

Разработка планировщика задач на языке программирования Python

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 локального планировщика задач с графическим интерфейсом на языке программирования Python. Основное внимание уделяется созданию двухвкладочного приложения, включающего календарь для ввода задач по датам и список задач с возможностью их завершения и удаления. Реализованы функции добавления заметок к конкретным датам, визуального зачёркивания выполненных задач с помощью чекбоксов, а также удаления всех завершённых задач по кнопке. Пользовательский интерфейс разработан с использованием библиотеки PyQt6, обеспечивая поддержку как тёмной, так и светлой тем оформления с возможностью переключения в реальном времени. Данные хранятся локально в базе SQLite, что исключает зависимость от интернета или внешних серверов. Статья демонстрирует интеграцию графического интерфейса, управления состоянием задач и работы с локальной базой данных, что делает её полезной для студентов и начинающих разработчиков, изучающих создание автономных настольных приложений на Python.

Ключевые слова: Python, PyQt6, планировщик задач, графический пользовательский интерфейс, SQLite, настольное приложение.

Creating a Task Planner in the Python Programming Language

Belikov Mikhail Gennadievich

Sholom-Aleichem Priamursky State University

Student

Yazykov Kirill Alexandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article examines the development of a local task planner with a graphical user interface using the Python programming language. The focus is on creating a two-tab application featuring a calendar for date-based task entry and a task list with

completion marking and bulk deletion capabilities. Key functionalities include adding notes to specific dates, visually striking through completed tasks via checkboxes, and removing all finished tasks with a single button. The user interface is built using the PyQt6 library, supporting both dark and light themes with real-time switching. Data is stored locally in an SQLite database, ensuring full offline operation without reliance on external servers or internet connectivity. The article demonstrates the integration of GUI design, task state management, and local database handling, making it a valuable resource for students and beginner developers learning to build standalone desktop applications in Python.

Keywords: Python, PyQt6, task planner, graphical user interface, SQLite, desktop application.

1 Введение

1.1 Актуальность

Планировщики задач остаются востребованным инструментом для повышения личной продуктивности, организации учебного и рабочего процесса. В условиях роста нагрузки на студентов и специалистов особенно актуальны простые, автономные и надёжные решения, не требующие регистрации, интернета или хранения данных в облаке. Разработка локального планировщика на Python с использованием PyQt6 позволяет продемонстрировать ключевые принципы построения настольных приложений: работу с графическим интерфейсом, управление состоянием данных, взаимодействие с локальной базой и поддержку пользовательских предпочтений (например, переключение тем оформления). Такой проект служит отличной учебной моделью для освоения современных подходов к созданию интерактивных, функциональных и эстетически выразительных приложений без зависимости от внешних сервисов.

1.2 Обзор исследований

Для ознакомления с языком программирования Python и его применением в разработке графических интерфейсов, рекомендуем обратиться к следующим источникам. 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, надёжности и удобства использования [5].

1.3 Цель исследования

Цель исследования — разработать автономное приложение-планировщик задач с графическим интерфейсом на языке программирования Python, поддерживающее ввод задач через календарь, визуальное отображение выполненных задач с возможностью их массового удаления, а также переключение между светлой и тёмной темами оформления.

2 Материалы и методы

В данном исследовании для разработки приложения использовалась интегрированная среда разработки PyCharm. Язык программирования Python был выбран благодаря своей простоте, высокой читаемости и богатой экосистеме библиотек для создания графических приложений. Для построения пользовательского интерфейса применена библиотека PyQt6 — современный инструмент для разработки кроссплатформенных настольных приложений с поддержкой сложных виджетов, событийной модели и кастомизации внешнего вида. Для хранения данных локально использована встроенная СУБД SQLite, не требующая отдельной установки и обеспечивающая надёжное хранение задач с минимальными накладными расходами.

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).

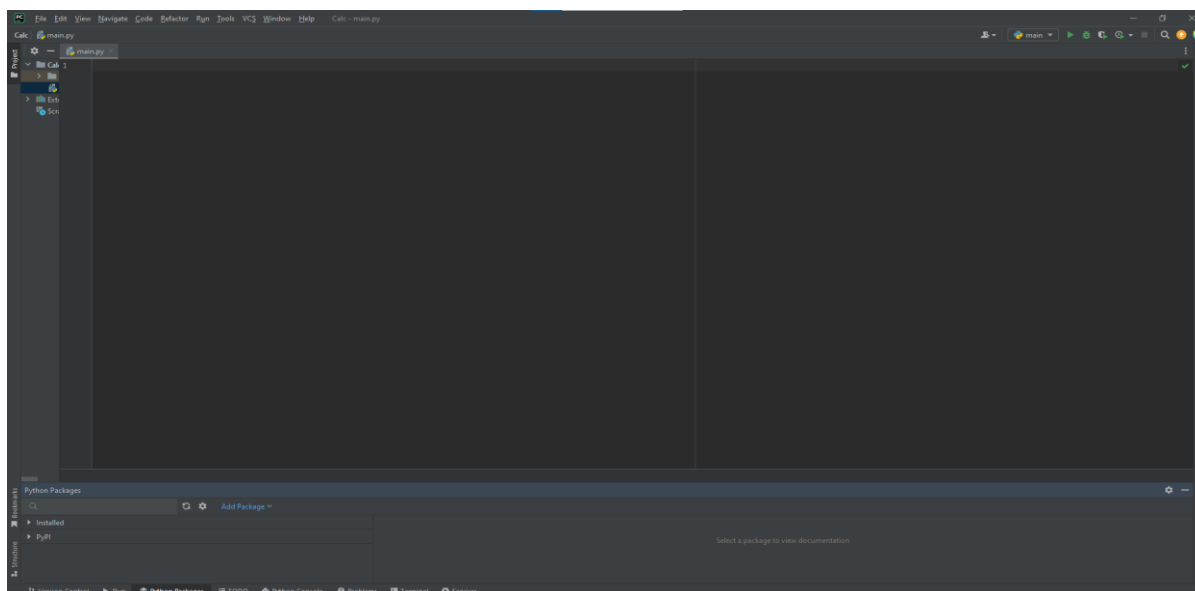


Рисунок 1 – Интерфейс IDE PyCharm

После входа, необходимо установить библиотеку datetime, PyQt6 для этого переходим в терминал и вводим `pip install datetime` и `pip install PyQt6`, или установить через Python Packages (рис. 2).

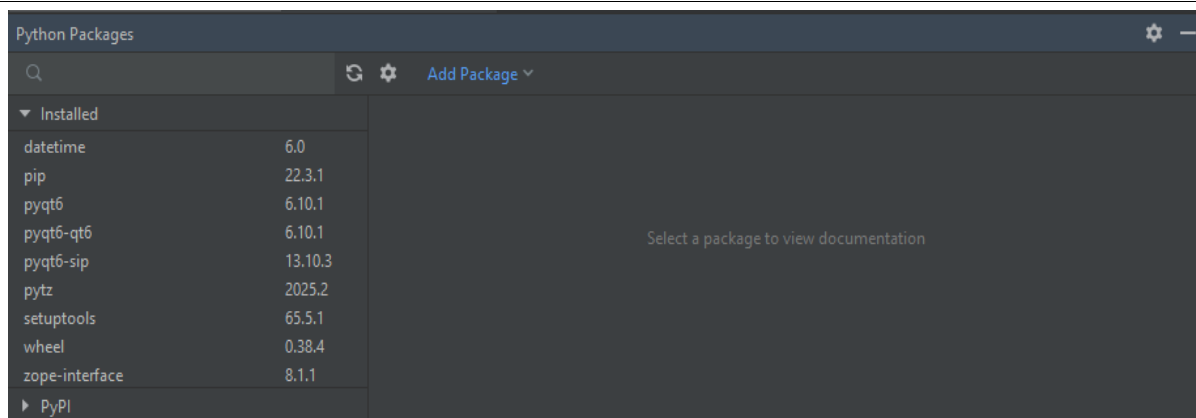


Рисунок 2 – Установка библиотек datetime и PyQt6 в Python Packages

Перед работой с графической оболочкой следует создать файл базы данных, в котором будут храниться записи пользователей под названием database.py (рис. 3).

```

1  import sqlite3
2  from pathlib import Path
3
4  DB_PATH = Path.home() / "Documents" / "TaskPlanner" / "tasks.db"
5
6  2 usages
7  def init_database():
8      DB_PATH.parent.mkdir(parents=True, exist_ok=True)
9      conn = sqlite3.connect(DB_PATH)
10     conn.execute("""
11         CREATE TABLE IF NOT EXISTS tasks (
12             id INTEGER PRIMARY KEY AUTOINCREMENT,
13             date TEXT NOT NULL,
14             text TEXT NOT NULL,
15             completed INTEGER NOT NULL DEFAULT 0
16         )
17     """)
18     conn.commit()
19     conn.close()
20
21  2 usages
22  def add_task(date_str: str, text: str):
23     if not text.strip():
24         return
25     conn = sqlite3.connect(DB_PATH)
26     conn.execute("INSERT INTO tasks (date, text, completed) VALUES (?, ?, 0)", (date_str, text.strip()))
27     conn.commit()
28     conn.close()
29
30  2 usages
31  def get_all_tasks():
32     conn = sqlite3.connect(DB_PATH)
33     cur = conn.execute("SELECT id, date, text, completed FROM tasks ORDER BY date DESC, id DESC")
34     rows = cur.fetchall()
35     conn.close()
36     return rows
37
38  2 usages
39  def toggle_task_completed(task_id: int, completed: int):
40     conn = sqlite3.connect(DB_PATH)
41     conn.execute("UPDATE tasks SET completed = ? WHERE id = ?", (completed, task_id))
42     conn.commit()
43     conn.close()
44
45  2 usages
46  def delete_completed_tasks():
47     conn = sqlite3.connect(DB_PATH)
48     conn.execute("DELETE FROM tasks WHERE completed = 1")
49     conn.commit()
50     conn.close()

```

Рисунок 3 – Код файла database.py

Для написания коды базы данных потребуется библиотека `sqlite3` и `pathlib`, установить можно также через `Python Packages`. `DB_PATH = Path.home()` возвращает домашнюю директорию пользователя и формируется путь к сохранению файла. `init_database()` инициализирует базу данных для записи и чтения. `CREATE TABLE IF NOT EXISTS` проверяет наличие таблицы. `add_task(date_str: str, text: str)` – добавляет новые задачи в базу данных. `get_all_tasks()` – функция для отображения всех задач.

`toggle_task_completed(task_id: int, completed: int)` – функция, позволяющая изменять статус задач. `delete_completed_tasks()` – функция, которая удаляет все выполненные задачи.

Следующим шагом является написание главного приложения `main.py` (рис. 4-7).

```

1  import sys
2      from datetime import datetime
3      from PyQt6.QtWidgets import (
4          QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
5          QWidget, QCalendarWidget, QLineEdit, QPushButton,
6          QListWidget, QListWidgetItem, QLabel, QMessageBox, QCheckBox
7      )
8      from PyQt6.QtCore import Qt, QSettings
9      from PyQt6.QtGui import QFont, QPalette
10     from database import (
11         init_database, add_task, get_all_tasks,
12         toggle_task_completed, delete_completed_tasks
13     )
14     from themes import apply_theme
15
16     1 usage
17     class TaskItemWidget(QWidget):
18     def __init__(self, task_id: int, date_str: str, text: str, completed: int, parent_list, is_dark: bool):
19         super().__init__()
20         self.task_id = task_id
21         self.completed = bool(completed)
22         self.parent_list = parent_list
23         self.is_dark = is_dark
24
25         try:
26             formatted_date = datetime.strptime(date_str, "%Y-%m-%d").strftime("%d.%m.%Y")
27         except:
28             formatted_date = date_str
29
30         self.text_label = QLabel(f"[{formatted_date}] {text}")
31         self.text_label.setWordWrap(True)
32         self.update_font()
33
34         self.checkbox = QCheckBox()
35         self.checkbox.setChecked(self.completed)
36         self.checkbox.stateChanged.connect(self.on_checkbox_changed)
37         self.checkbox.setFixedSize(30, 30)
38
39         layout = QHBoxLayout()
40         layout.addWidget(self.text_label, 1)
41         layout.addWidget(self.checkbox)
42         layout.setContentsMargins(8, 6, 8, 6)
43         self.setLayout(layout)
44
45     2 usages
46     def update_font(self):
47         font = self.text_label.font()
48         font.setStrikeOut(self.completed)
49         self.text_label.setFont(font)
50         color = "#7a7a7a" if self.completed else ("#e0e0e0" if self.is_dark else "#000000")
51         self.text_label.setStyleSheet(f"font-size: 11pt; color: {color};")
52
53     1 usage
54     def on_checkbox_changed(self, state):
55         self.completed = (state == Qt.CheckState.Checked.value)
56         toggle_task_completed(self.task_id, int(self.completed))
57         self.update_font()
58         self.parent_list.update_tasks()

```

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

```

59 class CustomListWidget(QListWidget):
60     def __init__(self):
61         super().__init__()
62         self.setHorizontalScrollBarPolicy(Qt.ScrollBarPolicy.ScrollBarAlwaysOff)
63         self.is_dark = True
64
65     def update_tasks(self):
66         load_tasks_into_list(self, self.is_dark)
67
68
69 def load_tasks_into_list(list_widget: CustomListWidget, is_dark: bool):
70     list_widget.clear()
71     list_widget.is_dark = is_dark
72     tasks = get_all_tasks()
73     for task_id, date_str, text, completed in tasks:
74         item_widget = TaskItemWidget(task_id, date_str, text, completed, list_widget, is_dark)
75         item = QListWidgetItem(list_widget)
76         item.setSizeHint(item_widget.sizeHint())
77         list_widget.addItem(item)
78         list_widget.setItemWidget(item, item_widget)
79
80
81 def on_delete_completed(list_widget):
82     reply = QMessageBox.question(
83         None, "Подтверждение",
84         "Удалить все выполненные задачи?",
85         QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
86     )
87     if reply == QMessageBox.StandardButton.Yes:
88         delete_completed_tasks()
89         list_widget.update_tasks()
90         QMessageBox.information(None, "Готово", "Выполненные задачи удалены.")
91
92
93 class TaskPlanner(QMainWindow):
94     def __init__(self):
95         super().__init__()
96         self.setWindowTitle("Планировщик задач")
97         self.resize(850, 650)
98
99         init_database()
100
101         self.settings = QSettings("TaskPlanner", "Theme")
102         self.is_dark = self.settings.value("dark_mode", True, type=bool)
103
104         self.theme_info = apply_theme(QApplication.instance(), self.is_dark)
105
106         central = QWidget()
107         self.setCentralWidget(central)
108         main_layout = QVBoxLayout(central)
109
110         self.theme_button = QPushButton("Светлая тема" if self.is_dark else "Тёмная тема")
111         self.theme_button.clicked.connect(self.toggle_theme)
112         self.theme_button.setFixedWidth(150)

```

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

```

112         self.theme_button.setFixedWidth(150)
113         top_layout = QHBoxLayout()
114         top_layout.addStretch()
115         top_layout.addWidget(self.theme_button)
116         main_layout.addLayout(top_layout)
117
118         self.tabs = QTabWidget()
119         main_layout.addWidget(self.tabs)
120
121         self.calendar_tab = self.create_calendar_tab()
122         self.tabs.addTab(self.calendar_tab, "Календарь")
123
124         self.tasks_list = CustomListWidget()
125         self.tasks_tab = self.create_tasks_tab()
126         self.tabs.addTab(self.tasks_tab, "Задачи")
127
128         load_tasks_into_list(self.tasks_list, self.is_dark)
129         self.selected_date = None
130
131     1 usage
132     def create_calendar_tab(self):
133         widget = QWidget()
134         layout = QVBoxLayout()
135
136         self.calendar = QCalendarWidget()
137         self.calendar.setGridVisible(True)
138         self.calendar.clicked.connect(self.on_calendar_date_selected)
139         layout.addWidget(self.calendar)
140
141         input_layout = QHBoxLayout()
142         self.task_input = QLineEdit()
143         self.task_input.setPlaceholderText("Введите задачу...")
144         self.update_placeholder_color()
145         self.add_button = QPushButton("Добавить")
146         self.add_button.clicked.connect(self.on_add_task)
147         input_layout.addWidget(self.task_input)
148         input_layout.addWidget(self.add_button)
149         layout.addLayout(input_layout)
150
151         self.status_label = QLabel("Выберите дату и введите задачу")
152         layout.addWidget(self.status_label)
153
154         widget.setLayout(layout)
155         return widget
156
157     2 usages
158     def update_placeholder_color(self):
159         """Обновляет цвет placeholder в зависимости от темы"""
160         palette = self.task_input.palette()
161         if self.is_dark:
162             palette.setColor(QPalette.ColorRole.PlaceholderText, Qt.GlobalColor.white)
163         else:
164             palette.setColor(QPalette.ColorRole.PlaceholderText, Qt.GlobalColor.gray)
165         self.task_input.setPalette(palette)
166
167     1 usage
168     def create_tasks_tab(self):
169         widget = QWidget()
170         layout = QVBoxLayout()

```

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

```

163     self.task_input.setPalette(palette)
164
165     1 usage
166     def create_tasks_tab(self):
167         widget = QWidget()
168         layout = QVBoxLayout()
169         layout.addWidget(self.tasks_list)
170         delete_btn = QPushButton("Удалить выполненные")
171         delete_btn.clicked.connect(lambda: on_delete_completed(self.tasks_list))
172         layout.addWidget(delete_btn)
173         widget.setLayout(layout)
174         return widget
175
176     1 usage
177     def on_calendar_date_selected(self, qdate):
178         self.selected_date = qdate.toPyDate().isoformat()
179         self.status_label.setText(f"Выбрана дата: {qdate.toString('dd.MM.yyyy')}")
180
181     1 usage
182     def on_add_task(self):
183         if not self.selected_date:
184             self.status_label.setText("Сначала выберите дату в календаре!")
185             return
186         text = self.task_input.text().strip()
187         if not text:
188             self.status_label.setText("Введите текст задачи!")
189             return
190         add_task(self.selected_date, text)
191         self.task_input.clear()
192         self.status_label.setText("Задача добавлена!")
193         self.tasks_list.update_tasks()
194
195     1 usage
196     def toggle_theme(self):
197         self.is_dark = not self.is_dark
198         self.settings.setValue("dark_mode", self.is_dark)
199         self.theme_info = apply_theme(QApplication.instance(), self.is_dark)
200         self.theme_button.setText("Светлая тема" if self.is_dark else "Темная тема")
201         self.update_placeholder_color()
202         self.tasks_list.update_tasks()
203
204     1 usage
205     def main():
206         app = QApplication(sys.argv)
207         app.setStyle("Fusion")
208
209         window = TaskPlanner()
210         window.show()
211         sys.exit(app.exec())
212
213     if __name__ == "__main__":
214         main()

```

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

В данном коде class TaskItemWidget(QWidget) отображает задачи во вкладке «Задачи» в виде виджета, с использованием форматирования даты и стилей. CustomListWidget создается для предотвращения ошибок с горизонтальным скроллом, а также сохранением темы из предыдущей вкладки. load_tasks_into_list загружает все задачи из базы данных с определенными стилями для правильной визуализации пользователю.

Функция `on_delete_completed` позволяет удалять все выполненные задачи после подтверждения.

Чтобы графический интерфейс выглядел более симпатичней, следует создать файл стилей `themes.py`, в котором будет переключение между темами (светлой и тёмной). Данный файл позволит пользователю настроить интерфейс программы под свой вкус (рис. 8).

```

1  from PyQt6.QtGui import QColor, QPalette
2  from PyQt6.QtCore import Qt
3
4  def apply_theme(app, is_dark: bool):
5      app.setStyle("Fusion")
6      palette = QPalette()
7
8      if is_dark:
9          bg = QColor(30, 30, 30)
10         bg_light = QColor(37, 37, 38)
11         bg_widget = QColor(45, 45, 48)
12         text = QColor(224, 224, 224)
13         text_disabled = QColor(120, 120, 120)
14         border = QColor(60, 60, 60)
15         selection = QColor(13, 110, 253)
16         palette.setColor(QPalette.ColorRole.Window, bg)
17         palette.setColor(QPalette.ColorRole.WindowText, text)
18         palette.setColor(QPalette.ColorRole.Base, bg)
19         palette.setColor(QPalette.ColorRole.AlternateBase, bg_light)
20         palette.setColor(QPalette.ColorRole.ToolTipBase, text)
21         palette.setColor(QPalette.ColorRole.ToolTipText, text)
22         palette.setColor(QPalette.ColorRole.Text, text)
23         palette.setColor(QPalette.ColorRole.Button, bg_widget)
24         palette.setColor(QPalette.ColorRole.ButtonText, text)
25         palette.setColor(QPalette.ColorRole.BrightText, Qt.GlobalColor.red)
26         palette.setColor(QPalette.ColorRole.Link, selection)
27         palette.setColor(QPalette.ColorRole.Highlight, selection)
28         palette.setColor(QPalette.ColorRole.HighlightedText, Qt.GlobalColor.white)
29         palette.setColor(QPalette.ColorGroup.Disabled, QPalette.ColorRole.Text, text_disabled)
30         palette.setColor(QPalette.ColorGroup.Disabled, QPalette.ColorRole.WindowText, text_disabled)
31     else:
32         bg = QColor(255, 255, 255)
33         bg_light = QColor(240, 240, 240)
34         bg_widget = QColor(230, 230, 230)
35         text = QColor(0, 0, 0)
36         text_disabled = QColor(160, 160, 160)
37         border = QColor(200, 200, 200)
38         selection = QColor(13, 110, 253)
39         palette.setColor(QPalette.ColorRole.Window, bg)
40         palette.setColor(QPalette.ColorRole.WindowText, text)
41         palette.setColor(QPalette.ColorRole.Base, Qt.GlobalColor.white)
42         palette.setColor(QPalette.ColorRole.AlternateBase, bg)
43         palette.setColor(QPalette.ColorRole.ToolTipBase, Qt.GlobalColor.white)
44         palette.setColor(QPalette.ColorRole.ToolTipText, text)
45         palette.setColor(QPalette.ColorRole.Text, text)
46         palette.setColor(QPalette.ColorRole.Button, bg_widget)
47         palette.setColor(QPalette.ColorRole.ButtonText, text)
48         palette.setColor(QPalette.ColorRole.BrightText, Qt.GlobalColor.red)
49         palette.setColor(QPalette.ColorRole.Link, selection)
50         palette.setColor(QPalette.ColorRole.Highlight, selection)
51         palette.setColor(QPalette.ColorRole.HighlightedText, Qt.GlobalColor.white)
52         palette.setColor(QPalette.ColorGroup.Disabled, QPalette.ColorRole.Text, text_disabled)
53         palette.setColor(QPalette.ColorGroup.Disabled, QPalette.ColorRole.WindowText, text_disabled)
54     app.setPalette(palette)
55     return {
56         "is_dark": is_dark, "bg": bg, "bg_widget": bg_widget, "text": text, "text_disabled": text_disabled, "border": border, "selection": selection
57     }

```

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 "TaskPlanner" main.py` (рис. 10).

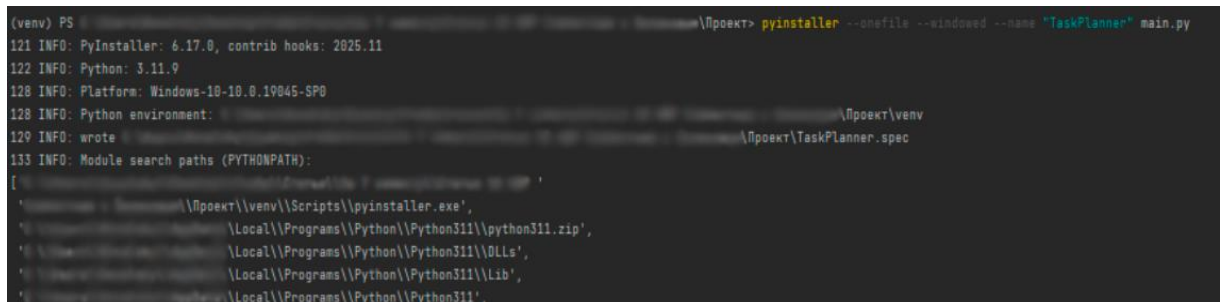


Рисунок 10 – Упаковка приложения

Приложение запаковано и находится в папке `dist` (рис. 11).

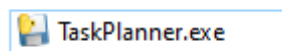


Рисунок 11 – Локальное приложение

При запуске приложения появляется светлая тема, которую можно сменить на темную, нажав на кнопку «Тёмная тема» (рис. 12-13).

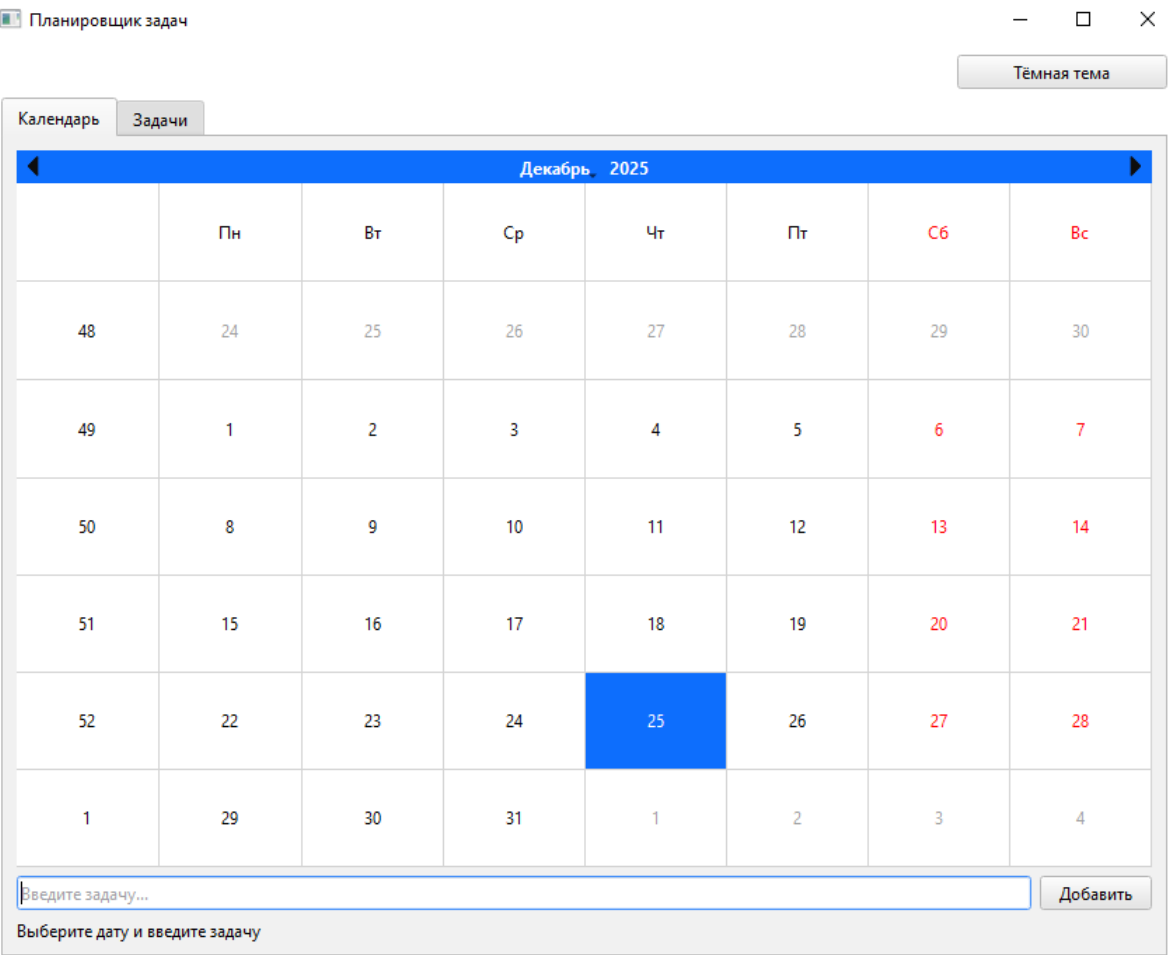


Рисунок 12 – Светлая тема приложения

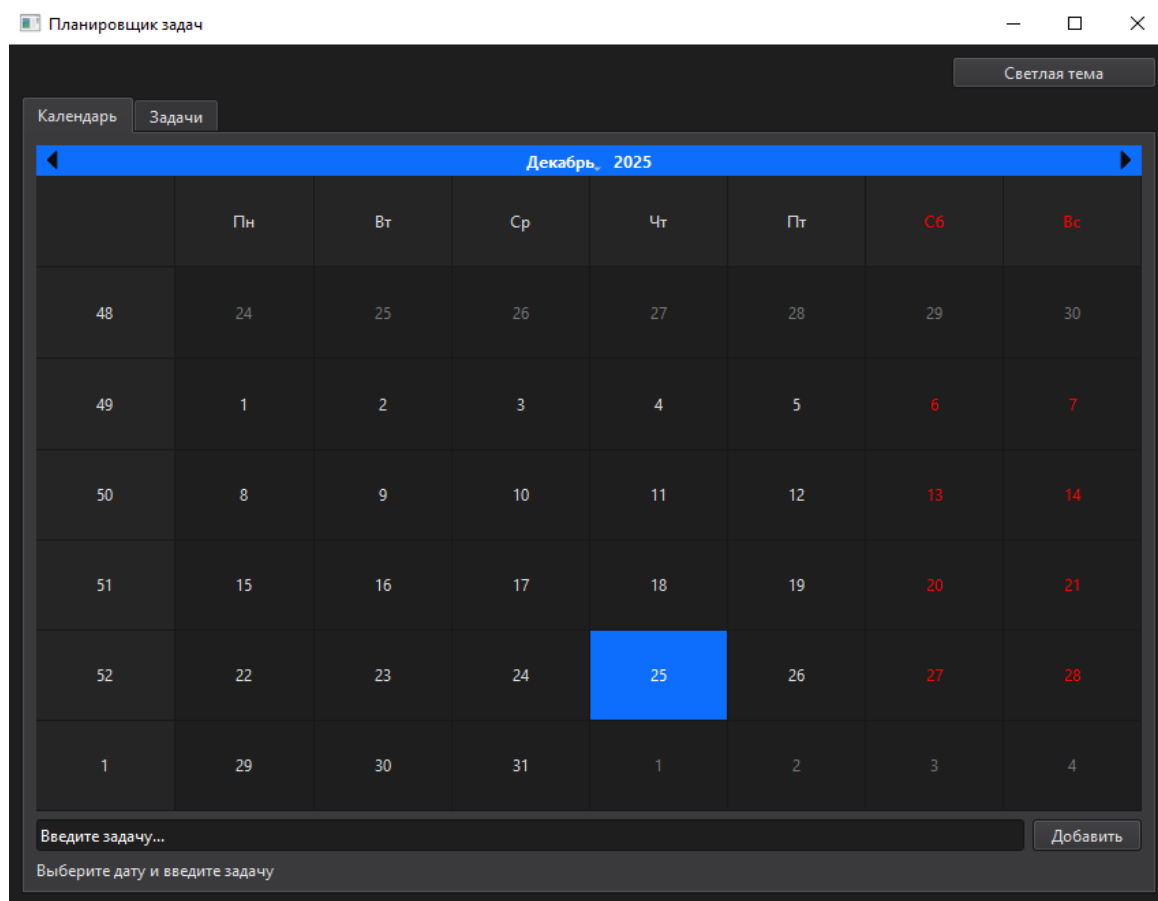


Рисунок 13 – Темная тема приложения

В левом нижнем углу есть подсказка, которая просит пользователя выбрать дату на календаре, а выше нее в поле ввода пользователь может вводить задачу, которую необходимо выполнить (рис. 14).

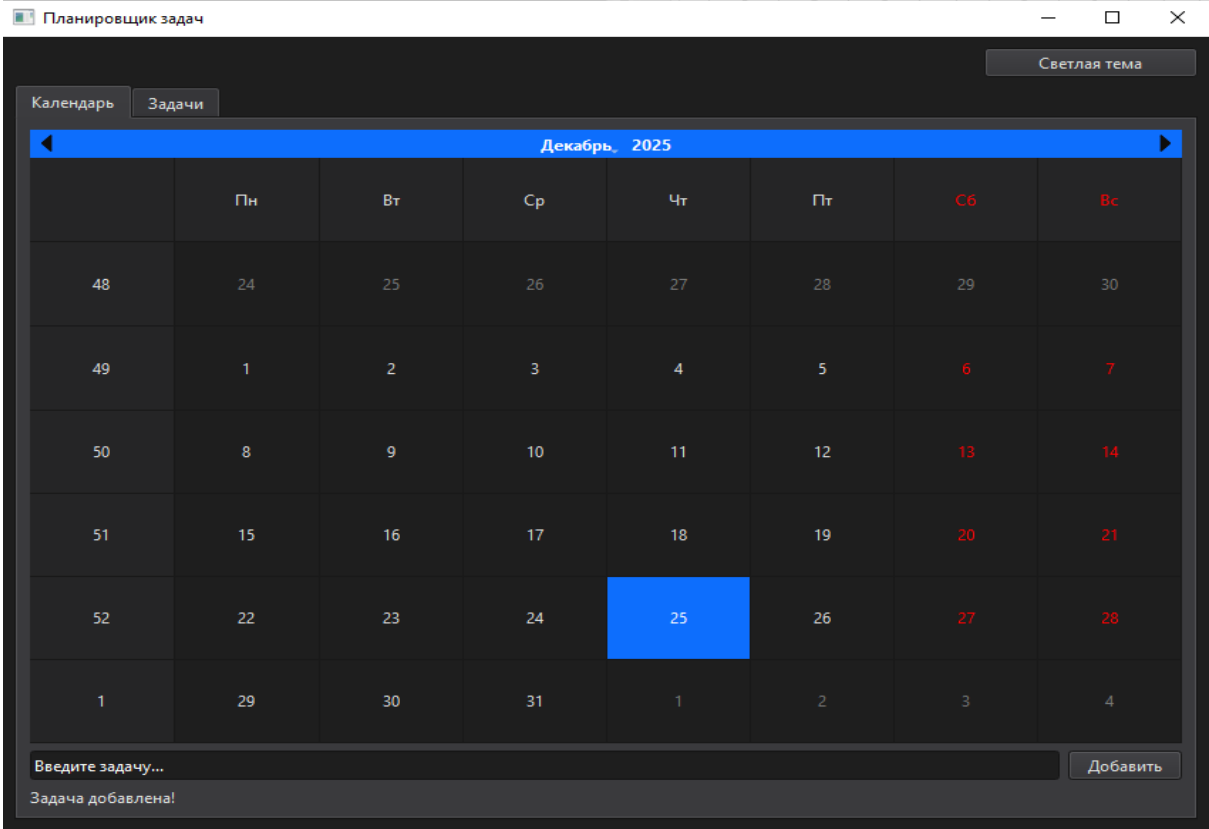


Рисунок 14 – Задача добавлена

Чтобы увидеть все задачи, необходимо перейти во вкладку «Задачи» (рис. 15).

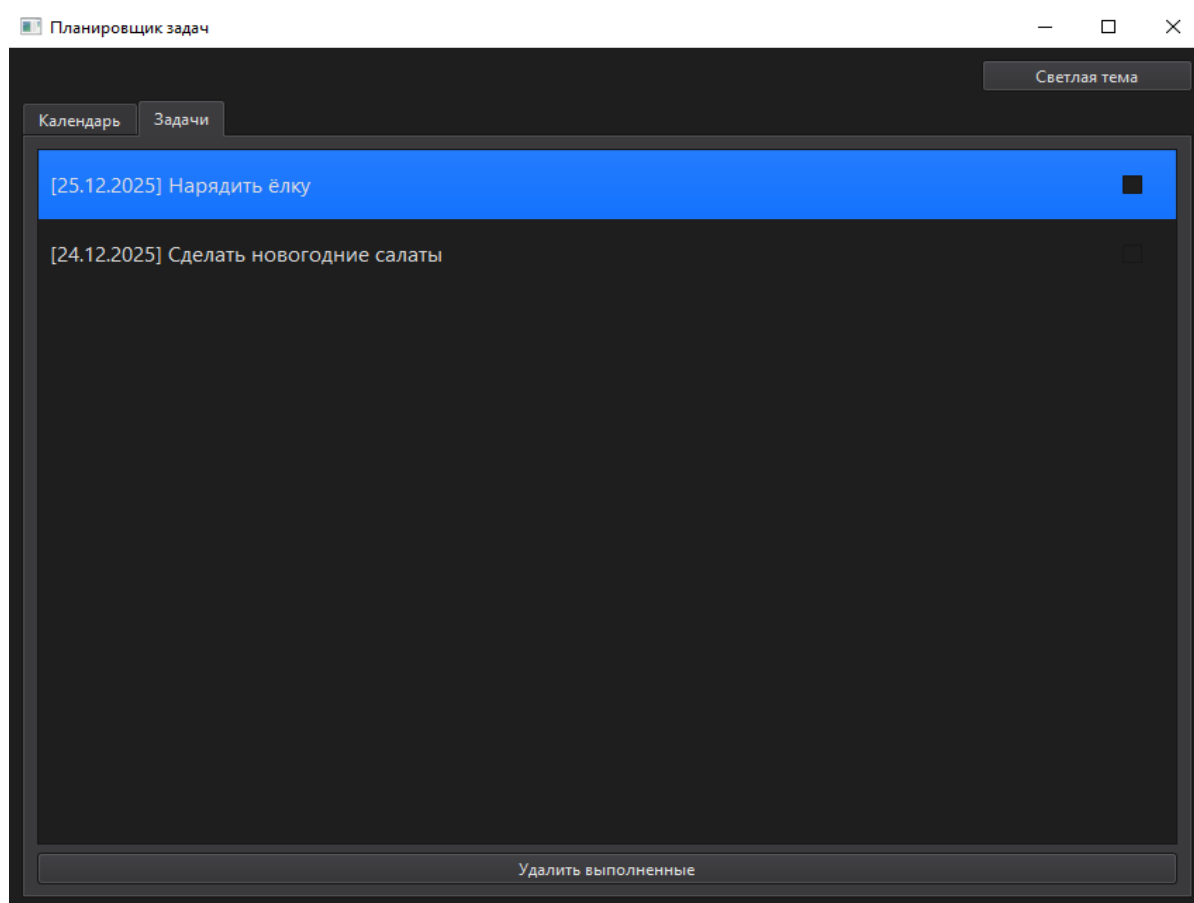


Рисунок 15 – Вкладка «Задачи»

Справа от названия задачи, есть квадратное поле, при нажатии на которое в нем появляется галочка, а текст задачи вычеркивается (рис. 16).

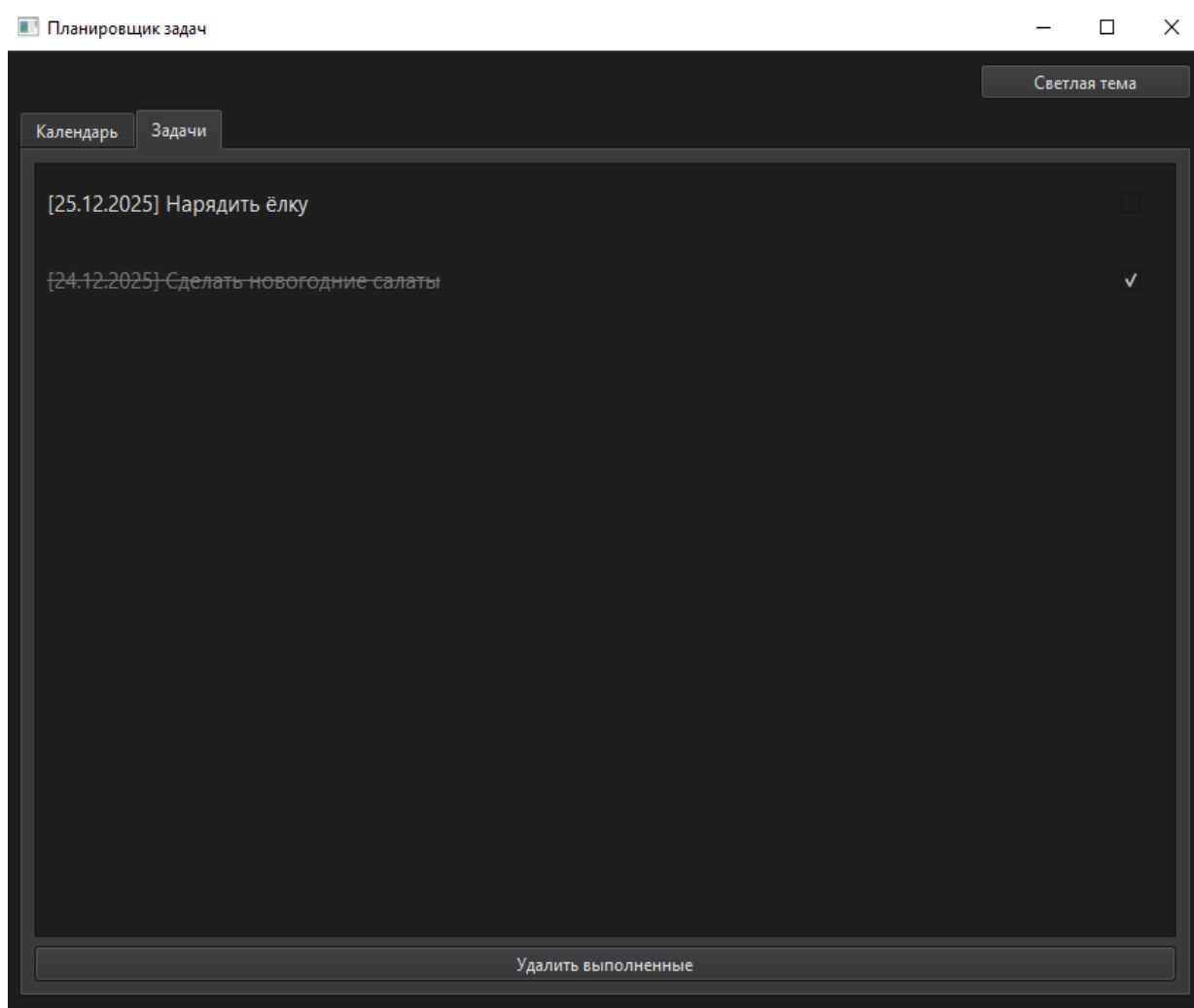


Рисунок 16 – Выполнение задачи

Когда будет большой список выполненных задач, можно удалить все выполненные по кнопке «Удалить выполненные» (рис. 17).

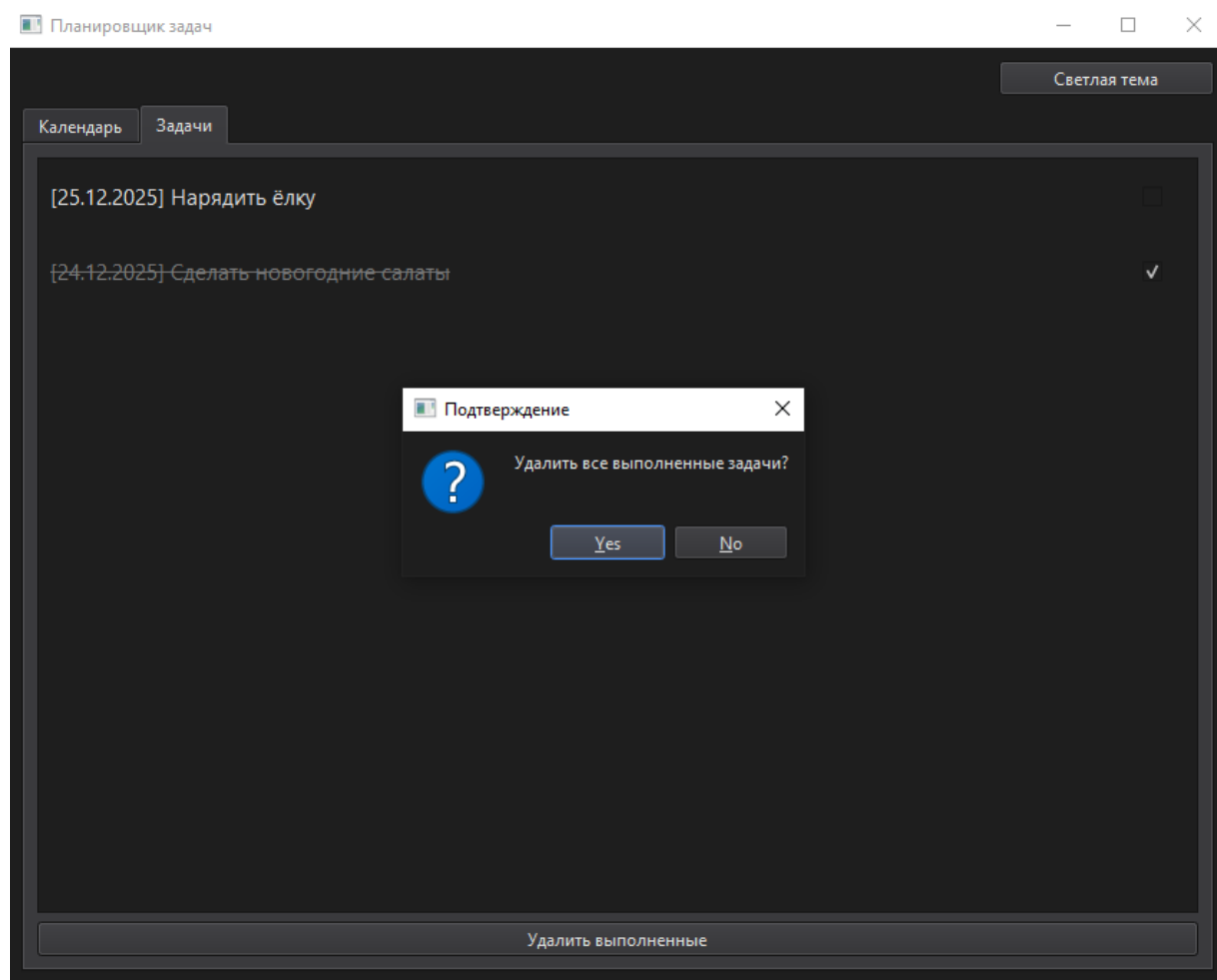


Рисунок 17 – Удаление всех выполненных задач

После данной операции остаются только задачи, которые не были выполнены (рис. 18).

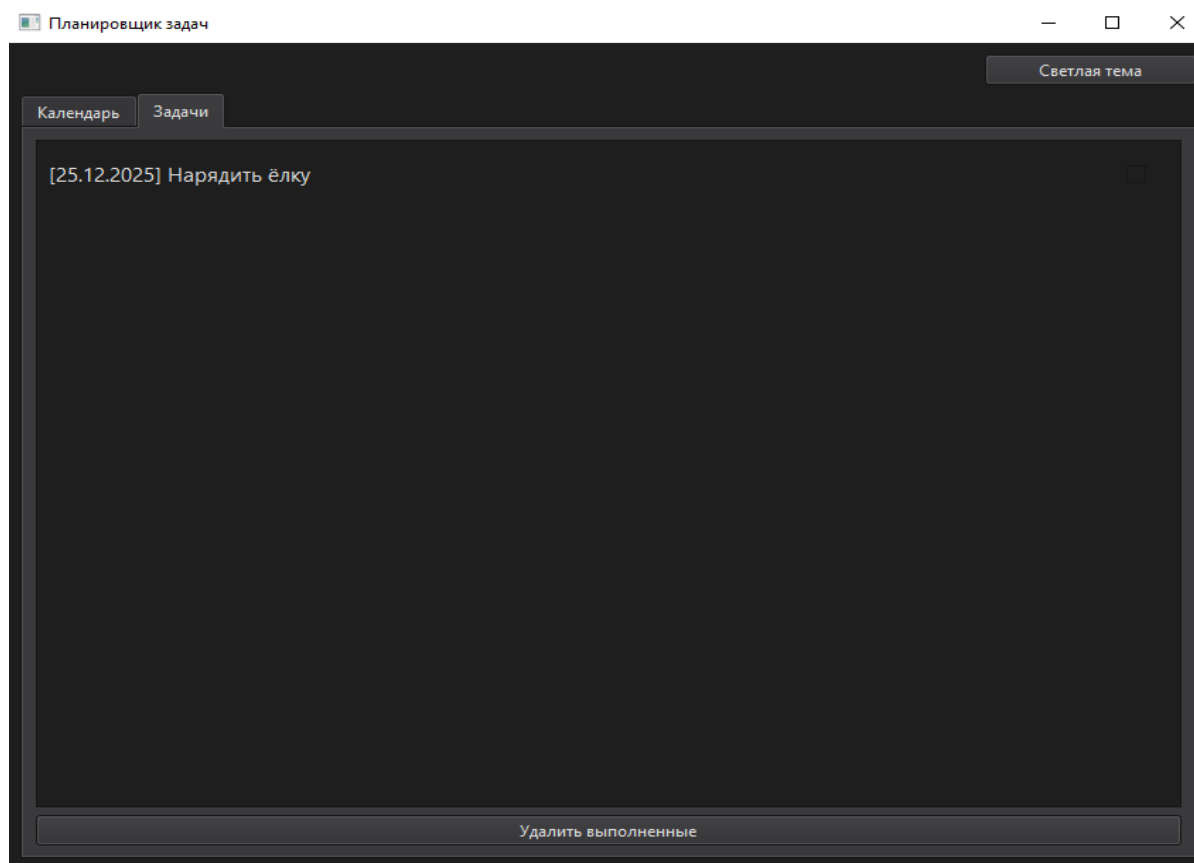


Рисунок 18 – Список только не выполненных задач

4 Выводы

В результате проведённого исследования разработан функциональный планировщик задач на языке Python с использованием библиотеки PyQt6. Приложение обеспечивает удобное взаимодействие пользователя с календарём, позволяет добавлять задачи на конкретные даты, визуально отмечать их выполнение с помощью чекбоксов и управлять списком задач через интуитивно понятный двухвкладочный интерфейс.

Реализованы ключевые функции: зачёркивание завершённых задач, массовое удаление выполненных элементов, а также переключение между светлой и тёмной темами с сохранением настроек между запусками. Все данные хранятся локально в базе SQLite, что гарантирует автономность и безопасность без передачи информации во внешние сети.

Создание подобного приложения даёт начинающим разработчикам ценный практический опыт проектирования графического интерфейса, управления состоянием данных и интеграции логики приложения с возможностями современных GUI-библиотек.

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

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.
3. Сейдаметов Г.С. Особенности использования языка программирования python в подготовке будущих инженеров-программистов // International scientific review. 2019. №. 9(10). С. 50-51
4. Усеинов Э.А. Объектно-ориентированное программирование в рамках дисциплины «Язык программирования python» // Ученые записки крымского инженерно-педагогического университета. 2012. №. 34. С. 157-160.
5. Матвеева А. С. Реализация паттерна поведения «Команда» в графическом интерфейсе пользователя на Python // Постулат. 2025. №1.