMemory()

{

UpdateMemoryInfo();

return FreeMemory;

}

public double GetMemoryLoadPercentage()

{

UpdateMemoryInfo();

return Convert.ToDouble(LoadedMemory) / Convert.ToDouble(TotalMemory) \* 100;

}

public ulong GetLoadedMemory()

{

UpdateMemoryInfo();

return LoadedMemory;

}

}

}

**Disk.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HardwareMonitoringLibrary.Drive

{

public class Disk

{

public string Name { get; set; }

public string FileSystem { get; set; }

public long TotalSizeGb { get; set; }

public double FreeSpaceGb { get; set; }

public double UsedSpaceGb { get; set; }

public int OccupancyPercentage { get; set; }

public int FreePercentage { get; set; }

}

}

**Drive.cs**

using LibreHardwareMonitor.Hardware;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Management;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace HardwareMonitoringLibrary.Drive

{

public class Drive

{

private static ObjectQuery query = new ObjectQuery("SELECT \* FROM Win32\_DiskDrive");

private ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);

public int DriveSizeGb { get; set; }

public string DriveModel { get; set; }

public string DriveMediaType { get; set; }

public List<Disk> Disks { get; set; }

public Drive()

{

Disks = new List<Disk>();

GetDriveInfo();

GetDisksInfo();

}

public void GetDriveInfo()

{

foreach (ManagementObject info in searcher.Get())

{

DriveModel = info["Model"].ToString()!;

DriveMediaType = info["MediaType"].ToString()!;

DriveSizeGb = Convert.ToInt32(Convert.ToUInt64(info["Size"]) / (1024 \* 1024 \* 1024));

}

}

public void GetDisksInfo()

{

DriveInfo[] allDrives = DriveInfo.GetDrives();

foreach (DriveInfo drive in allDrives)

{

Disks.Add(new Disk

{

Name = drive.Name,

FileSystem = drive.DriveFormat,

TotalSizeGb = drive.TotalSize / (1024 \* 1024 \* 1024),

FreeSpaceGb = Math.Round(drive.TotalFreeSpace / (1024.0 \* 1024.0 \* 1024.0), 1), //округляю объем свободной памяти до 1 знака после запятой

FreePercentage = (int)Math.Round(drive.TotalFreeSpace \* 100 / Convert.ToDouble(drive.TotalSize)), //процент свободного пространства на диске, округляю процент

UsedSpaceGb = Math.Round((drive.TotalSize - drive.TotalFreeSpace) / (1024.0 \* 1024.0 \* 1024.0), 1), //округляю объем занятой памяти до 1 знака после запятой

OccupancyPercentage = (int)Math.Round((drive.TotalSize - drive.TotalFreeSpace) \* 100 / Convert.ToDouble(drive.TotalSize)) //процент загруженности диска, округляю процент

});

}

}

}

}

# **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Функциональная схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ Г**

**(обязательное)**

**Блок схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ Д**

**(обязательное)**

**Графический интерфейс пользователя**