

Разработка программы мониторинга системных ресурсов с использованием Python, PyQt6 и NVIDIA

Беликов Михаил Геннадьевич

*Приамурский государственный университет имени Шолом-Алейхема
Студент*

Языков Кирилл Александрович

*Приамурский государственный университет имени Шолом-Алейхема
Студент*

Аннотация

В статье рассматривается процесс разработки приложения для мониторинга системных ресурсов компьютера с использованием Python, библиотеки PyQt6 для графического интерфейса и модуля pynvml для работы с видеокартами NVIDIA. Исследование демонстрирует практическую реализацию приложения с отображением загрузки процессора, оперативной памяти, дисков и GPU в реальном времени. Особое внимание уделяется архитектуре приложения, работе с системными API, обработке исключений и настройке пользовательского интерфейса. Результатом работы является полнофункциональное кроссплатформенное приложение с возможностью гибкой настройки отображаемых параметров.

Ключевые слова: Python, PyQt6, мониторинг системы, pynvml, системные ресурсы.

Developing a system resource monitoring program using Python, PyQt6, and NVIDIA CUDA

Belikov Mikhail Gennadievich

*Sholom-Aleichem Priamursky State University
Student*

Yazykov Kirill Alexandrovich

*Sholom-Aleichem Priamursky State University
Student*

Abstract

The article discusses the process of developing an application for monitoring computer system resources using Python, the PyQt6 library for the graphical interface, and the pynvml module for working with NVIDIA video cards. The study demonstrates the practical implementation of an application that displays real-time CPU, RAM, disk, and GPU usage. Special attention is given to the application's

architecture, working with system APIs, handling exceptions, and customizing the user interface. The result is a fully functional cross-platform application with flexible customization options for the displayed parameters.

Keywords: Python, PyQt6, system monitoring, pynvml, system resources.

1 Введение

1.1 Актуальность

Мониторинг системных ресурсов является важной задачей для разработчиков, системных администраторов и обычных пользователей, позволяя отслеживать производительность системы, выявлять узкие места и оптимизировать работу приложений. Использование Python в сочетании с PyQt6 предоставляет мощный инструмент для создания кроссплатформенных GUI-приложений, а интеграция с pynvml позволяет расширить функционал для мониторинга графических процессоров NVIDIA. Такое решение актуально для геймеров, майнеров, инженеров машинного обучения и разработчиков графических приложений.

1.2 Обзор исследований

В.А. Машков и В.И. Литвиненко рассматривают применение Python для задач самодиагностики на системном уровне [1]. Г.Д. Бухарова и П.С. Комельских подчеркивают важность и необходимость включения Python в учебный процесс [2]. Г.С. Сейдаметов демонстрирует особенности использования Python в подготовке инженеров-программистов [3]. Э.А. Усеинов иллюстрирует применение объектно-ориентированного программирования в рамках изучения Python [4]. Разработка приложений с графическим пользовательским интерфейсом (GUI) на Python с использованием Tkinter также активно исследовалась. Например, А.С. Матвеева в своей статье рассматривает применение паттерна «Команда» для создания модульных и структурированных приложений на Tkinter, демонстрируя повышение гибкости, расширяемости, надежности и удобства использования [5].

1.3 Цель исследования

Целью работы является создание кроссплатформенного приложения для мониторинга системных ресурсов с графическим интерфейсом, поддержкой NVIDIA GPU и возможностью гибкой настройки отображаемых параметров в реальном времени.

2 Материалы и методы

Для решения поставленной цели выбрана платформа PyCharm [6] версии 3.10, благодаря её кроссплатформенности, обширной экосистеме библиотек и простоте интеграции с системными API. В качестве основы для создания графического интерфейса пользователя использовалась библиотека PyQt6, позволяющая разрабатывать современные, отзывчивые и настраиваемые десктопные приложения с поддержкой всех основных операционных систем.

Для получения данных о системных ресурсах применялся модуль `psutil`, предоставляющий удобный доступ к информации о загрузке центрального процессора, использовании оперативной памяти, состоянии дисков и других показателях. Для взаимодействия с графическими процессорами NVIDIA задействована библиотека `runvml` (NVIDIA Management Library), обеспечивающая программный доступ к статистике использования GPU, памяти, частотам и другим параметрам видеокарт через официальный API NVIDIA.

3 Результаты

Перед началом работы необходимо установить PyCharm (IDE), для легкой работы со структурированным кодом, а также подгружать нужные библиотеки, не прибегая к постоянному использованию командной строки или PowerShell (рис. 1).

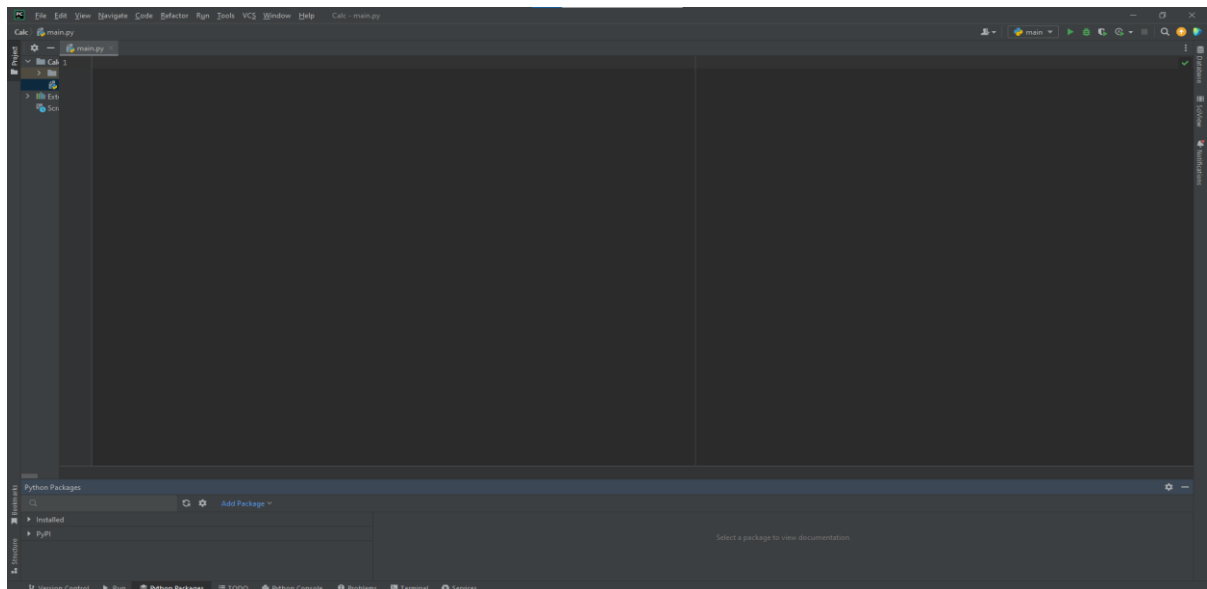


Рисунок 1 – Интерфейс IDE PyCharm

Перед началом работы следует создать файл `main.py`. Для дальнейшей работы потребуются библиотеки `PyQt6`, `psutil` и `nvidia-ml-py` для этого в терминал необходимо написать `pip install PyQt6 psutil nvidia-ml-py`, или установить через Python Packages (рис. 2).

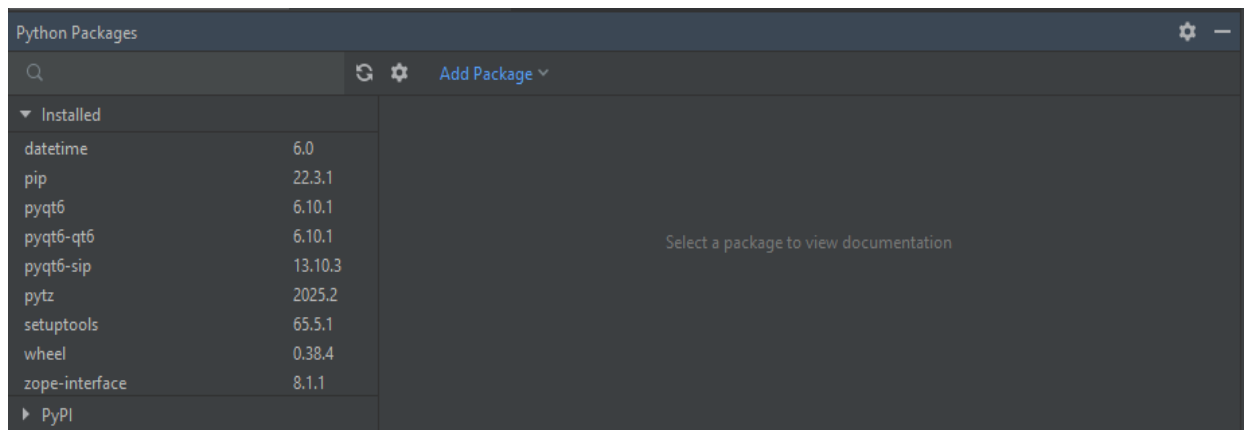


Рисунок 2 – Установка библиотек PyQt6 psutil nvidia-ml-py в Python Packages

Перед работой с графической оболочкой следует создать файл, в котором написан код для будущей программы main.py (рис. 3).

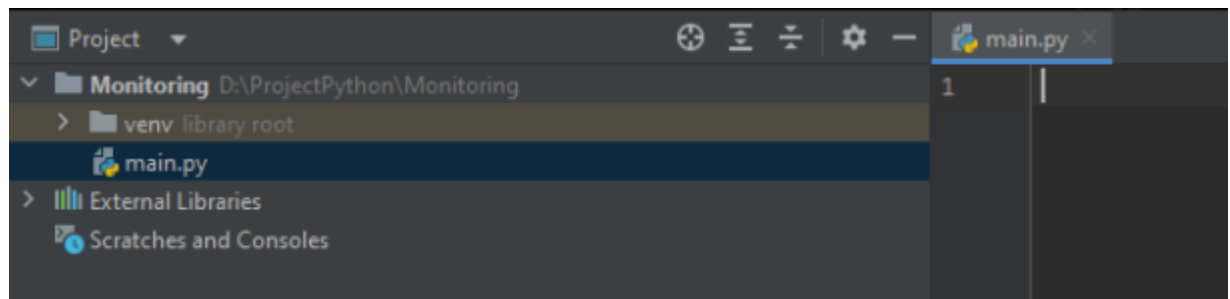


Рисунок 3 –Название файла main.py

Инициализация приложения мониторинга. Код начинается с импорта необходимых библиотек: sys, psutil и компонентов PyQt6 для создания интерфейса. Проверяется доступность библиотеки runvm1 для работы с видеокартами NVIDIA. Создаётся главное окно класса ARM_Monitor с заголовком и размерами. Настраивается словарь параметров отображения (CPU, RAM, диск, GPU). Формируется интерфейс с двумя вкладками: "Главная" для показа данных и "Настройки" для управления отображаемыми компонентами. Вызываются методы инициализации этих вкладок.

```
1 import sys
2 import psutil
3 from PyQt6.QtWidgets import (
4     QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,
5     QLabel, QCheckBox, QGroupBox, QGridLayout, QScrollArea
6 )
7 from PyQt6.QtCore import QTimer, Qt
8
9 NVIDIA_AVAILABLE = False
10 try:
11     import pynvml
12     pynvml.nvmlInit()
13     NVIDIA_AVAILABLE = True
14 except Exception:
15     pass
16
17
18 class ARM_Monitor(QMainWindow):
19     def __init__(self):
20         super().__init__()
21         self.setWindowTitle("Мониторинг АРМ")
22         self.resize(740, 580)
23
24         self.settings = {
25             'cpu': True,
26             'ram': True,
27             'disk': True,
28             'gpu': NVIDIA_AVAILABLE
29         }
30
31         self.tabs = QTabWidget()
32         self.setCentralWidget(self.tabs)
33
34         self.main_tab = QWidget()
35         self.settings_tab = QWidget()
36
37         self.tabs.addTab(self.main_tab, "Главная")
38         self.tabs.addTab(self.settings_tab, "Настройки")
39
40         self.init_main_tab()
41         self.init_settings_tab()
```

Рисунок 4 – Код файла main.py (часть 1)

```
42
43     self.update_timer = QTimer()
44     self.update_timer.timeout.connect(self.update_main_display)
45     self.update_timer.start(1000)
46
47     self.update_main_display()
48
49     def init_main_tab(self):
50         layout = QVBoxLayout()
51
52         self.main_labels = {}
53         for key in ['cpu', 'ram', 'disk', 'gpu']:
54             label = QLabel("")
55             label.setStyleSheet("font-family: Consolas, monospace; font-size: 11pt; padding: 6px;")
56             label.setTextInteractionFlags(Qt.TextInteractionFlag.TextSelectableByMouse)
57             self.main_labels[key] = label
58             layout.addWidget(label)
59
60         scroll = QScrollArea()
61         container = QWidget()
62         container.setLayout(layout)
63         scroll.setWidget(container)
64         scroll.setWidgetResizable(True)
65
66         main_layout = QVBoxLayout()
67         main_layout.addWidget(scroll)
68         self.main_tab.setLayout(main_layout)
69
70     def init_settings_tab(self):
71         layout = QVBoxLayout()
72         group = QGroupBox("Отображаемые компоненты:")
73         grid = QGridLayout()
74
75         options = [
76             ('cpu', 'ЦП (CPU)'),
77             ('ram', 'ОЗУ (RAM)'),
78             ('disk', 'Диски (SSD/HDD)'),
79             ('gpu', 'Видеокарта (GPU NVIDIA)')
80         ]
```

Рисунок 5 – Код файла main.py (часть 2)

```
self.checkboxes = {}
for i, (key, name) in enumerate(options):
    cb = QCheckBox(name)
    cb.setChecked(self.settings[key])
    if key == 'gpu' and not NVIDIA_AVAILABLE:
        cb.setEnabled(False)
        cb.setText(name + " (не обнаружена)")
    cb.stateChanged.connect(lambda state, k=key: self.toggle_setting(k, state))
    self.checkboxes[key] = cb
    grid.addWidget(cb, i // 2, i % 2)

group.setLayout(grid)
layout.addWidget(group)
layout.addStretch()
self.settings_tab.setLayout(layout)

def toggle_setting(self, key, state):
    self.settings[key] = (state == Qt.CheckState.Checked.value)

def update_main_display(self):
    if self.settings['cpu']:
        cpu_percent = psutil.cpu_percent(interval=0.01)
        freq = psutil.cpu_freq()
        freq_str = f"{freq.current:.0f} MHz" if freq else "N/A"
        cores = psutil.cpu_count(logical=True)
        text = f"• <b>CPU</b>: {cpu_percent:.1f}% | Частота: {freq_str} | Ядра: {cores}"
        self.main_labels['cpu'].setText(text)
        self.main_labels['cpu'].show()
    else:
        self.main_labels['cpu'].hide()

    if self.settings['ram']:
        mem = psutil.virtual_memory()
        used_gb = mem.used / (1024**3)
        total_gb = mem.total / (1024**3)
        text = f"• <b>RAM</b>: {used_gb:.2f} GB / {total_gb:.2f} GB ({mem.percent:.1f}%)"
        self.main_labels['ram'].setText(text)
        self.main_labels['ram'].show()
```

Рисунок 6 – Код файла main.py (часть 3)

```

159         name = "Неизвестная GPU"
160     try:
161         util = pynvml.nvmlDeviceGetUtilizationRates(handle)
162         gpu_util = util.gpu
163         mem_util = util.memory
164     except Exception:
165         gpu_util = mem_util = "N/A"
166     try:
167         mem_info = pynvml.nvmlDeviceGetMemoryInfo(handle)
168         mem_used_gb = mem_info.used / (1024**3)
169         mem_total_gb = mem_info.total / (1024**3)
170     except Exception:
171         mem_used_gb = mem_total_gb = "N/A"
172     try:
173         clock = pynvml.nvmlDeviceGetClockInfo(handle, pynvml.NVML_CLOCK_GRAPHICS)
174     except Exception:
175         clock = "N/A"
176
177     gpu_texts.append(
178         f" * {name} – GPU: {gpu_util}%, Память: {mem_util}%, "
179         f"Используется: {mem_used_gb:.2f}/{mem_total_gb:.2f} GB, Частота: {clock} MHz"
180     )
181     if gpu_texts:
182         text = " * <b>GPU (NVIDIA)</b>:\n" + "\n".join(gpu_texts)
183     else:
184         text = " * <b>GPU</b>: нет устройств"
185     except Exception as e:
186         text = f" * <b>GPU</b>: ошибка ({str(e)})"
187     self.main_labels['gpu'].setText(text)
188     self.main_labels['gpu'].show()
189 else:
190     self.main_labels['gpu'].hide()
191
192 def closeEvent(self, event):
193     if NVIDIA_AVAILABLE:
194         try:
195             pynvml.nvmlShutdown()
196         except Exception:
197             pass
198     event.accept()
199
200
201 if __name__ == "__main__":
202     app = QApplication(sys.argv)
203     window = ARM_Monitor()
204     window.show()
205     sys.exit(app.exec())

```

Рисунок 7 – Код файла main.py (часть 4)

Оптимизация пользовательского интерфейса. Текущая версия интерфейса выполнена в строгом минималистичном стиле, обеспечивающем интуитивную понятность и быстрый доступ к ключевым функциям мониторинга. Лаконичное расположение элементов, отсутствие избыточных деталей и чёткая визуальная иерархия делают интерфейс сбалансированным и функционально завершённым. Существующая структура не требует значительных изменений, поскольку оптимально решает поставленные задачи: предоставляет пользователю актуальную информацию о системных ресурсах в реальном времени и позволяет гибко настраивать отображаемые параметры без перегруженности интерфейса излишними элементами управления (рис. 8).

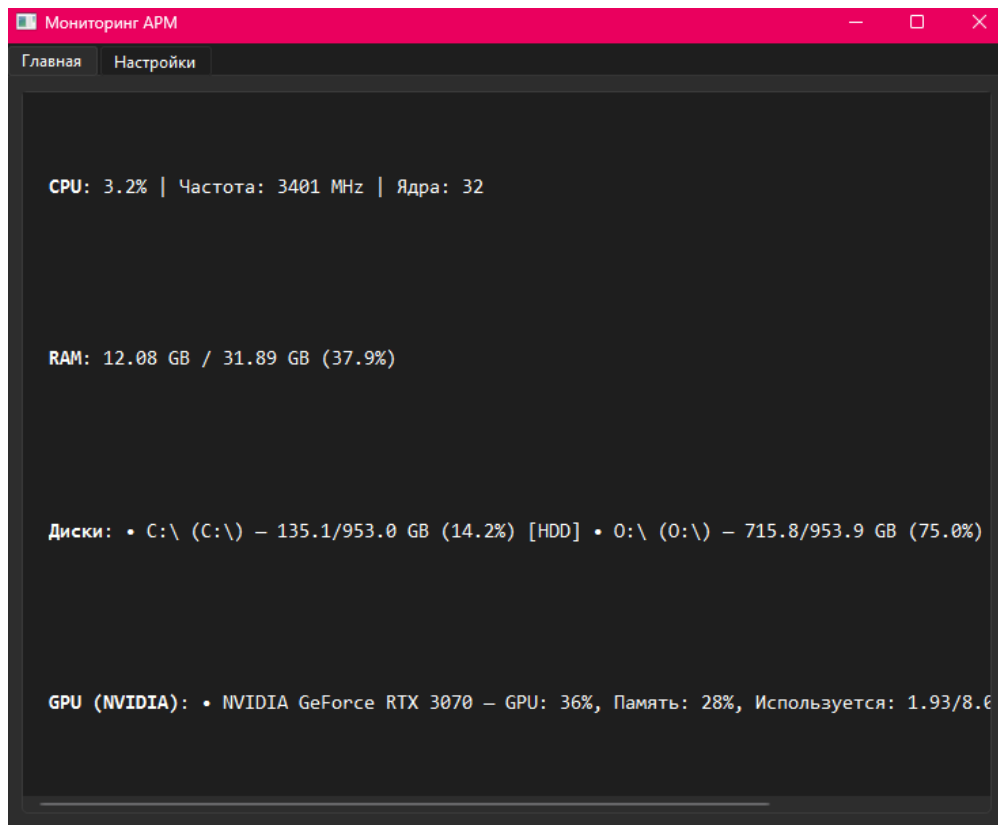


Рисунок 8 – Код файла themes.py

Последним шагом следует упаковка приложения в один исполняемый файл .exe с помощью библиотеки pyinstaller. Чтобы реализовать данную операцию, для начала необходимо установить данную библиотеку через `pip install pyinstaller` или в Python Packages (рис. 9).

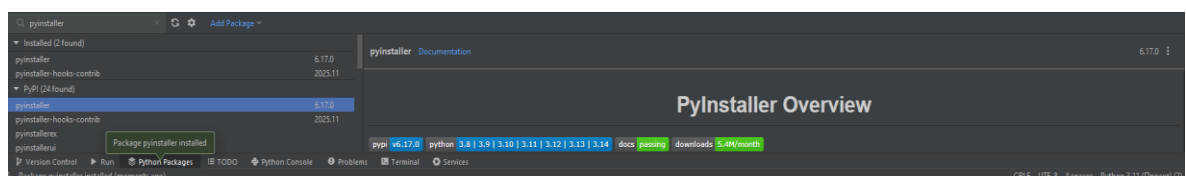


Рисунок 9 – Установка библиотеки pyinstaller

Последним шагом необходимо в консоли терминала прописать команду: `pyinstaller --onefile --windowed --name "Мониторинг АРМ" main.py` (рис. 10).

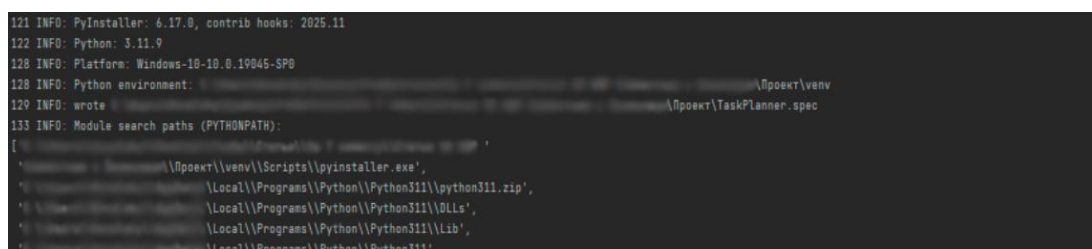


Рисунок 10 – Упаковка приложения

При запуске приложения открывается приложение, имеющая две вкладки «Главная», «Настройки» (рис. 11-12).

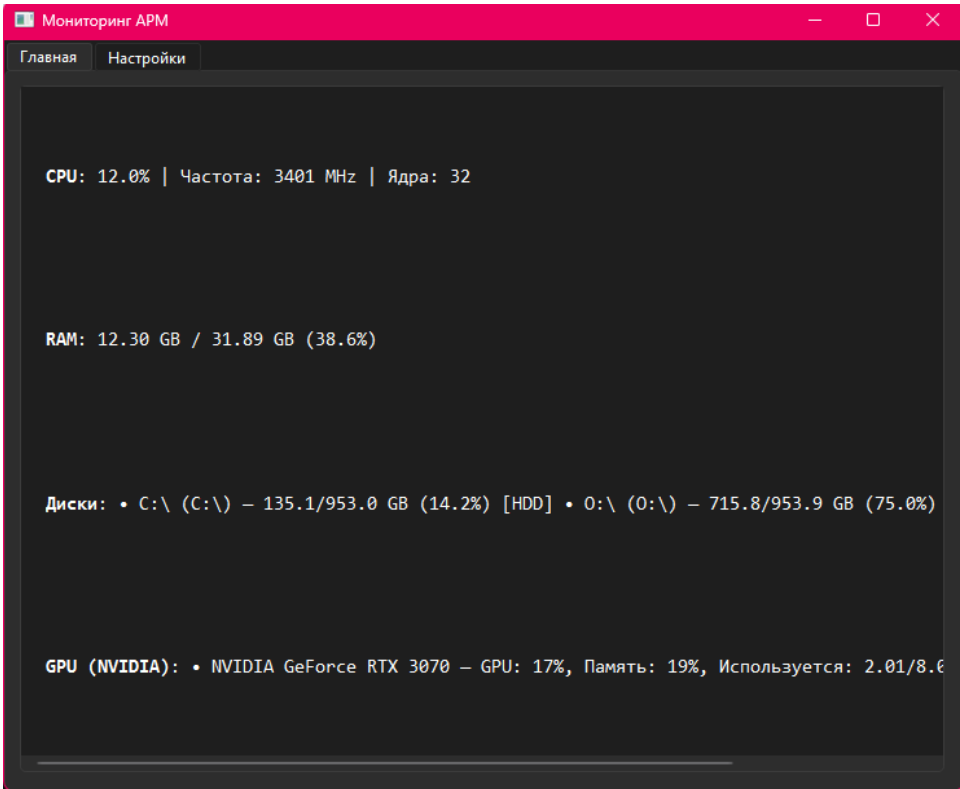


Рисунок 11 – Отображение системных характеристик

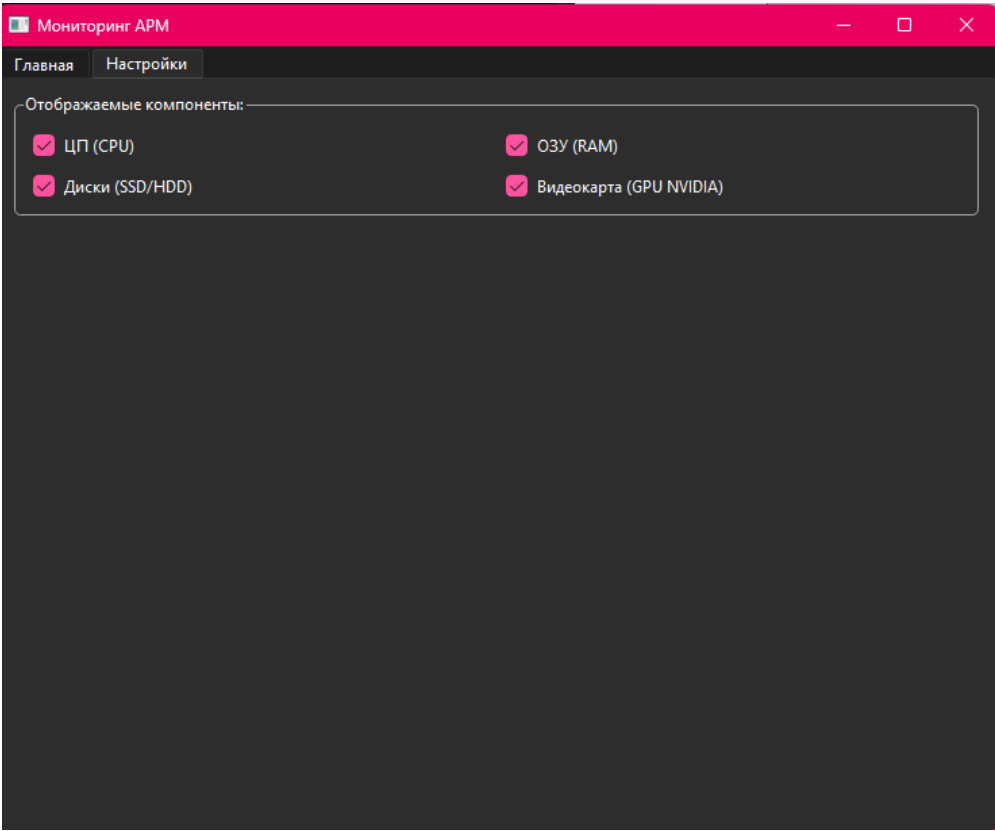


Рисунок 12 – Настройки отображения характеристик

Во вкладке «Настройки», пользователь может настроить то, что хочет видеть на главной странице приложения (рис. 14).

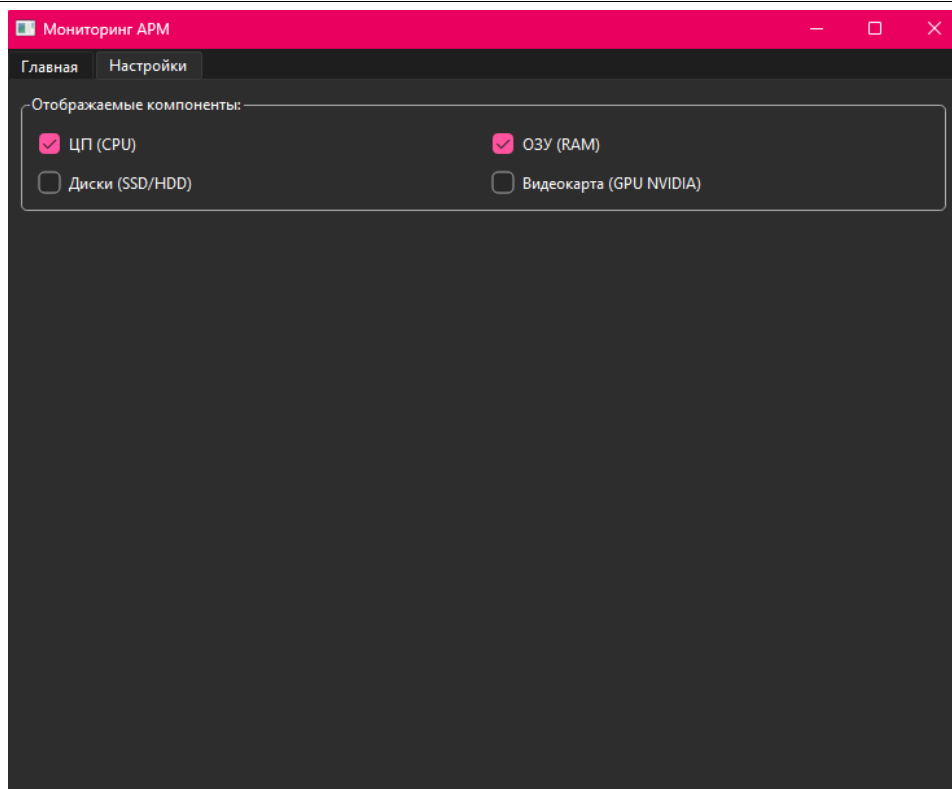


Рисунок 13 – Выбор необходимых характеристик

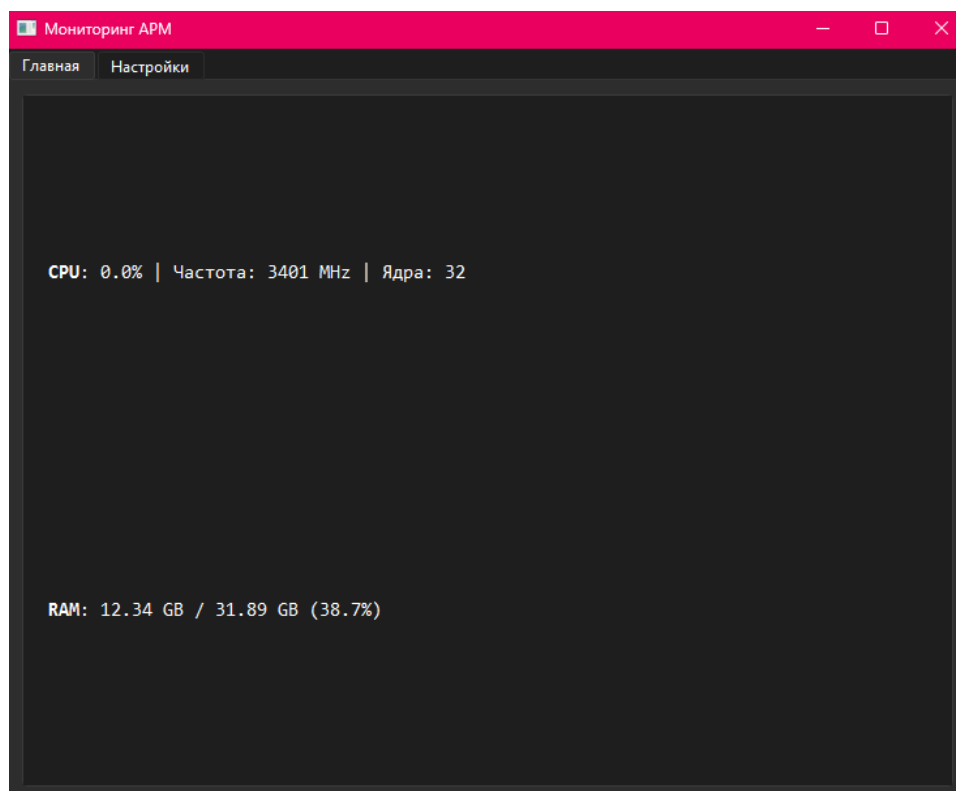


Рисунок 14 – Отображение необходимых характеристик

4 Выводы

В результате проведённой работы было разработано кроссплатформенное приложение для мониторинга системных ресурсов с поддержкой видеокарт NVIDIA. Использование библиотеки PyQt6 позволило

создать интуитивно понятный графический интерфейс, а интеграция с модулем `runvml` обеспечила возможность получения детальной информации о состоянии комплектующих АРМ.

Приложение может быть использовано для оперативной диагностики производительности компьютера, в качестве учебного примера разработки GUI-приложений на Python, а также как основа для построения более сложных систем мониторинга сетевых или серверных ресурсов.

Библиографический список

1. Машков В.А., Литвиненко В.И. Использование языка программирования python 3 и системы компьютерной алгебры sympy на факультативных занятиях по теории чисел // Электротехнические и компьютерные системы. 2017. С. 48-54. URL: <https://elibrary.ru/item.asp?id=25552474> (дата обращения: 12.01.2026).
2. Бухарова Г.Д., Комельских П.С. Важность и необходимость внедрения языка программирования python в процесс обучения студентов // Новые информационные технологии в образовании Материалы VII международной научно-практической конференции. 2014. С. 40-42. URL: <https://elar.uspu.ru/handle/ru-uspu/5?mode=full> (дата обращения: 12.01.2026).
3. Сейдаметов Г.С. Особенности использования языка программирования python в подготовке будущих инженеров-программистов // International scientific review. 2019. №. 9(10). С. 50-51. URL: <https://cyberleninka.ru/article/n/osobennosti-ispolzovaniya-yazyka-programmirovaniya-python-v-podgotovke-buduschih-inzhenerov-programmistov> (дата обращения: 12.01.2026).
4. Усеинов Э.А. Объектно-ориентированное программирование в рамках дисциплины «Язык программирования python» // Ученые записки Крымского инженерно-педагогического университета. 2012. №. 34. С. 157-160. URL: <https://elibrary.ru/item.asp?id=24836776> (дата обращения: 12.01.2026).
5. Матвеева А. С. Реализация паттерна поведения «Команда» в графическом интерфейсе пользователя на Python // Постулат. 2025. №1. URL: <https://e-postulat.ru/index.php/Postulat/article/view/5920/6015> (дата обращения: 12.01.2026).
6. IDE PyCharm URL: <https://www.jetbrains.com/pycharm/>