

Колледж космического машиностроения и технологий

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка приложения «Информер погоды» на Python»

Выполнил студент:

Васькевич Ю.А

Группа П2-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Проверил преподаватель:

Гусятинер Леонид Борисович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королев 2021 г.

**Оглавление**

[**Введение** 3](#_Toc74744286)

[**1.** **Теоретическая часть** 4](#_Toc74744287)

[**1.1.** **Описание предметной области** 4](#_Toc74744288)

[**1.2.** **Описание существующих разработок** 5](#_Toc74744289)

[**1.2.1.** **Yandex.** 5](#_Toc74744290)

[**1.2.2.** **Gismeteo** 6](#_Toc74744291)

[**2.** **Проектная часть** 7](#_Toc74744292)

[**2.1.** **Диаграмма прецедентов** 7](#_Toc74744293)

[**2.2.** **Выбор инструментов** 8](#_Toc74744294)

[**2.3.** **Проектирование сценария** 9](#_Toc74744295)

[**2.4.** **Описание главного модуля** 10](#_Toc74744296)

[**2.5.** **Описание спецификаций к модулям** 23](#_Toc74744297)

[**2.6.** **Описание модулей** 25](#_Toc74744298)

[**2.7.** **Описание тестовых наборов модулей** 26](#_Toc74744299)

[**Описание применения средств отладки** 32](#_Toc74744300)

[**3.** **Эксплуатационная часть** 34](#_Toc74744301)

[**3.1.** **Руководство пользователя** 34](#_Toc74744302)

[**3.2** **Элементы управления** 38](#_Toc74744303)

[**Заключение** 39](#_Toc74744304)

[**Список литературы и интернет-источников** 40](#_Toc74744305)

# **Введение**

Целью данного курсового проекта является написание программы «Информер погоды» для заранее узнавание погоды в вашей городе или в ином.

Эта тема является актуальной на данный момент, потому что в наше время погода может помешать нашим планам или поставить в неудобное положение. Данный курсовой проект позволит путём нахождения по поиску нужного вашего города, и узнать какая будет в нём погода на неделю, так же можно узнать дополнительную информацию. Так же в данном проекте будет простой для понимания интерфейс и небольшой порог вхождения.

В первой части будет рассмотрена предметная область данной темы, а также несколько продуктов по данной теме.

Во второй части будут рассмотрены инструменты и модули, которые были разработаны, структура программной части и листинги ключевых частей программных модулей.

В третьей части будет рассмотрено руководство для пользователей.

В заключительной части будет приведен To-do лист с планами по доработки программы, а также сделаны общие выводы о получившемся проекте.

# **Теоретическая часть**

## **Описание предметной области**

Каждому человеку необходимо знать прогноз погоды, для того чтобы планировать свои дела и даже подбирать гардероб на определенный день недели.

Для того чтобы узнать достоверный прогноз погоды раньше все смотрели новости, дожидаясь специального выпуска. Теперь все стало гораздо проще, ведь прогноз можно в любое время изучить.

В данных приложениях(сайтах) можно ознакомиться с прогнозом погоды практически для всех российских и украинских городов, а также узнать о том, как обстоят дела с погодой на популярных курортах мира.

Можно изучить прогноз погоды на ближайшие сутки, а также на несколько дней вперед. На сайте «Погодник» можно узнать не только прогноз погоды, но и основную информацию об определенном населенном пункте, например, численность его населения.

**Прогноз погоды** — научно обоснованное предположение о будущем состоянии погоды в определённом пункте или регионе на определённый период. Составляется (разрабатывается) метеорологическими службами на основе методов метеорологии.

**Среднесрочный прогноз погоды**

Прогнозы на средние сроки (4—10 дней) опираются на текущую информацию о развитии синоптических процессов в течение всего сезона, а также на будущие поля давления (геопотенциала) и температуры на 1—5 суток вперед, рассчитанные с помощью интегрирования уравнений динамики и термодинамики атмосферы, с использованием в качестве начальных данных фактических наблюдений в момент времени, исходный для начала интегрирования. Однако в прогнозах на 5—10 дней уже нет возможности прогнозировать изменения погоды по дням, так как невозможно проследить за изменением свойств воздушных масс на протяжении всего прогнозируемого периода. Поэтому прогнозируются средняя температура (либо аномалия температуры) и среднее количество осадков на предстоящие 10 дней, а также величины максимальной и минимальной температуры в течение будущих 5—7 суток.

## **Описание существующих разработок**

В этом разделе рассмотрены некоторые уже имеющиеся программы для узнавания погоды, и также дополнительной информации населенного пункта.

Интерфейс таких программ и сайтов для узнавания информации о погоде чаще всего простой и понятный для обычных пользователей.

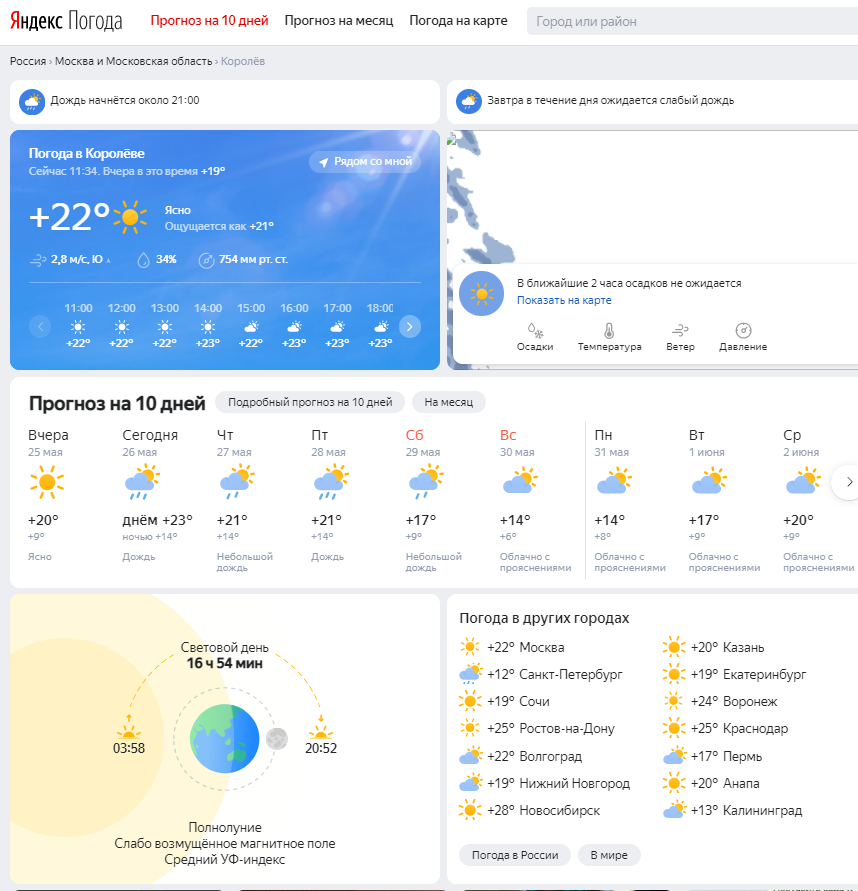
### **Yandex.**

Полное название – прогноз погоды под названием “Яндекс Погода” от одноименной компании “Yandex”

Один из самых популярных прогнозов погоды, в интернете есть множество положительных отзывов о данном инструменте.

Достоинства, которые подтверждают отзывы:

1. Легкодоступная для обычного пользователя.
2. Приятный интерфейс и дизайн, который хорошо гармонирует между собой.
3. Соответствует самым требовательным запросам.
4. Гарантирует достоверную информацию которую предоставляет пользователю.
5. Имеет множество дополнительный информации о погоде.

 Рисунок 4. Интерфейс Yandex.

### **Gismeteo**

Гисметео(Gismeteo.ru) — один из самых популярных российских сайтов о погоде. Сайт был создан 12 декабря 2000 года, однако прогнозы погоды публиковались с середины 1990-х на BBS и в сети FIDONet.

Особенности:

1. Сайт позволяет узнать о погоде в различных городах России и других стран мира. Кроме прогноза температуры (на сегодня, завтра, 3, 10, 14 дней и месяц), сайт содержит информацию об осадках, направлении ветра, давлении, влажности, видимости, продолжительности дня и магнитных бурях. Также имеется сервис радара осадков и гроз.
2. Заслужил доверие пользователей за время своего существования.
3. Имеет минималистический дизайн приятный глазу.

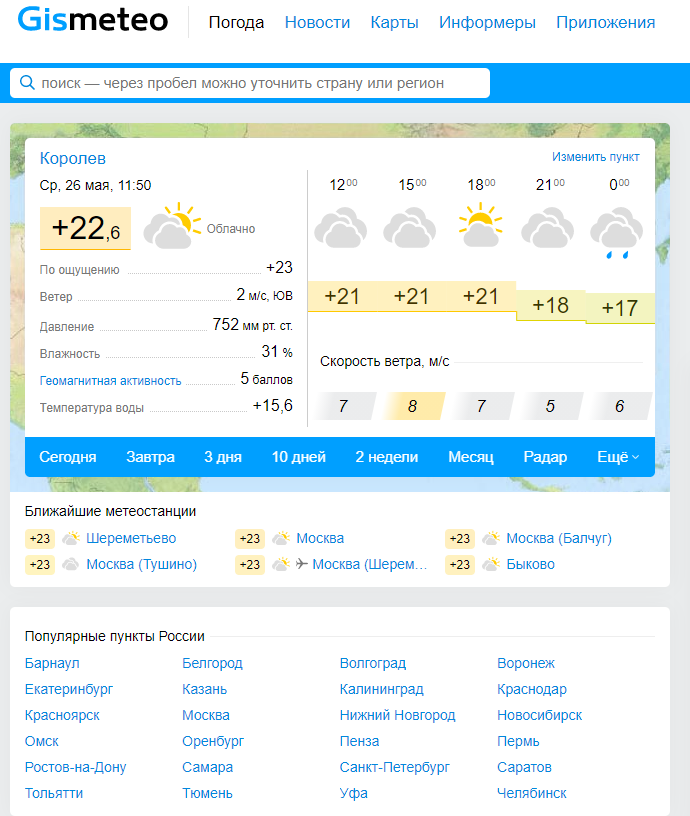


Рисунок 5. Дизайн Gismeteo

# **Проектная часть**

## **Диаграмма прецедентов**

В этом разделе представлена одна диаграммы прецедентов. На ней показана вся возможная функциональная и поведенческое отношение.



Рисунок 6. Диаграмма прецедентов работы в программе

## **Выбор инструментов**

При выборе инструментов было проведено сравнение по критериям, представленных в таблице 1.

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1.

Критерии выбора инструмента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Участие в корпоративном проекте | Простота сопровождения | Наличие библиотек | Наличие документации на русском языке | Скорость разработки |
| Важность критерия | Низкая | Средняя | Высокая | Ниже средней | Ниже высокой |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий.

Таблица 2.

Оценка языков программирования.

|  |  |  |  |
| --- | --- | --- | --- |
|  | C++ | Python | Object Pascal |
| Участие в корпоративном проекте | 10 | 8 | 4 |
| Простота сопровождения | 7 | 10 | 3 |
| Наличие библиотек | 6 | 10 | 4 |
| Наличие документации на русском языке | 8 | 6 | 6 |
| Скорость разработки | 6 | 10 | 3 |
| Итого баллов | 37 | 44 | 20 |

По результатам сравнения был выбран язык программирования Python.

## **Проектирование сценария**

В данном разделе приведен сценарий использования программы обычным пользователем.

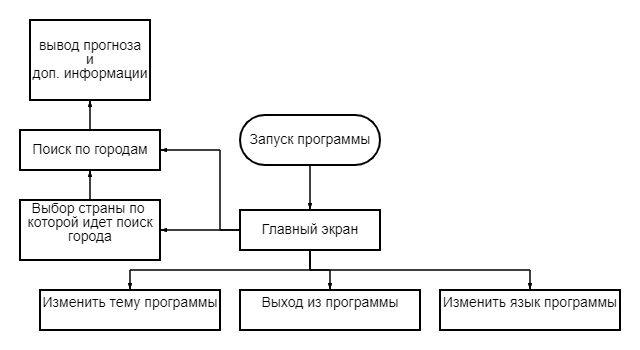


Рисунок 8. Сценарий использования

Пользователь после запуска программы может выполнить 4 действия: изменить тему программы, изменить язык программы, начать поиск по городам всех стран, либо выбрать страну для того случая если в мире несколько таких названий, и непосредственно выход из программы.

При выборе выхода программа заканчивает свою работу, при вписывании города в поиск, программа выдает результаты по поиску, после этого она переходит в режим ожидания

## **Описание главного модуля**

Главный модуль называется “main.py”, он же является точкой запуска. В него подключаются вспомогательные модули “getwikidata.py”,“weather.py”, а также набор библиотек:

1. PyQt5
2. PyOWM
3. Requests
4. Beatifulsoup4
5. Re
6. Webbrowser
7. sys

**Листинг 1. Обработка всех событий в программе.**

def \_\_init\_\_(self, parent=None):

QtWidgets.QWidget.\_\_init\_\_(self, parent)

self.ui = Ui\_MainWindow()

def wiki\_open(self):

def translate(self):

def set\_theme(self):

def find\_owm(self):

Данный метод отслеживает все действия пользователя и в зависимости от конкретного действия вызывает нужный метод.

**Листинг 2. Поиск на Вики**

def wiki\_open(self):

Данный метод наследует основную функцию из модуля getwikedata.py описание функции приведено ниже:

def get\_data(url):

'В начале программа преобразовывает собранный html код в сплошной текст

А затем с помощью языка регулярных выражений находит нужные данные'

rester = []

poyas = ''

plos = ''

html = requests.get(url).text

soup = BeautifulSoup(html, 'lxml')

# Пояс

try:

h2 = soup.findAll('td', {"class": "plainlist"})

h2 = str(h2)

rs = ''.join(re.findall(r"UTC.{4,9}</", h2))

for i in rs:

if i != '<':

poyas += i

else:

break

except Exception:

poyas = '-'

# Население

try:

naselenie = ''.join(re.findall(r"\d", rs[0]))

h1 = soup.findAll('span', {"class": "nowrap"})

h1 = str(h1)

rs = re.findall(r">.{4,10}<su", h1)

naselenie = ''.join(re.findall(r"\d", rs[0]))

except Exception:

naselenie = '-'

# Площадь

try:

h3 = soup.findAll('td', {"class": "plainlist"})

h3 = str(h3)

rs = ''.join(re.findall(r"\d{1,3},?.? ?\d{1,3} ? ?км²</", h3))

rs.replace(' ', ' ')

if rs != '-':

for i in rs:

if i != '<':

plos += i

else:

break

plos = re.sub(r',', '.', plos)

if rs == '':

rs = ''.join(re.findall(r"\d{1,4},\d<", h3))

for i in rs:

if i != '<':

plos += i

else:

break

plos = re.sub(r',', '.', plos)

if rs == '':

rs = ''.join(re.findall(r"\d{1,4},\d{1,4}<", h3))

for i in rs:

if i != '<':

plos += i

else:

break

plos = re.sub(r',', '.', plos)

except:

plos = '-'

# Плотность

try:

plot = int(naselenie) / float(plos[0:-3])

plot = round(plot, 2)

except:

plot = '-'

# Страна

h5 = soup.findAll('th', {"class": "infobox-above"})

h5 = str(h5)

rs = ''.join(re.findall(r">\D{3,25}<", h5))

ct = rs[1:-1]

while '<' in ct:

ct = ct[0:-1]

try:

h6 = soup.findAll('span', {"title": "Показать карту"})

h6 = str(h6)

la = ''.join(re.findall(r"l?o?n?=?ш?\.?\"-?\d{1,3}\.\d{1,10}\"", h6))

lo = ''.join(re.findall(r"l?a?t?=?с?\.?\"-?\d{1,3}\.\d{1,10}\"", h6))

while la.count('l') > 1:

la = la[0:len(la) // 2]

while lo.count('l') > 1:

lo = lo[0:len(lo) // 2]

coords = str(la) + ' ' + str(lo)

while coords[0] != 'l':

coords = coords[1:]

while coords.count('"') > 4:

coords = coords[0:-1]

coords = coords[0:-1]

la = ''.join(re.findall(r'lat=\"\d{1,3}\.?,?\d{1,10}', coords))

lo = ''.join(re.findall(r'lon=\"\d{1,3}\.?,?\d{1,10}', coords))

coords = str(la) + ' ' + str(lo)

rs = re.findall(r">", h6)

except Exception:

coords = '-'

rester.append(str(naselenie))

rester.append(str(plos))

rester.append(str(plot))

rester.append(str(ct))

rester.append(str(coords))

rester.append(str(poyas))

return rester

Данный метод парсит и заполняет поля в зависимости от того какой город выбрал пользователь.

Для того, чтобы поднять универсальность программы была добавлена функция смены языка, за это отвечает модуль translate(self), для сокращения логической нагрузки этот метод сделан структурно, и при масштабировании может испытывать проблему мульти условий. Код приложен в листинге 3.

**Листинг 3. Перевод**

def translate(self):

a = self.ui.info.text()

if a == 'Информация':

self.ui.people.setText('Population')

self.ui.plos.setText('Area')

self.ui.plot.setText('Density')

self.ui.country1.setText('Country')

self.ui.klim.setText('Lat|LON')

self.ui.poyas.setText('Timezone')

self.ui.info.setText('Information')

self.ui.pushButton.setText('RU')

self.ui.comment1.setText('Enter: City or Lat/Lon coordinates')

self.ui.comment2.setText('Accurate country search in alpha order')

if self.ui.city2result.text() == 'Город':

self.ui.city2result.setText('City')

else:

self.ui.people.setText('Население')

self.ui.plos.setText('Площадь')

self.ui.plot.setText('Плотность')

self.ui.country1.setText('Страна')

self.ui.klim.setText('Ш|Д')

self.ui.poyas.setText('Пояс')

self.ui.info.setText('Информация')

self.ui.pushButton.setText('ENG')

self.ui.comment1.setText('Введите: Город либо Ш/Д координаты')

self.ui.comment2.setText('Точный поиск по странам в алф. порядке')

if self.ui.city2result.text() == 'City':

self.ui.city2result.setText('Город')

pass

Данный метод запускает проверку на текущия язык и заменяет все найденные слова на другой язык, в данном случает выступает пара РУС/ENG

**Листинг 4. Смена темы**

В программе предусмотрена смена тема, и она выполняется посредством метода set\_theme(self) Всего в программе реализовано 5 различных тем:

Moon – основной стиль программы

Cocktail-летний стиль

Oni-стиль с высокой контрасностью

Social – Стиль пониженной Контрастности

BigOrange – Доп Стиль

Ниже будет приложен пример оформления кода для одного из стилей:

#PyQt5 использует язык каскадных таблиц CSS

**Cocktail:**

self.ui.centralwidget.setStyleSheet("background-color: rgb(255, 255, 255);")

self.ui.pushButton.setStyleSheet("QPushButton{\n"

"border:0px solid rgb(12, 40, 71);\n"

"background-color: rgb(255, 17, 108);\n"

"border-radius:20px;\n"

"color: rgb(255, 255, 255);\n"

"}\n"

"QPushButton:hover{\n"

"background-color: rgb(85, 85, 255);\n"

"}\n"

"QPushButton:pressed{\n"

"}")

self.ui.enterbut\_2.setStyleSheet("QPushButton{\n"

"background-color: rgb(255, 17, 108);\n"

"border:none;\n"

"}\n"

"")

self.ui.finding.setStyleSheet("QLineEdit{\n"

"background-color: rgb(255, 17, 108);\n"

"border:2px solid rgb(206, 224, 255);\n"

"border-color:rgb(21, 70, 113);\n"

"border-radius:11px;\n"

"color:rgb(222, 252, 255);\n"

"padding-left:45px;\n"

"}\n"

"QLineEdit:hover{\n"

"border-color: rgb(34, 115, 186);\n"

"}\n"

"QLineEdit:focus{\n"

"border-color: rgb(85, 85, 255);\n"

"}")

self.ui.people.setStyleSheet("border:2px solid black;\n"

"color:rgb(222, 252, 255);\n"

"border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.plos.setStyleSheet("border:2px solid black;\n"

"color:rgb(222, 252, 255);\n"

"border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.plot.setStyleSheet("border:2px solid black;\n"

"color:rgb(222, 252, 255);\n"

"border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.poyas.setStyleSheet("border:2px solid black;\n"

"color:rgb(222, 252, 255);\n"

"border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.klim.setStyleSheet("border:2px solid black;\n"

"color:rgb(222, 252, 255);\n"

"border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.country1.setStyleSheet("border:2px solid black;\n"

"color:rgb(222, 252, 255);\n"

"border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.cityresult.setStyleSheet("background-color: rgb(255, 17, 108);\n"

"color: rgb(255, 255, 255);\n"

"border-radius:15px;\n"

"padding-left:50px;\n"

"padding-right:20px;")

self.ui.peopleresult.setStyleSheet("color: rgb(85, 85, 255);\n"

"border-bottom:2px solid black;\n"

"padding-bottom:4px;")

self.ui.plosresult.setStyleSheet("color: rgb(85, 85, 255);\n"

"border-bottom:2px solid black;\n"

"padding-bottom:4px;")

self.ui.plotresult.setStyleSheet("color: rgb(85, 85, 255);\n"

"border-bottom:2px solid black;\n"

"padding-bottom:4px;")

self.ui.countryresult.setStyleSheet("color: rgb(85, 85, 255);\n"

"border-bottom:2px solid black;\n"

"padding-bottom:4px;")

self.ui.klimresult.setStyleSheet("color: rgb(85, 85, 255);\n"

"border-bottom:2px solid black;\n"

"padding-bottom:4px;")

self.ui.poyasresult.setStyleSheet("color: rgb(85, 85, 255);\n"

"border-bottom:2px solid black;\n"

"padding-bottom:4px;")

self.ui.infoicon.setAutoFillBackground(False)

self.ui.infoicon.setStyleSheet("background:none;\n"

"")

self.ui.info.setStyleSheet("color: rgb(255, 255, 255);\n"

"background-color: rgb(255, 17, 108);\n"

"border-radius:15px;")

self.ui.wikibut.setStyleSheet("QPushButton{\n"

"border:0px solid rgb(12, 40, 71);\n"

"background-color: rgb(255, 17, 108);\n"

"border-radius:20px;\n"

"}\n"

"QPushButton:hover{\n"

"background-color: rgb(85, 85, 255);\n"

"}\n"

"QPushButton:pressed{\n"

"}")

self.ui.homeicon.setStyleSheet("border-radius:10px;\n"

"background-color: rgb(255, 17, 108);")

self.ui.city2result.setStyleSheet("color: rgb(255, 255, 255);\n"

"border-bottom:2px solid white;\n"

"background-color: rgb(255, 17, 108);\n"

"padding-bottom:7px;\n"

"border-radius:16px;")

self.ui.scrollArear.setStyleSheet("background-color:rgb(255, 17, 108);\n"

"color:white;\n"

"")

self.ui.dfmain.setStyleSheet("color:white;")

self.ui.d11.setStyleSheet("background-color:none;")

self.ui.data1.setStyleSheet("border:none;")

self.ui.d02.setStyleSheet("background-color:none;")

self.ui.data2.setStyleSheet("border:none;")

self.ui.d5f.setStyleSheet("color:white;")

self.ui.d35.setStyleSheet("color:white;")

self.ui.d185.setStyleSheet("color:white;")

self.ui.d05.setStyleSheet("background-color:none;;")

self.ui.d185result.setStyleSheet("color:white;")

self.ui.d215result.setStyleSheet("color:white;")

self.ui.d35result.setStyleSheet("color:white;")

self.ui.d155.setStyleSheet("color:white;")

self.ui.d125result.setStyleSheet("color:white;")

self.ui.d95.setStyleSheet("color:white;")

self.ui.d215.setStyleSheet("color:white;")

self.ui.d65result.setStyleSheet("color:white;")

self.ui.d95result.setStyleSheet("color:white;")

self.ui.d155result.setStyleSheet("color:white;")

self.ui.data5.setStyleSheet("color:white;")

self.ui.d65.setStyleSheet("color:white;")

self.ui.d05result.setStyleSheet("color:white;")

self.ui.d125.setStyleSheet("color:white;")

self.ui.d03.setStyleSheet("background-color:none;")

self.ui.data3.setStyleSheet("border:none;")

self.ui.d04.setStyleSheet("background-color:none;")

self.ui.data4.setStyleSheet("border:none;")

self.ui.setcontry.setStyleSheet("QComboBox::drop-down{\n"

"image: url(:/newPrefix/rpoint.png);\n"

"padding-left:3px;\n"

"}\n"

"\n"

"QComboBox:hover{\n"

"image: none;\n"

"}")

self.ui.enterbut.setStyleSheet("QPushButton{\n"

"border:0px solid rgb(12, 40, 71);\n"

"background-color: rgb(255, 17, 108);\n"

"border-radius:20px;\n"

"}\n"

"QPushButton:hover{\n"

"background-color: rgb(85, 85, 255);\n"

"}\n"

"QPushButton:pressed{\n"

"}")

self.ui.comment1.setStyleSheet("color: black;")

Данный метод изменяет тему окна для более удобного использования в зависимости от времени суток и места нахождения пользователя

**Листинг 5. Поиск погоды**

Метод реализован также в главном модуле и называтся find\_owm, функционал описан ниже:

def find\_owm(self):

try:

lagr = self.ui.setcontry.currentText()

if lagr == '':

lagr = 'RU'

a = self.ui.finding.text().capitalize().strip()

r, m = wk.get\_id(a, lagr.upper())

if r == 524901 and (a != 'Moscow' and a != 'Москва'):

ress = ['---------', '---------', '---------', '---------', '---------', '---------', '---------',

'---------', '---------', '---------', '---------', '---------', '---------', '---------',

'---------', '---------', '---------', '---------', '---------', '---------', '---------',

'---------', '---------', '---------', '---------', '---------', '---------', '---------',

'---------', '---------', '---------', '---------', '---------', '---------', '---------',

'---------', '---------', '---------', '---------', '---------', '---------', '---------',

'---------', '---------']

m = ''

else:

ress = wk.request\_forecast(r)

ss = gwd.get\_data('https://ru.wikipedia.org/wiki/' + self.ui.finding.text())

print(ss)

self.ui.data1.setText('Сегодня:')

self.ui.data2.setText('Завтра:')

self.ui.data3.setText('Послезавтра:')

self.ui.data4.setText('Через 3 дня:')

self.ui.data5.setText('Через 4 дня:')

self.ui.peopleresult.setText(ss[0])

self.ui.plosresult.setText(ss[1])

self.ui.plotresult.setText(ss[2])

self.ui.countryresult.setText(m)

self.ui.klimresult.setText(ss[4])

self.ui.poyasresult.setText(ss[5])

self.ui.d31result.setText('Темп:' + ress[1][1] + '°C')

self.ui.d11result.setText('Темп:' + ress[0][1] + '°C')

self.ui.d61result.setText('Темп:' + ress[2][1] + '°C')

self.ui.d91result.setText('Темп:' + ress[3][1] + '°C')

self.ui.d121result.setText('Темп:' + ress[4][1] + '°C')

self.ui.d151result.setText('Темп:' + ress[5][1] + '°C')

self.ui.d181result.setText('Темп:' + ress[6][1] + '°C')

self.ui.d211result.setText('Темп:' + ress[7][1] + '°C')

self.ui.d02result.setText('Темп:' + ress[8][1] + '°C')

self.ui.d32result.setText('Темп:' + ress[9][1] + '°C')

self.ui.d62result.setText('Темп:' + ress[10][1] + '°C')

self.ui.d92result.setText('Темп:' + ress[11][1] + '°C')

self.ui.d122result.setText('Темп:' + ress[12][1] + '°C')

self.ui.d152result.setText('Темп:' + ress[13][1] + '°C')

self.ui.d182result.setText('Темп:' + ress[14][1] + '°C')

self.ui.d212result.setText('Темп:' + ress[15][1] + '°C')

self.ui.d03result.setText('Темп:' + ress[16][1] + '°C')

self.ui.d33result.setText('Темп:' + ress[17][1] + '°C')

self.ui.d63result.setText('Темп:' + ress[18][1] + '°C')

self.ui.d93result.setText('Темп:' + ress[19][1] + '°C')

self.ui.d123result.setText('Темп:' + ress[20][1] + '°C')

self.ui.d153result.setText('Темп:' + ress[21][1] + '°C')

self.ui.d183result.setText('Темп:' + ress[22][1] + '°C')

self.ui.d213result.setText('Темп:' + ress[23][1] + '°C')

self.ui.d04result.setText('Темп:' + ress[24][1] + '°C')

self.ui.d34result.setText('Темп:' + ress[25][1] + '°C')

self.ui.d64result.setText('Темп:' + ress[26][1] + '°C')

self.ui.d94result.setText('Темп:' + ress[27][1] + '°C')

self.ui.d124result.setText('Темп:' + ress[28][1] + '°C')

self.ui.d154result.setText('Темп:' + ress[29][1] + '°C')

self.ui.d184result.setText('Темп:' + ress[30][1] + '°C')

self.ui.d214result.setText('Темп:' + ress[31][1] + '°C')

self.ui.d05result.setText('Темп:' + ress[32][1] + '°C')

self.ui.d35result.setText('Темп:' + ress[33][1] + '°C')

self.ui.d65result.setText('Темп:' + ress[34][1] + '°C')

self.ui.d95result.setText('Темп:' + ress[35][1] + '°C')

self.ui.d125result.setText('Темп:' + ress[36][1] + '°C')

self.ui.d155result.setText('Темп:' + ress[37][1] + '°C')

self.ui.d185result.setText('Темп:' + ress[38][1] + '°C')

self.ui.d215result.setText('Темп:' + ress[39][1] + '°C')

self.ui.d215result.setText('Темп:' + ress[40][1] + '°C')

self.ui.cityresult.setText(self.ui.finding.text().capitalize() + ' ' + m)

self.ui.city2result.setText(self.ui.finding.text().capitalize() + ' ' + m)

except Exception:

self.ui.cityresult.setText(self.ui.finding.text().capitalize() + ' ' + m)

self.ui.city2result.setText(self.ui.finding.text().capitalize() + ' ' + m)

pass

Стоит отметить что функция get\_data реализована также, в другом модуле,

а именно weather.py, в нем определена функция которая собирает нужные данные о погоде, код функции приложен ниже:

**Листинг 6. Поиск погоды**

def get\_id(s\_city\_name, gpp="RU"):

appid = "3fd96f5c9c5ac865da969d95913e2196"

try:

res = requests.get("http://api.openweathermap.org/data/2.5/find",

params={'q': s\_city\_name, 'type': 'like', 'units': 'metric', 'lang': 'ru', 'APPID': appid})

data = res.json()

cities = ["{} ({})".format(d['name'], d['sys']['country'])

for d in data['list']]

if gpp != 'RU':

for i in range(len(cities)):

if str(cities[i][-3:-1]) == gpp:

city\_id = data['list'][i]['id']

dd = cities[i][-3:-1]

break

else:

city\_id = data['list'][1]['id']

dd = cities[i][-3:-1]

else:

city\_id = data['list'][0]['id']

dd = cities[0][-3:-1]

except Exception as e:

city\_id = 524901;

dd = "RU"

pass

return city\_id, dd

def request\_forecast(city\_id):

appid = "3fd96f5c9c5ac865da969d95913e2196"

res = requests.get("http://api.openweathermap.org/data/2.5/forecast",

params={'id': city\_id, 'units': 'metric', 'lang': 'BZ', 'APPID': appid})

data = res.json()

fa = []

n = (data['city']['name'], data['city']['country'])

for i in data['list']:

e = (i['dt\_txt'])[:16], '{0:+3.0f}'.format(i['main']['temp']),

'{0:2.0f}'.format(i['wind']['speed']) + " м/с",

i['weather'][0]['description']

fa.append(e)

if len(fa) < 48:

m = ''.join(list(fa[0]))

l = ('---------')

m = int(m[11:13])

e = 8 - (24 - m) // 3

for i in range(e):

fa.insert(0, l)

for i in range((24 - m) // 3):

fa.append(l)

return fa

Данный метод запрашивает данные с сайта, который собирает данные о погоде в мире.

## **Описание спецификаций к модулям**

В данном разделе описаны публичные члены модулей курсового проекта.

В главном модуле (main.py) содержатся публичные методы класса MyWin который наследуется от класса Ui\_MainWindow из модуля MoonStyle.py. Так как все эти методы были описаны в разделе 2.4, в данном разделе описание данных методов опущено. В данном модуле также есть несколько объектов:

1. app – объект, который создает системное окно и объект самого приложения.
2. myapp – объект интерфейса представленный в виде стека окон.
3. ui – объект который представляет собой сам интерфейс.

В модуле, который реализует интерфейс (MoonStyle.py) также содержатся публичные методы класса Ui\_MainWindow, которые будут описаны ниже в разделе 2.7, в данном разделе они описаны не будут. В данном классе так же есть публичные объекты, все они будут перечислены и описаны в этом разделе.

Объекты класса Ui\_MainWindow:

1. comboBox – объект посредством которого осуществляется выбор темы .
2. info – надпись информации
3. peopleresult – окно вывода населения
4. plosresult – окно вывода площади
5. plotresult –окно вывода плотности
6. countryresult –окно вывода страны
7. klimresult–окно вывода координат
8. poyasresult–окно вывода пояса
9. dfmain–окно вывода информации о погоде
10. centralwidget – разделитель, который позволяет прижать виджет
11. finding–Поле ввода города
12. enterbut–Кнопка отправки запроса
13. cityresult–Вывод справки о городе
14. wikibut–кнопка для перехода на вики города
15. city2result–Кнопка для вставки названия
16. comment1–Пояснение к вводимым данным 1
17. comment2– Пояснение к вводимым данным 2
18. pushButton–Кнопка смены языка

## **Описание модулей**

Кроме главного модуля программа содержит еще один модуль который реализует интерфейс (moomStyle.py).

Полный код модуля приведен в приложении 2. В этом разделе приведены все методы, которые содержаться в класс QtMaibWindow из модуля moonStyle.py.

**Листинг 7. Методы класса QtMainWindow.**

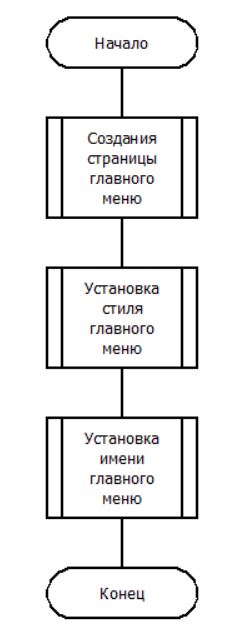


Рисунок 11. Блок-схема метода show

## **Описание тестовых наборов модулей**

В этом разделе будет визуализирован процесс работы приложения в виде тестов:

Тест 1. Запуск приложения.

Действия: Запустить exe файл

Ожидаемый результат: Успешный запуск окна

Результат теста:

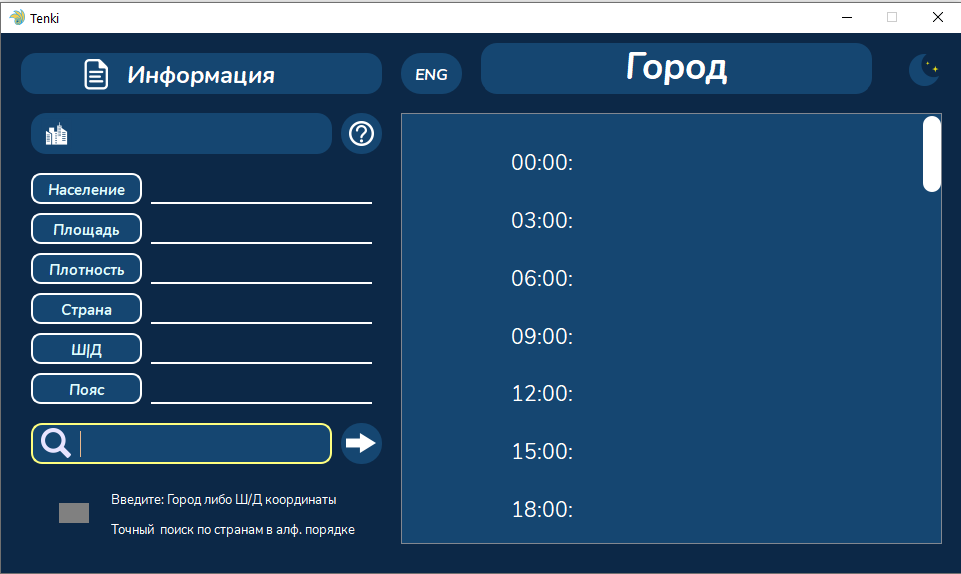


Рисунок 12. Запуск программы

Тест 2. Смена темы:

Действия: Нажать на кнопку Вверху справа, в появившейся новой колонке выбрать нужную иконку

Ожидаемы результат: Смена палитры цветов в приложение

Результат тесте:

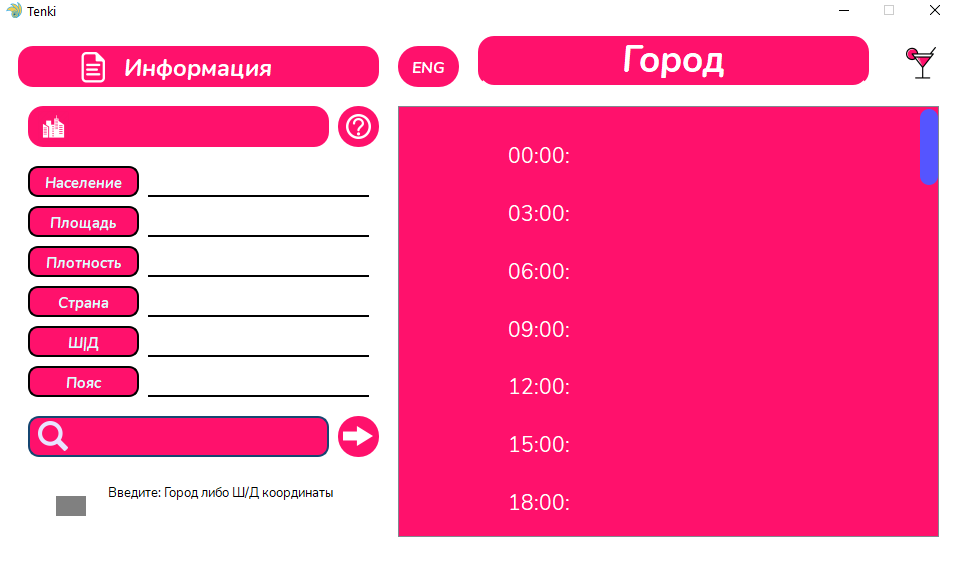


Рисунок 13. Тема Cocktail

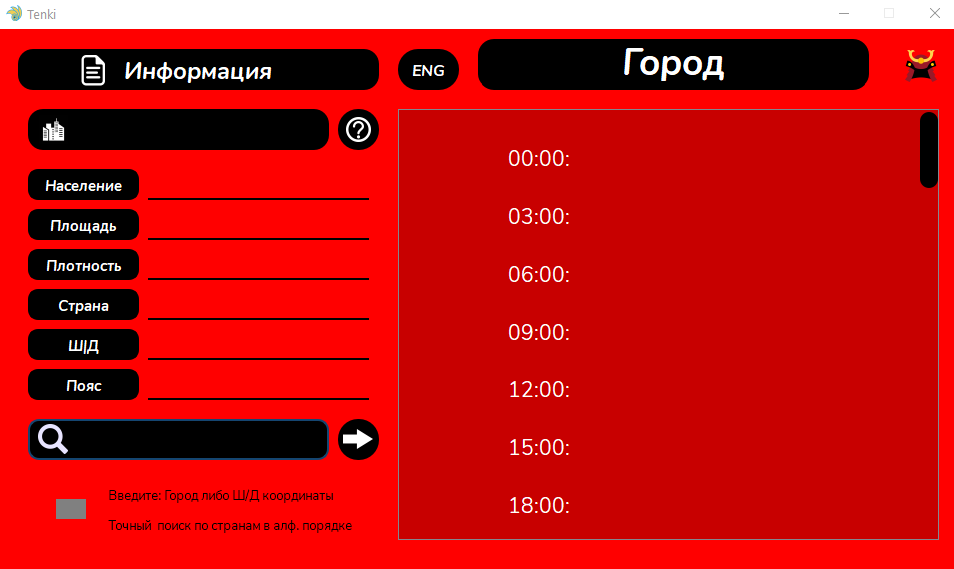


Рисунок 14. Тема Oni

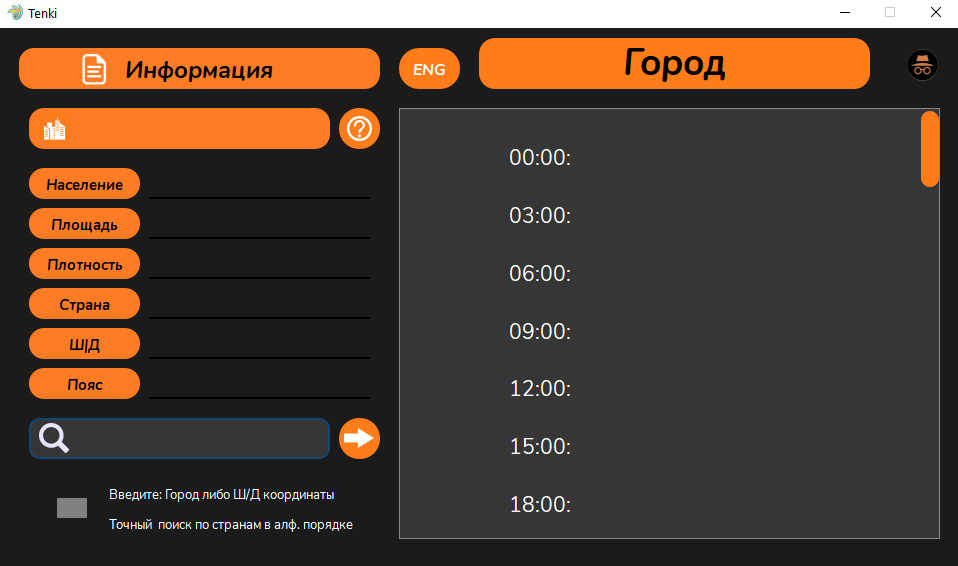


Рисунок 15. Тема BigOrange

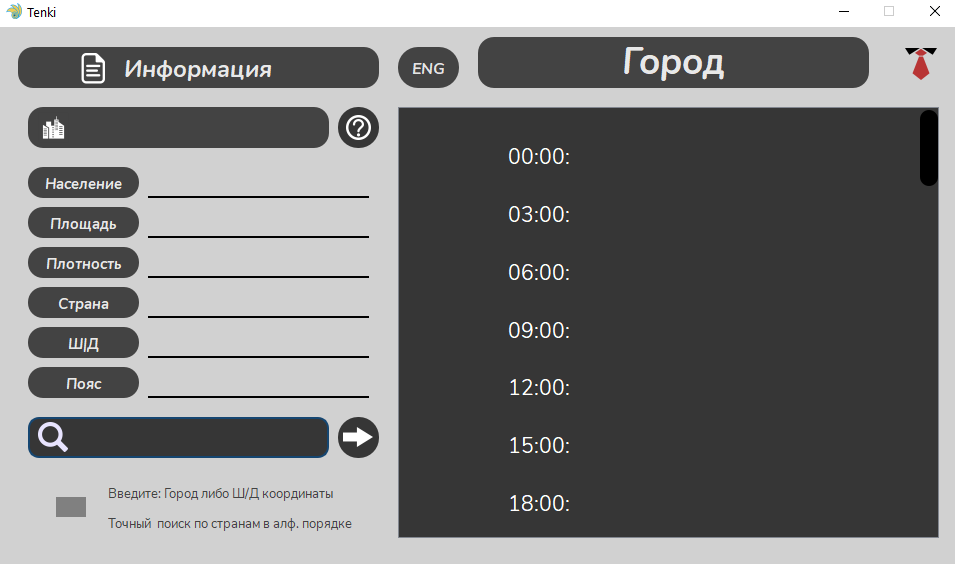


Рисунок 16. Тема Social

Тест 3. Поиск любого города мира на английском или русском языке

Действия: Ввод желаемого города на в поли внизу слева и указание страны если вдруг есть города (тезки) например в США есть город Москва(население 300чел)

Ожидаемый результат: после клика по кнопке или нажатия Enter выведется информация о городе(Поиск вики только для крупных или уникальных русских городов или больших иностранных)

Результат теста:

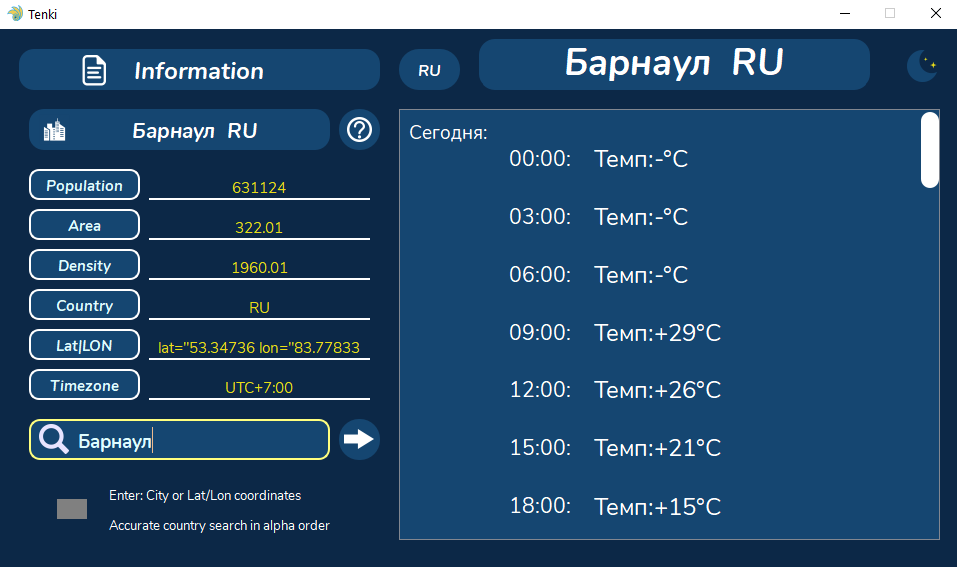


Рисунок 17. Пример поиска на русском русского города



Рисунок 18. Пример поиска Иностранного города на английском

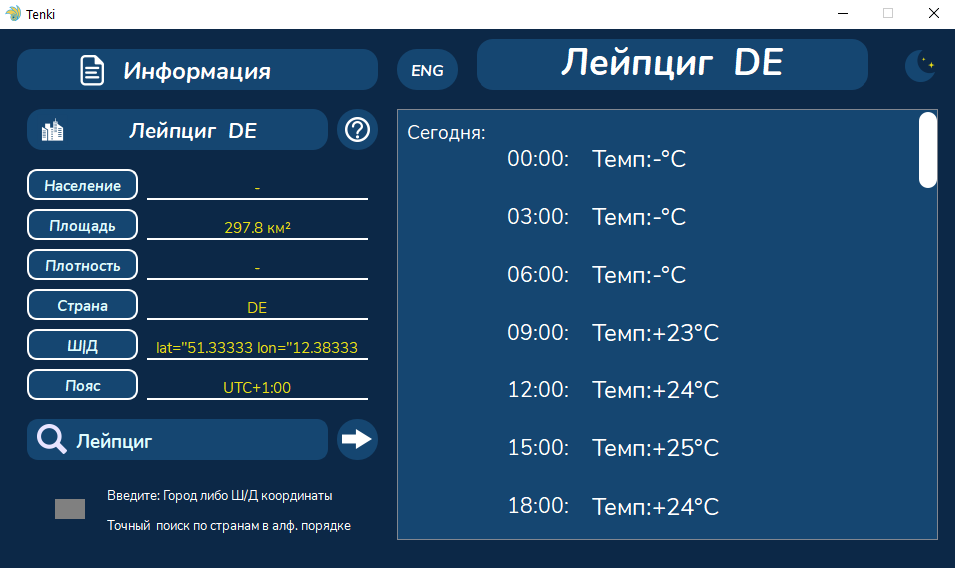


Рисунок 19. Пример поиска Иностранного города на русском

Тест 4. Смена языка в приложении

Действия: Нажать на Кнопку Eng

Ожидаемый результат: Смена названия 6-ти ячеек и полей информации и города

Результат теста:

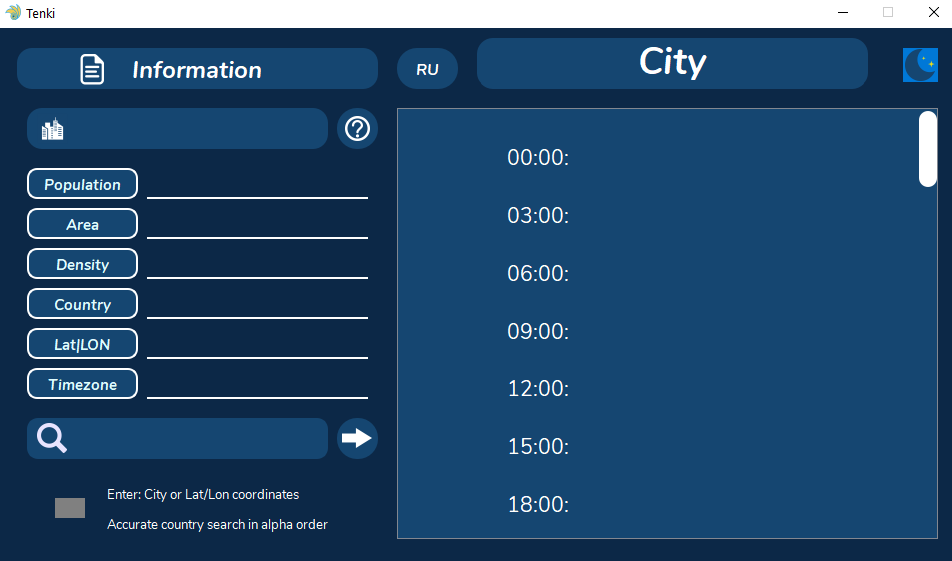


Рисунок 20. Пример Английского языка

Тест 5. Поиск на Вики

Действия: Нажать на Кнопку с вопросительным зыком

Ожидаемый результат: Откроется википедия если город есть в России, иначе откроет стартовую страницу википедии

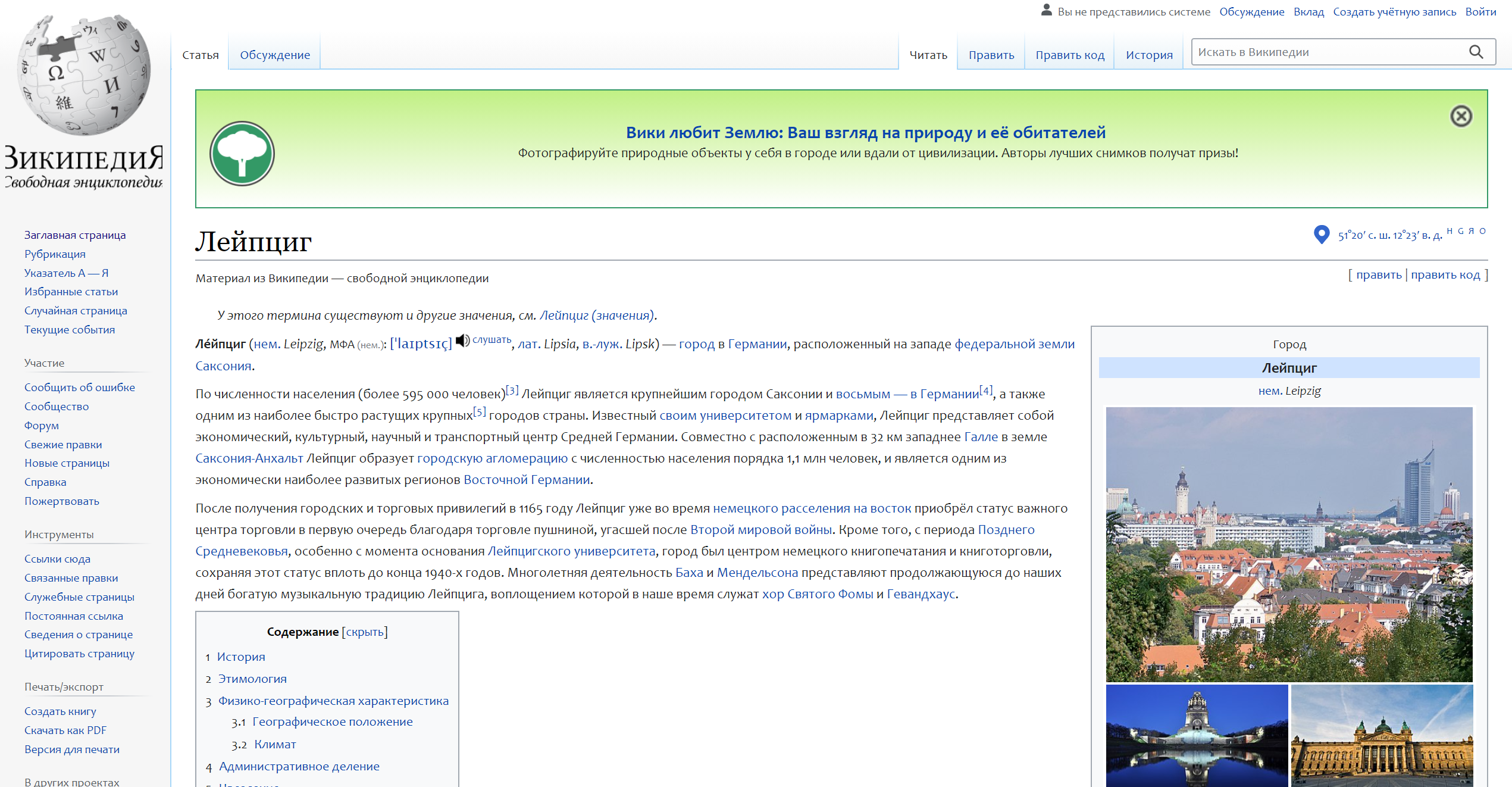
Результат теста: 

Рисунок 21. Пример результата работы кнопки

## **. Описание применения средств отладки.**

Для Грамотной отладки в PyCharm предусмотрен Debug режим который не только позволяет наглядно узнать что где находится но еще и строит иерархию зависимостей и наследования

Ниже представлен пример данных из отладчика PyCharm

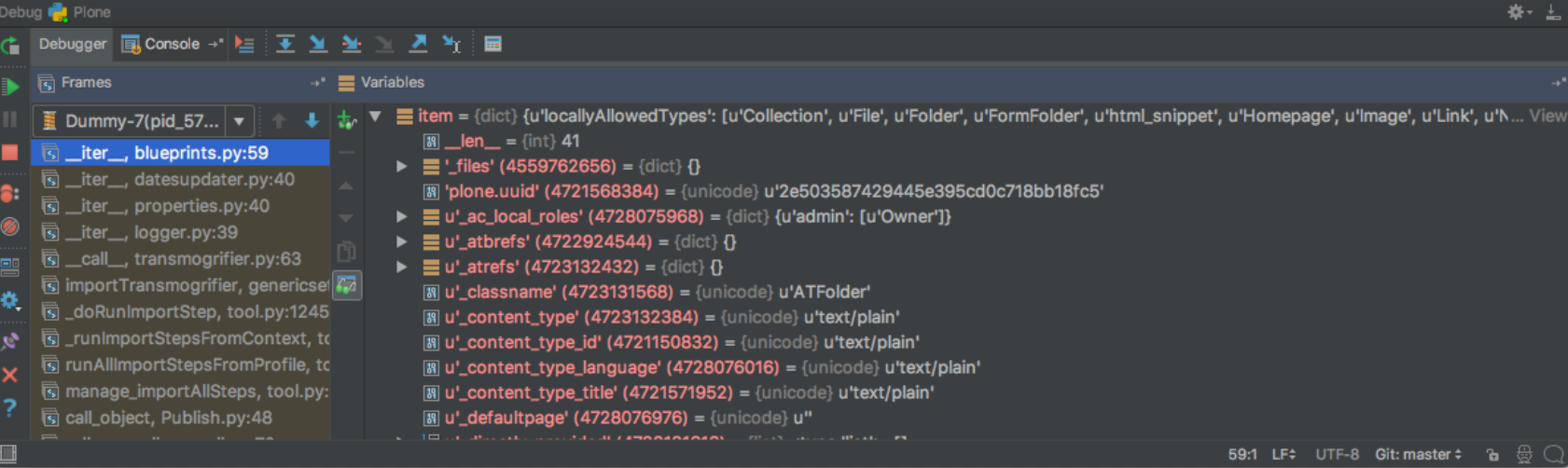


Рисунок 22. Пример Запущенного отладчика

С помощью такого подхода можно отловить не только те баги которые связаны с неточностями во время написания синтаксиса и явно себя проявляющие на этапе запуска, по типу ‘ ValueError’ , но и определить пример неправильного наследования или утечку памяти.

Анализ оптимальности использования памяти и быстродействия

В данном разделе будет проведен анализ оптимальности использования памяти и быстродействия программы.

Список принятых оптимальных решений для оптимизации на стороне пользователя:

1. Весь код был разделен на функции и методы чтобы избежать недостатки структурного подхода, а именно построчных, линейных вычислений, так как не каждое действие приводит к полному обновлению всех элементов
2. Для оптимизации размеров все дополнительные данные приходят из вне а именно с википедии, ведь хранение этих данных в БД снизило бы производительность программы и увеличило бы время получения ответа на запрос
3. По возможности принимались решения, которые не предполагают хранение данных в массивах.

# **Эксплуатационная часть**

## **Руководство пользователя**

**АННОТАЦИЯ**

В данном программном документе приведено руководство пользователя по применению и эксплуатации программы «Tenki», предназначенной для определения температуры на текущую неделю

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых пользователь управляет выполнением программы, а также ответы программы на эти команды.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 [[1]](#footnote-1)1), ГОСТ 19.103-77 [[2]](#footnote-2)2), ГОСТ 19.104-78\* [[3]](#footnote-3)3), ГОСТ 19.105-78\* [[4]](#footnote-4)4), ГОСТ 19.106-78\* [[5]](#footnote-5)5), ГОСТ 19.505-79\* [[6]](#footnote-6)6), ГОСТ 19.604-78\* [[7]](#footnote-7)7)).

1. **Назначение программы**
   1. **Функциональное назначение программы**

Специальное программное обеспечение «Tenki» используется для Удобного мониторига температуры

* 1. **Эксплуатационное назначение программы**

Специальное программное обеспечение «Tenki» может эксплуатироваться любым пользователем компьютера, который хочет узнать погоду в городе и знает русский или английский язык

* 1. **Состав функций**
     1. **Функция вывода погоды.**

Эта функция выполняет запрос Api, на сайт который собирает данные о погоде, после чего заполняет элементы интерфейса данными, которые он получил

* + 1. **Функция смены языка**

Эта функция позволяет изменить пунктов приложения с руского на английский и обратно

* + 1. **Функция изменения темы**

Эта функция позволяет сменить тему приложения

* + 1. **Поиска на вики**

Эта функция осуществляет запрос на вики, и собирает данных для их дальнейшего оборота внутри программы

* + 1. **Функция возвращения в главное меню.**

Эта функция позволяет вернуться в главное меню.

1. **Условия выполнения программы**
   1. **Минимальный состав аппаратных средств**

ОС: Windows 10

Процессор: Как минимум 1 ГГц

ОЗУ: 256 МБ(для 32-разрядных систем).

Место на жестком диске: 103 МБ (для 32-разрядных систем) или 120 МБ (для 64-разрядных систем).

Дисплей: 1024 x 768.

* 1. **Минимальный состав программных средств**

Дополнительные программные средства не требуются. Программа поставляется в .exe

* 1. **Требование к персоналу (пользователю)**

Конечный пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

1. **Выполнение программы**
   1. **Загрузка и запуск программы**

После загрузки на Компьютер необходимо создать папку где будет храниться программа, а также создать ярлык на рабочем столе

Для запуска необходимо дважды кликнуть на «Tenki.exe» После чего подождать запуск.

Путь до каталога «Tenki.exe» должен содержать только латиницу.

Запустите программу «Tenki», откроется окно главного меню:

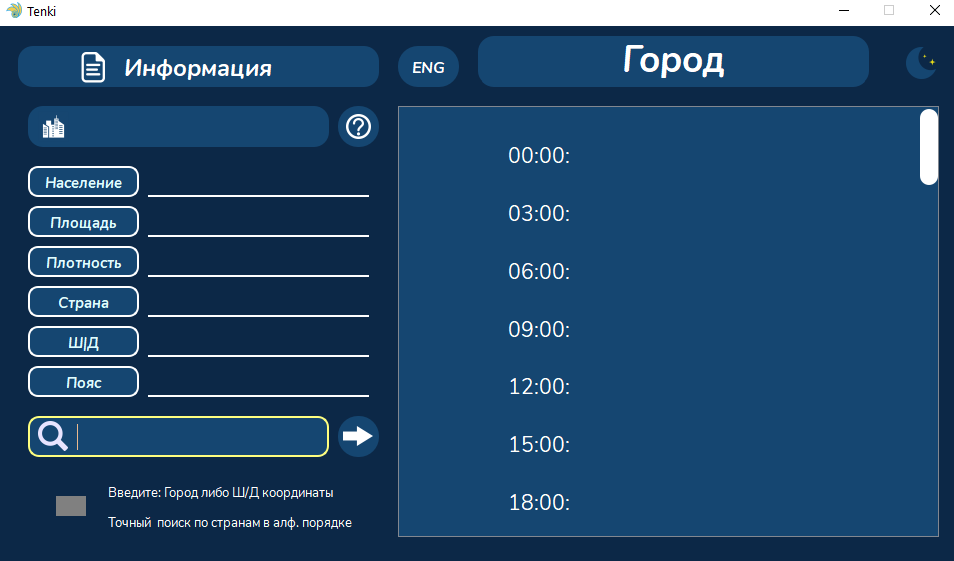


Рисунок 22. Главное меню

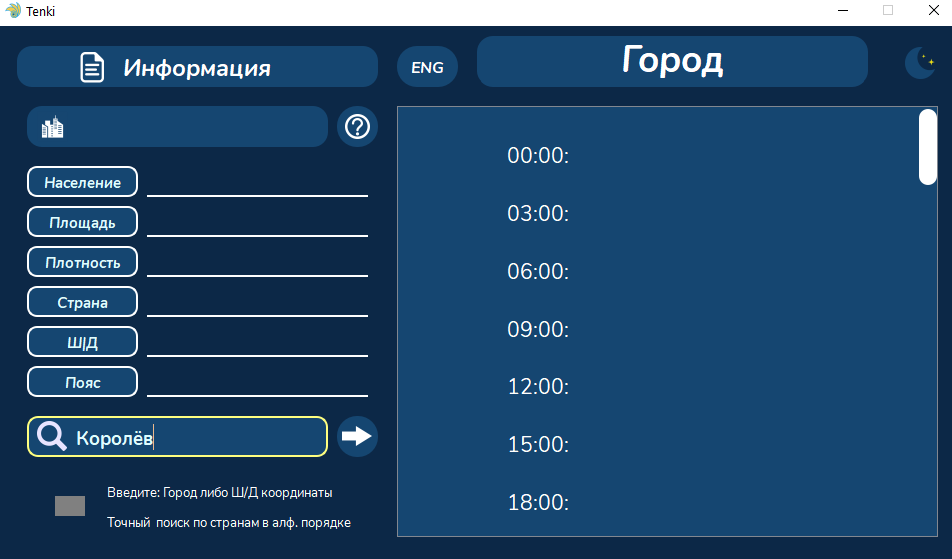


Рисунок 23. Ввод города на примере Королёва

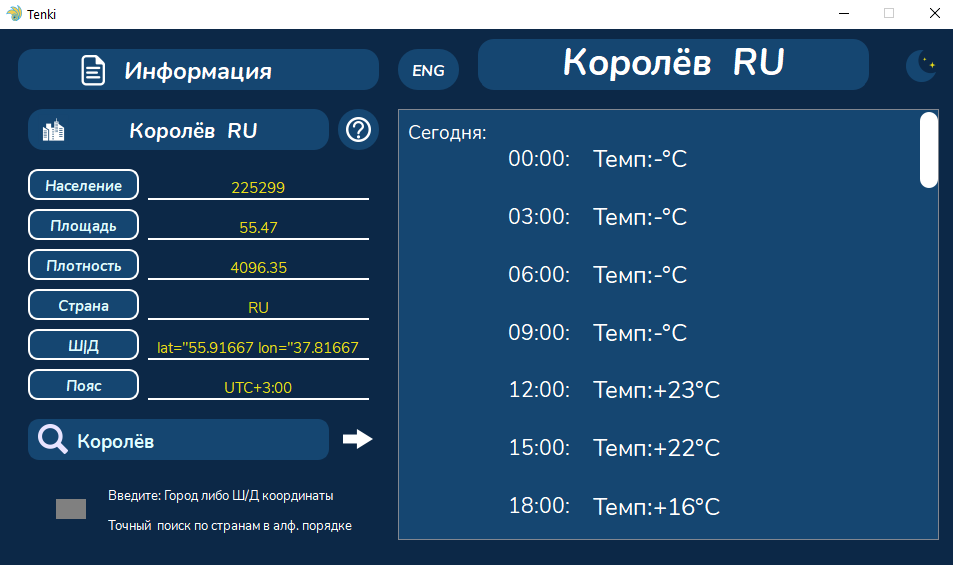


Рисунок 24. Нажатие на кнопку ввода(стрелочка)

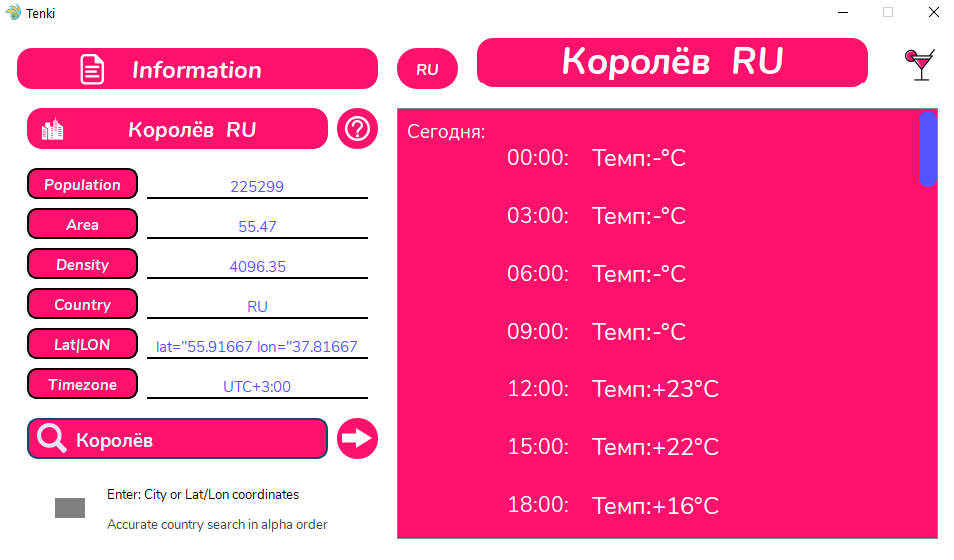


Рисунок 25. (Опционально) Сменить тему и язык

## **Элементы управления**

Кнопки меню имеют следующие функции:

Стрелочка – Осуществляет выполнения запроса

Серый квадрат –открывает меню для уточнения страны

Иконка луны вверху справа – Открывает меня стилей

Ru – Меняет язык

Вопросительный знак – открывает Страницу на википедии

# **Заключение**

В результате выполнения курсового проекта была написана программа «Tenki.exe» для упрощения определения погоды в мире в любой момент

В ходе работы были проанализированы предметная область, существующие разработки, посвященные данному направлению, получены практические навыки по созданию UI с помощью библиотеки PyQt5.

Также планируется продолжать работу над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены ниже.

To-do лист:

1. Расширить список данных для вывода, например Скорость ветра
2. Адаптивность окна посредством динамического масштабирования и и переноса элементов внутри программы
3. Доработка интерфейса с целью обеспечения ввода без клавиатуры.
4. Создание кроссплатформенности.
5. Улучшения алгоритмов для ускорения процесса сбора данных.
6. Создание истории поисковых запросов.
7. Добавить иконки флагов государств.

# **Список литературы и интернет-источников**

1. Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Сидорова-Виснадул; под ред. Л.Г. Гагариной. — Москва: ИД «ФОРУМ»: ИНФРА-М, 2019. — 400 с. — (Высшее образование: Бакалавриат). - ISBN 978-5-16-104071-3. - Текст: электронный. - URL: <https://znanium.com/catalog/product/1011120>
2. Скрапинг веб-сайтов с помощью Python Райан Митчелл
3. Create Simple GUI Applications, with Python Qt5 by Martin Fitzpatrick
4. В. Дронов, Н. Прохорёнок - Python 3 и PyQt 5. Разработка приложений. 2 изд.

1. 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)