

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

**РАЗРАБОТКА ПРИЛОЖЕНИЯ «TEXTKING» ДЛЯ ГЕНЕРАЦИИ
СОЧИНЕНИЙ**

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-201(1)
Кривошапко Михаил Игоревич

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель
старший преподаватель кафедры
компьютерной инженерии и моделирования

(оценка)

Чабанов В.В.

(подпись, дата)

Симферополь, 2021

РЕФЕРАТ

Разработка приложения «TextKing» для генерации сочинений на свободную тему
– Симферополь: ФТИ КФУ им. В. И. Вернадского, 2021. – 25 с., 20 ил., 5 ист.

Объект разработки – приложения TextKing для генерации сочинений.

Цель работы – создание рабочего десктопного приложения с реализацией клиент-серверной системы. Создание клиента с использованием фреймворка Qt. Создание сервера с использованием фреймворка Flask языка программирования Python, реализация обмена данными клиента с сервером посредством POST и GET запросов.

Были рассмотрены различные варианты реализации клиент-серверных систем, в ходе рассмотрения которых было принято решение использовать POST и GET запросы для взаимодействия C++ Qt клиента и Python Flask сервера, в виду удобства использования.

ПРОГРАММИРОВАНИЕ, C++, Python, Qt Framework, Flask, API, нейронные сети.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	4
ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ.....	5
1.1 Цель проекта.....	5
1.2 Существующие аналоги.....	5
1.3 Основные отличия от аналогов.....	5
1.4 Техническое задание.....	6
ГЛАВА 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ.....	8
2.1 Анализ инструментальных средств.....	8
2.2 Описание структур данных.....	9
2.2.1 Сервер.....	9
2.2.2 Клиент.....	13
ГЛАВА 3 ТЕСТИРОВАНИЕ ПРОГРАММЫ.....	19
3.1 Тестирование исходного кода.....	19
3.2 Тестирование интерфейса пользователя.....	21
ГЛАВА 4 ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА.....	23
4.1 Перспективы технического развития.....	23
4.2 Перспективы монетизации.....	23
ЗАКЛЮЧЕНИЕ.....	24
ЛИТЕРАТУРА.....	25

ВВЕДЕНИЕ

Во время обучения, тестирований на работе и подобного, нередко приходится писать немалое количество сочинений или иного текста, и зачастую от руки. Притом, в большинстве случаев, проверяющие лица не особо вчитываются в написанное по их заданию. Для самих же сдающих, эти сочинения не всегда бывают полезны, особенно на фоне других дел, которые приходится отодвигать на другой план. Поскольку программированию свойственно искать способы упростить рутинные процессы, я решил выбрать эту тему.

Для достижения цели (создания десктопного приложения), требуется изучить: фреймворк C++ Qt для создания клиентской стороны приложения, средства фреймворка Python Flask и API сервиса-нейросети, генерирующей связанный текст на основе данного.

ГЛАВА 1

ПОСТАНОВКА ЗАДАЧИ

1.1 Цель проекта

Основными целями проекта являются:

1. Разработка приложения «TextKing» для генерации сочинений;
2. Получить опыт разработки на популярных фреймворках языков программирования C++ и Python;
3. Получить опыт разработки относительно крупного проекта;
4. Получить опыт разработки приложений с использованием нескольких языков программирования.

1.2 Существующие аналоги

- Essay-generator. Крупное десктопное приложение, позволяющее генерировать текст, однако на английском и без возможности скоропоследующего перевода из печатного в рукописный шрифт.
- PaperTyper. Представлено в виде сайта, однако платное и также, существует лишь на английском языке.
- PapersOwl. Бесплатное приложение, обладает стандартным функционалом.

1.3 Основные отличия от аналогов

Преимуществом моего приложения является понятный пользовательский интерфейс, который не путает пользователя, в отличие от интерфейса аналогов.

Главным преимуществом TextKing является русский язык генерации текста, а также возможность перевода в рукописную форму введенного текста.

Ещё одним преимуществом является “легковесность” и быстрота работы приложения.

1.4 Техническое задание

1. Терминология.

ТЗ – Техническое задание.

Qt – фреймворк для графического пользовательского интерфейса.

Генератор текста – левая половина основного экрана приложения. Запрос к API нейросети GPT-2.

Генератор рукописей – правая половина основного экрана приложения, запрос к серверу-генератору рукописного текста.

2. Цель создания приложения. Пользователь, при включении программы, увидит окно в котором будет два больших поля для ввода текста, генератор текста и генератор рукописей соответственно.

С помощью интуитивно понятного интерфейса, пользователь может:

- 1) настроить параметры дополняемого текста
- 2) контекстно дополнить свои предложения
- 3) перевести текст в изображение с рукописным текстом
- 4) выбрать параметры получаемых рукописных изображений.

3. Требования к приложению.

Проект будет состоять из:

- Клиент пользователя (Qt)
- Сервер обработчик запросов на генерацию. (Flask)

Клиент позволяет:

- сгенерировать текст по заданным пользователем предложениям
- перевести текст в картинку
- выбрать количество предложений генерируемого текста, кастомизировать рукописный текст

Программа должна работать, быть доступной пользователю, не требовала особых усилий при установке и деинсталляции.

4. Сценарии использования.

Грамотно использовать программу пользователь будет, если он:

1. введет свой текст в левую Text Edit графу
2. выберет подходящие параметры (при желании)
3. затем нажмет «сгенерировать текст»
4. получит дополненную версию своего текста.

Если проделать аналогичные действия в правой части программного интерфейса, можно получить изображения с рукописью.

5. Описание интерфейса (экранов). Левая половина приложения посвящена генерации текста, правая – его переводу в рукописную форму. Они будут похожи друг на друга, в них будет:

1. Поле для ввода текста, крупное относительно программного интерфейса.
2. Кнопки для генераций снизу от поля ввода
3. Параметры генераций, справа от полей ввода.

6. Требования к платформе: Windows 10, в будущем возможна компиляция на Ubuntu. Для работы интерпретируемого языка Python в системах, в которых он не установлен, будет произведена сборка приложения с помощью PyInstaller.

ГЛАВА 2

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Анализ инструментальных средств

Перед началом разработки было изучено несколько различных вариантов разработки графического интерфейса как на языке Python, так и на C++. В качестве среды разработки клиентского и серверного приложений был выбран IDE Qt Creator. Программа была переписана с нуля наоборот: изначально планировался C++ сервер и Python клиент. Однако, в связи с бедной документацией библиотеки сервера C++, было решено сделать, как оно есть сейчас: клиент на языке C++, сервер – на языке Python. Принято это решение было по следующим причинам:

- Как уже было сказано, крупные фреймворки имеют более обширную документацию, что облегчает поиск информации касательно необходимых методов
- Удобство визуального редактора Qt Designer для создания графического интерфейса;
- Удобство и минимализм фреймворка Flask по сравнению со всеми аналогами данного на языке C++. для создания серверной части приложения и API.

Также были изучены некоторые встроенные библиотеки фреймворка Qt, такие как:

- QJsonObject, QJsonDocument для обработки JSON;
- QNetwork для отправки HTTP запросов;
- Несколько библиотек для работы с картинками и переводом байтов.

На языке Python была изучена библиотека для работы с изображениями: PIL или pillow. Ключевым моментом в выборе языков программирования послужило обучение им на курсе программирования.

2.2 Описание структур данных

2.2.1 Сервер

Сервер функционирует следующим образом:

1. Если сервер получает POST-запрос на корневой url:
 - 1.1. Сервер забирает данные из запроса по установленному ключу;
 - 1.2. Сервер отправляет запрос с данными от клиента к API нейросети GPT-2, получает из него список результатов дополнения;
 - 1.3. Сервер возвращает клиенту одно из списка результатов дополнения, неотправленное сохраняется в буфере.
2. Если сервер получает GET-запрос на “/vars”:
 - 2.1. Сервер возвращает один из элементов буфера, оставшегося после API запроса.
3. Если сервер получает POST-запрос на “/get_handwrite”:
 - 3.1. Из имеющихся на сервере ресурсов составляется страница тетрадного или иного листа
 - 3.2. Из запроса пользователя составляется оформление страницы.
 - 3.3. После успешного составления страниц, пользователю возвращается его текст в рукописной форме и на листе.

Сервер написан, как уже было написано ранее, на языке программирования Python, с помощью фреймворка Flask.

Немного о Flask. Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые веб возможности.

```

1  # This Python file uses the following encoding: utf-8
2  import json
3  import requests
4  from flask import *
5  from PIL import Image, ImageDraw, ImageFont
6
7
8  class Buffer_msg():
9      def __init__(self, *args):
10         self.buff = []
11
12         for i in args:
13             self.buff.append(i)
14
15     def bring(self):
16         tempo = self.buff.pop()
17         return tempo
18
19     def release(self):
20         self.buff.clear()
21         return
22
23     def __iadd__(self, other):
24         self.buff.extend(other)
25         return self
26

```

Рисунок 2.1. Загрузка библиотек и создание класса буфера

В строках 2-25 в файл загружаются необходимые библиотеки и фреймворки, также создается класс `Buffer_msg`, который впоследствии будет использоваться для сохранения дополнительных строк, генерируемых API выбранной нейросети. Далее, внутри него будут сохраняться варианты дополнения, которые предлагает API.

В фреймворке Python Flask стандартный многопоточный сервер создается весьма просто, для этого достаточно создать переменную сервера и присвоить ей результат “функции `flask.Flask(__name__)`” (рис. 2.2.). Далее необходимо воспользоваться декоратором серверной переменной + “`.route()`” в которой описать конец url-пути.

```

28 global_buff = Buffer_msg()
29 new_imgs = 0
30
31 server = Flask(__name__)
32
33 @server.route("/", methods=['POST'])
34 def index():
35     global global_buff
36
37     global_buff.release()
38
39     req_res = request.get_json()
40
41     print(req_res["len"])
42     print("vse slomal")
43     print(req_res)
44     print(req_res["user_text"])
45
46     if not len(req_res["user_text"]):
47         return ""

```

Рисунок 2.2. Создание сервера и обработка запроса на “/”

На сервер приходит POST-запрос с файлом JSON. В нем содержатся параметры “user_text” и “len”, в которых находятся строка и число соответственно. Далее нужно отправить POST-запрос. С помощью библиотеки “requests” это осуществимо, например, так (рис.2.3). Здесь отправляется запрос “{“prompt” : req_res[“user_text”], “length” : req_res[“len”]}”, где “prompt” – строка, которую нужно продлить по смыслу, а “length” – длина, на которую нейросеть ориентируется при дополнении текста.

```

49 print(111111111111)
50 res = requests.post("https://pelevin.gpt.dobro.ai/generate/", json={"prompt" : req_res["user_text"], "length" : req_res["len"]})
51 print(222222222222)
52 #print(res.content)
53 #print(res.text)
54
55 print(333333333333)
56 print(res.json())
57 print("{\result_list\" : %r\" % (res.json()[\"replies\"][0]) + r\"}") #str.encode('unicode_escape')
58
59 resp = Response("{\result_list\" : %r\" % (res.json()[\"replies\"][0]) + r\"}")
60
61 resp.headers['Access-Control-Allow-Origin'] = '*'
62 resp.headers['Content-Type'] = 'application/json; charset=utf-8'
63
64 temp = res.json()[\"replies\"][1:]
65 global_buff += temp
66
67 return resp
68

```

Рисунок 2.3. Отправка запроса с сервера на сервис API

Как было сказано несколько выше, API нейросети возвращает список возможных дополнений; при запросе на “/” сервер отправляет нулевой (первый) элемент этого списка. Если пользователю не понравился результат, он может

нажать на кнопку “Варианты дополнения”, что отправит GET-запрос к серверу на адрес “/vars”. Ответом сервера будет строка из буфера (рис. 2.4.), состоящая из предыдущего пользовательского текста и одного из неиспользуемых дополнений относительно него.

```

69 @server.route("/vars", methods=["GET"])
70 def answering():
71     if len(global_buff.buff):
72         return global_buff.bring()
73
74     else:
75         return ""
76

```

Рисунок 2.4. Реализация дополнительных ответов генерации текста

Далее идет генерация изображений с рукописным текстом (рис. 2.5). Переменная `new_imgs` – числовая переменная, которая будет выдавать номер для очередного изображения с рукописным шрифтом для каждой сессии. Для улучшения работы с несколькими потоками, предполагается добавление ключа сессии или иного персонального идентификатора пользователя в названия изображений.

```

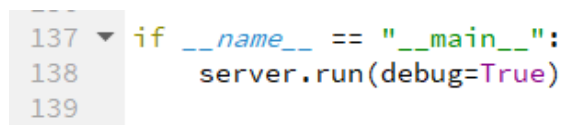
78 @server.route("/get_handwrite", methods=["POST"])
79 def receive_img():
80     global new_imgs
81     req_res = request.get_json()
82     print(req_res)
83
84     t_r = Image.open("Bg_best.png") #продолжить и закончить
85
86     idraw = ImageDraw.Draw(t_r)
87     text = req_res["to_handwrite"]
88     fsize = req_res["f_size"]
89
90     print(text)
91
92     text_f = text
93     text_f_list = []
94     i = 0
95
96     while i < len(text_f):
97         if round(2.5 * i * fsize * 5 * 1.3333333333333333 / 4) > (t_r.width - 15):
98             text_f_list.append(text_f[0:i] )
99             #text_f_list.append()
100
101             text_f_list.append(text_f[i:len(text_f)])
102             text_f = text_f[i:len(text_f)]
103             i = 0
104
105             i += 1
106
107     font = ImageFont.truetype("Gogol.ttf", size=fsize * 5)
108     text_f_list = "\n".join(text_f_list)
109     print(text_f_list)
110
111     #for line in range(len(text_f_list)):
112     idraw.text((10, 20), text_f_list, (0, 0, 0), font=font)
113
114     new_imgs += 1
115     t_r.save(f"result{new_imgs}.png")
116
117     return send_file(f"result{new_imgs}.png", mimetype='image/png')

```

Рисунок 2.5. Функция генерации изображения с текстом

Строка 97 может вызвать непонимание при чтении кода. Однако, секретов тут нет: левая часть логического выражения есть количество пикселей, занимаемых текстом в данном шрифте и размере. Правая – ширина фонового изображения в пикселях. Вся данная конструкция `if` предназначена для перевода строк, в случае, если она не вмещается на всю картинку.

Остальное в функции `receive_img()` – генерация рукописного шрифта и вставка его на фон; после этого сервер сохраняет, затем открывает и отправляет полученную картинку. Стоит не забывать вместе с файлом отправлять соответствующий MIME-type – маркер, обозначающий, какой тип файла отправляется по сети(и не только). В `receive_img()` это `'image/png'`.



```

137 if __name__ == "__main__":
138     server.run(debug=True)
139

```

Рисунок 2.6. Запуск сервера

В целом, сервер готов. Далее требуется его запустить (рис. 2.6.)

2.2.2 Клиент

Клиент написан на языке программирования C++, с помощью фреймворка Qt. Про Qt. Qt — фреймворк для создания веб-приложений на языке программирования C++. Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Начнем с необходимых библиотек (рис. 2.7.):

```

4  #include <QMainWindow>
5  #include <QtNetwork/QNetworkAccessManager>
6  #include <QNetworkRequest>
7  #include <QNetworkReply>
8  #include <QFileDialog>
9  #include <QJsonDocument>
10 #include <QJsonObject>
11 #include <QByteArray>
12 #include <string>
13 #include <QImage>
14 #include <QIcon>

```

Рисунок 2.7. Подключаемые библиотеки

И сразу же перейдем к самому клиенту. Первым делом, обсудим функцию `get_gen_msg()` (рис. 2.8), которая является слотом(одновременно исполняемой функцией) к кнопке “Сгенерировать текст”. Чтобы выполнить запрос, нужно создать менеджера сетевого доступа, или новый экземпляр класса `QNetworkAccessManager`. Теперь через него будет происходить общение между клиентом и сервером.

```

56 void MainWindow::get_gen_msg(){
57     user_text = ui->textEdit->toPlainText();
58     std::string user_text_str = user_text.toLocal8Bit().constData();
59
60     QNetworkAccessManager* manager_poster = new QNetworkAccessManager(this);
61     QUrl send_gen("http://localhost:5000");
62
63     connect(manager_poster, &QNetworkAccessManager::finished, this, &MainWindow::onfinish_post_text);
64
65     QNetworkRequest post_request(send_gen);
66     post_request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json;charset=utf-8");
67
68     qDebug() << user_text;
69
70     auto temp_len{ std::to_string(ui->spinBox->value()) };
71     QString QTemp_len = QString(temp_len.c_str());
72
73     auto user_text_req = "{\"user_text\":\"" + user_text + "\", \"len\":\"" + QTemp_len + "\"}"; // "{\"us
74     QByteArray temp{ user_text_req.toUtf8() };
75     manager_poster->post(post_request, temp);
76
77 }
78

```

Рисунок 2.8. `get_gen_msg()` – генерация запроса к серверу

Далее, при исполнении менеджера сетевого доступа, его функция `finished()` будет приведена в действие. Буфер ответа сервера будет сразу удален, если не успеть его взять во время исполнения `finished()`. Поэтому соединим(`connect`) `manager_poster` и его `finished()` как сигнал((в Qt) первая функция, заставляющая вторую(слот) начать свою работу) и напишем следующий слот (рис. 2.9):

```

30 void MainWindow::onfinish_post_text(QNetworkReply *reply){
31     get_res = reply->readAll();
32     ui->label_4->setText("");
33
34     json_f_post = QJsonDocument::fromJson(get_res);
35
36     QString user_text_fromrep_temp = json_f_post.object().value("result_list").toString();
37
38     qDebug() << user_text_fromrep_temp;
39     qDebug() << user_text;
40
41     if (user_text_fromrep_temp == ""){
42         ui->label_4->setText("Повторите попытку");
43     }
44     //exit(-1234);
45
46     previous_ut = user_text;
47     user_text_fromrep_temp = user_text_fromrep_temp.left(user_text_fromrep_temp.size() - 1);
48     user_text_fromrep_temp = user_text_fromrep_temp.right(user_text_fromrep_temp.size() - 1);
49
50     user_text += user_text_fromrep_temp;
51     ui->textEdit->setPlainText(user_text);
52
53     reply->deleteLater();
54 }
55

```

Рисунок 2.9. Обработка ответа сервера на запрос.

Здесь можно спокойно распарсить все нужные JSON файлы и отформатировать строки, полученные от сервера, не опасаясь, что ответ сервера будет удален из буфера прежде времени. К слову: принципиально GET и POST запросы в `QNetworkRequest` отличаются лишь набором передаваемых переменных. И `.setHeader()` необходим для установки хедера `'application/json'`, единственный принимаемый сервером Flask формат данных.

Теперь обратим внимание на расчет текущего состояния текста пользователя (рис. 2.10.). Это нужно для удобства пользователя – например, если ему не хочется превышать какую-то грань объемов текста, он сможет сразу увидеть интересующую его информацию в данном смысле.

```

79 void MainWindow::count_symb(){
80     QString s = ui->textEdit->toPlainText();
81     s = s.toLocal8Bit().constData();
82     auto t = std::to_string(s.size());
83
84     s = t.c_str();
85     //qDebug() << s;
86     ui->label_Symb->setText(s);
87 }
88
89 void MainWindow::count_word(){
90     QString s = ui->textEdit->toPlainText();
91     auto r{ s.split(" ") };
92     int t{ 0 };
93
94     for (auto i : r){
95         if (i != ""){
96             t++;
97         }
98     }
99
100     auto st = std::to_string(t);
101     s = st.c_str();
102     //qDebug() << s;
103     ui->label_Word->setText(s);
104 }
105
106 void MainWindow::count_sent(){
107     QString s = ui->textEdit->toPlainText();
108     auto r1{ s.split(".") };
109     int t{ 0 };
110
111     for (auto i : r1){
112         for (auto j : i.split("!")){
113             for (auto k : j.split("?")){
114                 if (k != ""){
115                     t++;
116                 }
117             }
118         }
119     }

```

Рисунок 2.10. Расчет текущего состояния текста в левом TextEdit

Далее, для отправки на сервер текста и параметры, нужные для трансформации в рукописный на определенном фоне, была написана функция-слот `get_gen_image()` (рис. 2.11.), которая, аналогично связи изображенной на рис. 2.8. с рис. 2.9., связывает через connect слот из рис. 2.12.. В этом «подслоте» парсится изображение по MIME-type, а затем записывается в общую переменную `QImage user_pic`.


```

211 void MainWindow::get_gen_image(){
212     QNetworkAccessManager* manager_poster = new QNetworkAccessManager(this);
213     QUrl send_gen("http://localhost:5000/get_handwrite");
214
215     connect(manager_poster, &QNetworkAccessManager::finished, this, &MainWindow::onfinish_get_image);
216
217     QNetworkRequest post_request(send_gen);
218     post_request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json;charset=utf-8");
219
220     qDebug() << ui->textEdit_2->toPlainText();
221
222     auto temp_fsize{ std::to_string(ui->spinBox_2->value()) };
223     QString QTemp_fsize = QString(temp_fsize.c_str());
224
225     auto user_text_req = "{\"to_handwrite\":\"\" + ui->textEdit_2->toPlainText() + "\", \"f_size\" : \" + QTemp_fsize + "\"}";
226     QByteArray temp{ user_text_req.toUtf8() };
227     manager_poster->post(post_request, temp);
228     ui->label_4->setText("Ошибка. Попробуйте позже.");
229 }
230

```

Рисунок 2.11. Отправка запроса на получение изображения с рукописным шрифтом

```

192 void MainWindow::onfinish_get_image(QNetworkReply* reply){
193     get_res = reply->readAll();
194
195     if (get_res == ""){
196         ui->label_4->setText("Повторите попытку");
197         return;
198     }
199
200     ui->label_4->setText("");
201     user_pic.loadFromData(get_res, "PNG");
202
203     qDebug() << user_pic.size();
204
205     QPixmap pixmap = QPixmap::fromImage(user_pic);
206     ui->label_image->setPixmap(pixmap);
207
208     reply->deleteLater();
209 }

```

Рисунок 2.12. Обработка ответа сервера

Для сохранения изображения, используется стандартное в Qt средство `QFileDialog` (рис. 2.13.):

```

231 void MainWindow::save_img(){
232     QString fileName = QFileDialog::getSaveFileName(this, "Save Image", "", "Допустимые форматы (*.png)");
233     if (fileName.size()){
234         user_pic.save(fileName);
235     }
236
237     else{
238         ui->label_4->setText("Страница не сохранена");
239     }
240 }

```

Рисунок 2.13. Сохранение изображения

Внешний вид проделанной над клиентом работы вы можете наблюдать на рис. 2.14:

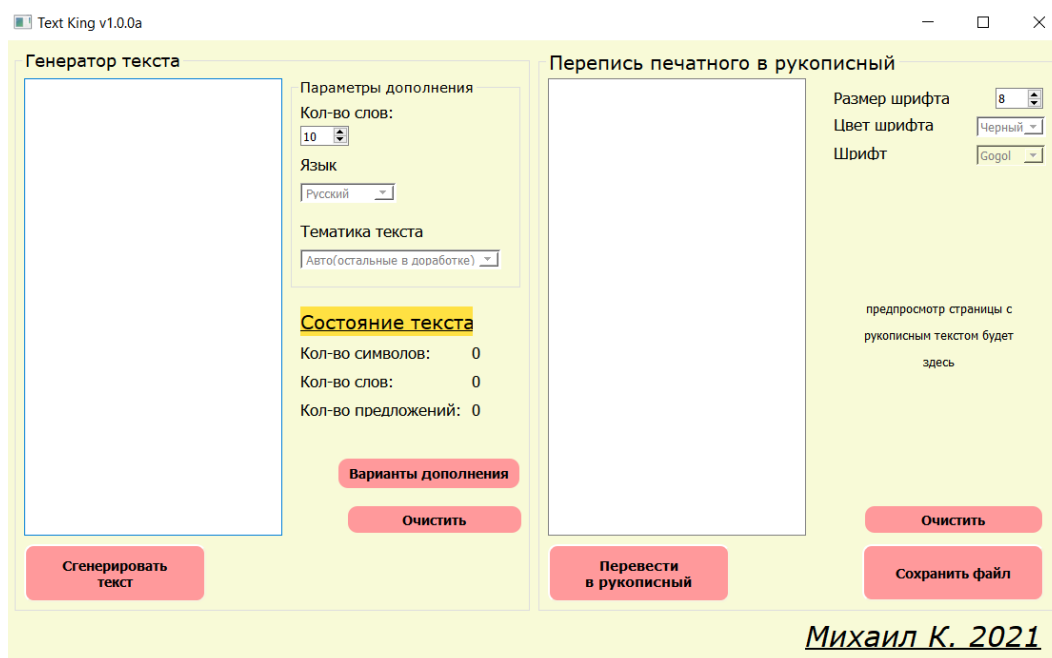


Рисунок 2.14. Итоговый внешний вид приложения

ГЛАВА 3

ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Тестирование исходного кода

Мой исходный код был протестирован Ильясовым Эмиром из группы ПИ-201. В результате были выявлены следующие ошибки:

Клиент

Недоработки:

1. "mainwindow.cpp", строки 108-117: нерациональный подсчет предложений
2. Очень много бесполезных переменных, некоторые из них не удаляются и остаются в памяти
3. Две функции clear_text_1, clear_text_2, делают практически одно и то же.
4. Нерационально сохраняется получаемая информация.
5. Некоторые combobox неактивны

Ошибки:

1. "mainwindow.cpp", строка 204 уходит пустая картинка в предпросмотр.

Недочеты:

1. Много мест, где применяется автотип "auto"; затрудняет читаемость кода.
2. Много комментариев, неиспользуемая часть кода (118-137), аналогично.
3. В (44-45) очень странный парсинг JSONa
4. В "Client_cpp.pro" в программу включаются несуществующие и ненужные заголовочные файлы
5. JSON объект отсылается посредством форматирования, предпочтительнее отсылать с помощью функций
6. Много отладочной печати, для релиза программа не подходит.

Сервер (main.py)

Недоработки:

1. Импортируется ненужная библиотека io

2. Класс `Buffer_msg` по сути является немного видоизмененным списком, нужно было сделать явное наследование.

3. Также много неиспользуемых переменных

4. Недоделан переход страниц, расширение букв и многие `receive_img()`

Ошибки:

1. Изображение стирается в строчке 97: было записано в буффер, перезаписано на полностью белый.

2. Нигде не указан формат отправляемой картинки: чревато многими багами

3. Нерабочий перевод строк рукописного текста посимвольно: строка 93

Недочеты:

1. Много кодо-комментариев, неиспользуемая часть кода (106-122).

2. Много отладочной печати

3. Некоторые функции можно было объединить в одну

Некоторые ошибки были исправлены в ходе работы после тестирования.

Вывод тестирования

В целом код написан грамотно и структурированно, с небольшими ошибками. В условиях явных ошибок не выявлено.

Стоит обратить внимание на переменные, которые можно сделать константами, но таковыми не являются.

Это повысит надежность приложения и уменьшит количество потенциальных сбоев.

Возможно мной были найдены не все такие переменные.

Также следует обратить внимание на незаконченные механики: либо закончить, либо исключить вариант неправильного использования.

3.2 Тестирование интерфейса пользователя

Данный проект был протестирован также Ильясовым Эмиром по написанному мной же тест-плану, все тесты кроме одного были успешно пройдены. Это стало доступно благодаря последовательному рефакторингу кода.

Тест 1/Test Name 1

Среда тестирования /Environment		QT Creator			
Предварительные действия /Pre Requisites		Запуск среды разработки QT Creator			
Комментарии /Comments		-			

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Отладка и запуск программы	Программа запускается, выставляются изначальные настройки spinbox и combobox ов, написанные в техническом задании. Корректно отображаются все кнопки и окна на экране.	Реальный результат совпадает с ожидаемым	Пройден	
2	Нажатие на кнопку "Выход"	Окно программы закрывается.	Реальный результат совпадает с ожидаемым	Пройден	

Рисунок 3.1. Тест-кейс 1

Тест 2/Test Name 2

Среда тестирования /Environment		QT Creator			
Предварительные действия /Pre Requisites		Запуск среды разработки QT Creator, отладка и запуск программы, включить сервер			
Комментарии /Comments		-			

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Набор слов в левое поле для ввода текста. Выбрать желаемые параметры.	При нажатии на кнопку «очистить» будет удален весь пользовательский текст. Должны меняться значения количества предложений, слов и символов.	Реальный результат совпадает с ожидаемым	Пройден	

Рисунок 3.2. Тест-кейс 2

Тест 3/Test Name 3

Среда тестирования /Environment		QT Creator			
Предварительные действия /Pre Requisites		Запуск среды разработки QT Creator, отладка и запуск программы, включить сервер			
Комментарии /Comments		-			

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Выбрать нужные параметры	Будут заданы параметры для дополнения текста на следующие генерации	Реальный результат совпадает с ожидаемым	Пройден	
2	Ввести свое предложение	Будет введен текст пользователя	Реальный результат совпадает с ожидаемым	Пройден	
3	Нажать на «сгенерировать текст»	Будет отправлен запрос API на сервер и дополнен пользовательский текст	Реальный результат совпадает с ожидаемым	Пройден	Не работает, если отправить пустую строку

Рисунок 3.3. Тест-кейс 3

Тест 4/Test Name 4

Среда тестирования /Environment	QT Creator
Предварительные действия /Pre Requisites	Запуск среды разработки QT Creator, отладка и запуск программы
Комментарии /Comments	-

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Нажатие на всевозможные кнопки в левой части окна проекта с целью выяснения их работоспособности.	Все кнопки функционируют и выполняют свою роль.	Реальный результат совпадает с ожидаемым	Пройден	Два combobox неактивны

Рисунок 3.4. Тест-кейс 4

Тест 5/Test Name 5

Среда тестирования /Environment	QT Creator
Предварительные действия /Pre Requisites	Запуск среды разработки QT Creator, отладка и запуск программы, включенный сервер
Комментарии /Comments	-

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Ввести текст в правое поле	Появится текст пользователя	Реальный результат совпадает с ожидаемым	Пройден	
2	Выбрать параметры	Будут заданы параметры для дополнения текста на следующие генерации	Реальный результат совпадает с ожидаемым	Пройден	
3	Нажать «перевести в рукописный текст»	Будет отправлен запрос на сервер	Реальный результат совпадает с ожидаемым	Пройден	
4	Получить картинку в предпросмотре	Будет отображена картинка в уменьшенном разрешении справа от поля ввода текста	Ничего не отображается	Не пройден	

Рисунок 3.5. Тест-кейс 5

Тест 6/Test Name 6

Среда тестирования /Environment	QT Creator
Предварительные действия /Pre Requisites	Запуск среды разработки QT Creator, отладка и запуск программы, выключенный сервер
Комментарии /Comments	-

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Отправить текст для дополнения при выключенном сервере	Строка состояния отобразит «Повторите попытку»	Реальный результат совпадает с ожидаемым	Пройден	
2	Отправить текст для перевода в рукописный при выключенном сервере	Строка состояния отобразит «Ошибка. Повторите попытку»	Реальный результат совпадает с ожидаемым	Пройден	
3	Отправить текст для перевода в рукописный при ошибках	Строка состояния отобразит «Ошибка. Повторите попытку»	Реальный результат совпадает с ожидаемым	Пройден	Не зависит от сервера

Рисунок 3.6. Тест-кейс 6

ГЛАВА 4

ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

4.1 Перспективы технического развития

В ходе разработки приложения были выполнены все поставленные задачи и реализованы многие запланированные на этапе разработки ТЗ функции. Однако программа все еще в очень сыром состоянии. Нужно отладить и исправить некоторые ошибки, добавить режим полной генерации сочинений, синонимайзер, саммарайзер по заданному тексту.

Это перспективы развития основного функционала программы, но и сам интерфейс может потребовать переработки.

Также, стоит добавить:

1. Несколько вариантов генерации рукописного шрифта;
2. Разнообразие способов дополнения текста;
3. Новые человеческие языки.

Также стоит рассмотреть возможности улучшения диалога с пользователем, добавление «быстрого старта» и другие бьютификаторы ГУИ. Стоит также и заняться рефакторингом уже написанного кода.

4.2 Перспективы монетизации

На данный момент приложение является бесплатным, open-сорс.

Также, TextKing пока не дает слишком широкого спектра возможностей. Однако, при улучшениях из 4.1, шансы на монетизацию появятся.

Идея монетизации: добавить премиум-версию программы, в которой будет присутствовать нулевой трафик и расширенный функционал, и, возможно, тех-поддержка.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была достигнута поставленная цель: написано приложение для дополнения и генерации сочинений «TextKing», функционал которого соответствует заявленному в ТЗ.

В ходе проделанной работы были приобретены полезные навыки девелопмента, такие, как:

1. Опыт работы с Python Flask, создание сервера с помощью этого фреймворка;
2. Опыт самообучения различным фреймворкам с поиском соответствующей документации и прочих материалов;
3. Знания о submodule C++ Qt;
4. Опыт работы с IDE Qt Creator, необходимый для создания приложений с фреймворком C++ Qt и его реализациями на других языках;
5. Основы работы с API нейросетей;
6. Девелопмент приложения с популярной сетевой структурой: клиент-серверной. Обеспечение контакта интерфейсов библиотек и частей приложения.
7. Параллельная разработка сервера и клиента.

ЛИТЕРАТУРА

1. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.
2. ГОСТ 19.003-80 Flask Web Development [Текст] – Введ. с 01.07. 1981 г. М.: Miguel Grinberg, 2018. – 456 с.
3. ГОСТ 19.002-80 М. Шлее Qt5.10. Профессиональное программирование на C++ [Текст] – Введ. с 01.05. 2018 г. М.: СПб VNP, 1981. – 1072 с.
4. “Мега-Учебник Flask” – <https://habr.com/ru/post/193242/>
5. C++ Data Structures and Algorithm Design Principles - Authors: Payas Rajan, John Carey, Et al, Published: October 31, 2019, ISBN: 978-1838828844