ОГЛАВЛЕНИЕ

[1.Введение…………………………………………………………………….](https://www.ronl.ru/knigi/informatika/784944/#_Toc193853850)

1.1 Цель работы………………………………………………………3

1.2 Описание приложения…………………………………………...4

2. Тестирование……………………………………………………………...

2.1 Вид приложения…………………………………………………..5

2.2 Тестирование приложения……………………………………….6

2.3 Внедрение приложения в платформу…………………………..16

Заключение…………………………………………………………………25

Приложение………………………………………………………………...26

1.Введение

**1.1 Цель работы**

Разработка приложения на C++ по созданию бота для платформы Discord посредствам Discord API .

**Требования к приложению**

Данное приложение создается исключительно для ПК и должно иметь такой функционал:

1. В главном окне присутствует переход к информации об авторах.
2. В главном окне есть кнопки выхода и скрытия приложения.
3. В главном окне есть кнопка перехода ко 2 окну по созданию бота.
4. В главном окне есть возможность перевести всю информацию на русский язык.
5. В окне по созданию бота присутствуют кнопки возврата к главному окну, скрытие приложения, закрытие приложения.
6. В окне по созданию бота есть возможность перевести всю информацию на русский язык.
7. В окне по созданию бота присутствуют все необходимые кнопки по добавлению команд для бота.
8. Рамка приложения собственно написанная.

**1.2 Описание приложения**

Данное приложение было создано на языке C++ с использованием кроссплатформенной среды разработки Qt Creator на операционной системе Windows 10. С помощью данного приложения пользователь может создать собственного бота для платформы Discord.

**Формы:**

1. Mainwindow.ui – главное окно приложения.
2. Info.ui – окно с информацией об авторах.
3. Create\_bot\_window.ui – окно по созданию бота.

**Заголовочные файлы:**

1. Bot.h – класс бота.
2. Create\_bot\_window.h – файл с прототипами функций и сигналов окна создания бота.
3. Creator.h – класс реализующий генерацию бота .
4. Info.h – файл с прототипами функций и сигналов окна с информацией.
5. Mainwindow.h – файл с прототипами функций и сигналов главного окна.

**Исходники:**

1. Bot.cpp – класс бота.
2. Create\_bot\_window.cpp – файл с функциями и сигналами окна создания бота.
3. Creator.cpp – класс реализующий генерацию бота.
4. Info.cpp – файл с функциями и сигналами окна с информацией.
5. Mainwindow.cpp – файл с функциями и сигналами главного окна.
6. Main.cpp – главный файл.

2. Тестирование

**2.1 Вид приложения**

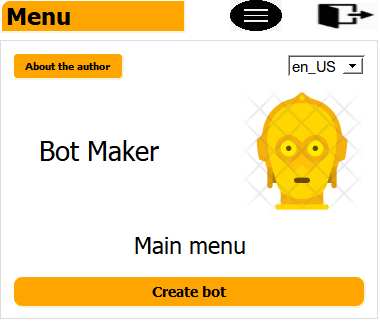
****

Рис. 1 Главное окно

****

Рис. 2 Окно по созданию бота

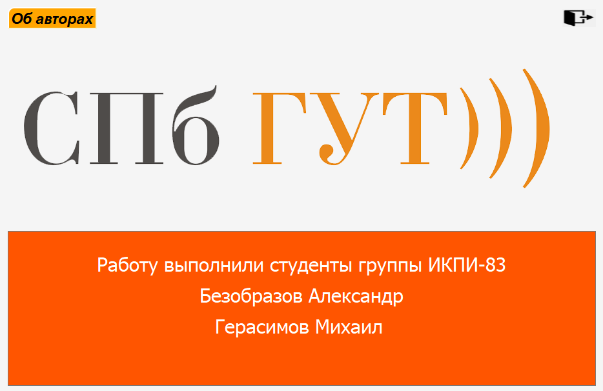
****

Рис. 3 Информация об авторах

**2.2 Тестирование приложения**

**Главное окно**

На данной панели расположены кнопки скрытия приложения и закрытия приложения.

****

Рис. 4 Кнопки скрытия и закрытия приложения

Реализация

1. **void** MainWindow::on\_pushButton\_2\_clicked()
2. {
3. close();
4. }
6. **void** MainWindow::on\_pushButton\_clicked()
7. {
8. showMinimized ();
10. }

На данной панели расположены кнопки перехода к окну информации об авторах и смене языка.

Рис. 5 Кнопки перехода к окну с информацией об авторах и смене языка



Реализация кнопки перехода к окну с информацией об авторах

1. **void** MainWindow::on\_info\_clicked()
2. {
3. Info ->show();
5. }

Реализация кнопки смена языка

Создаем в папке с проектом файл для смены языка с расширением qm.

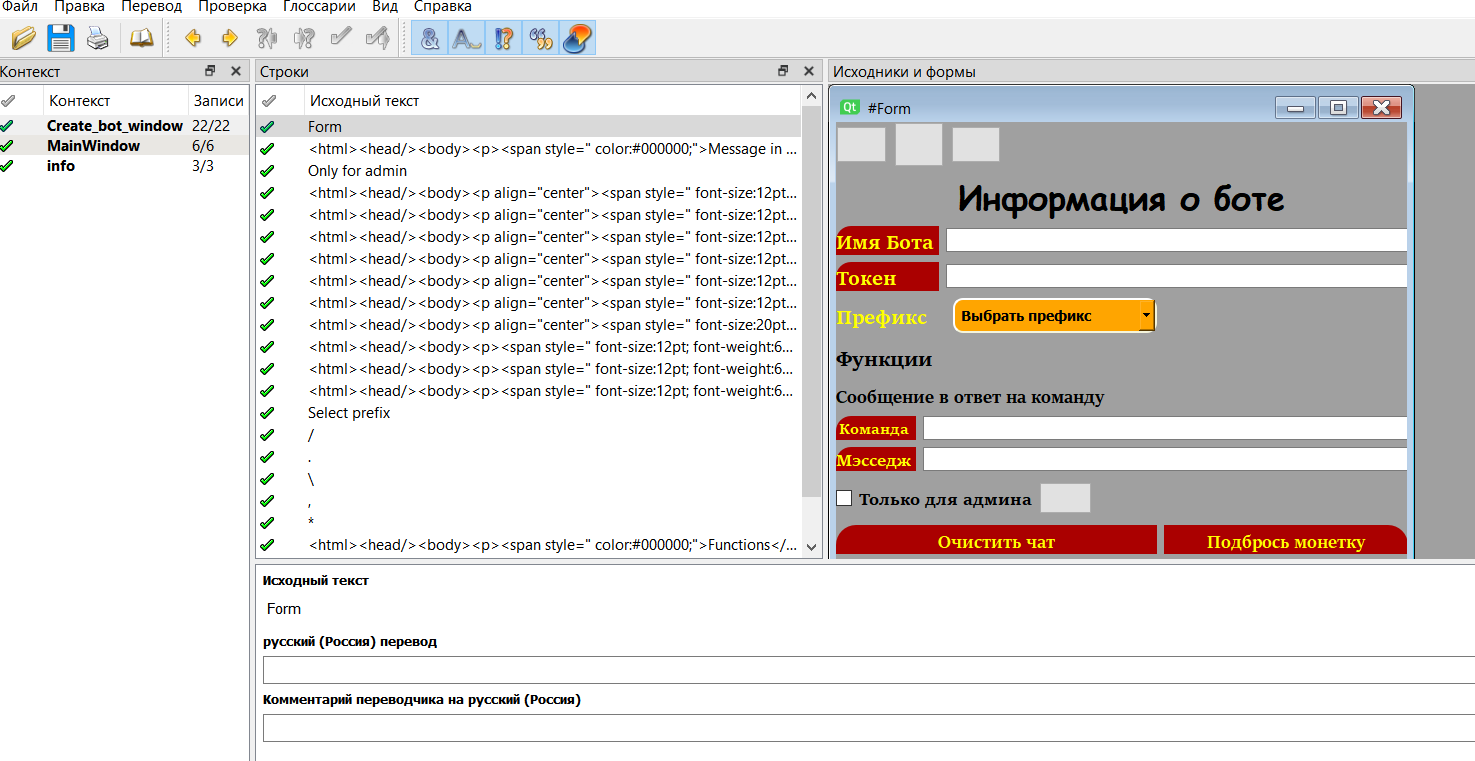
Далее открываем программу QtLinguist, после того как в QtCreator будут прописаны команды для перевода. Вставляем наш qm файл в QtLinguist и переводим в ручную все что нам необходимо.

Рис. 6 Работа с переводом в QtLinguist

Переходим в файл Bot.pro, указываем имя файла перевода в профайле проекта: TRANSLATIONS = QtLanguage\_ru.ts

Переходим в заголовочный файл mainwindow.h и подключаем библиотеку <QTranslator>. В заголовочном файле главного окна приложения необходимо объявить объекта класса **QTranslator** и переопределить метод **changeEvent(QEvent \*event)** , в котором будет определяться событие смены языка приложения.

1. **protected**:
2. // Метод получения событий в главном окне приложения
3. // В нём будет производиться проверка события смены перевода приложения
4. **void** changeEvent(QEvent \* event) override;
6. **private**:
7. Ui::MainWindow \*ui;
8. QTranslator qtLanguageTranslator;   // Выделяем перевод в отдельном поле, иначе ничего работать не будет

Переходим в исходный файл mainwindow.cpp и подключаем библиотеку <QTranslator>.Далее пишем данный код:

1. // Задаём два пункта с текстом локалей в комбобоксе
2. ui->comboBox->addItems(QStringList() << "ru\_RU" << "en\_US");
4. // подключаем к сигналу изменения пункта комбобокса лямбда функцию,
5. // в которой будет изменяться перевод приложения
6. // Здесь имеется интересный момент. Поскольку QComboBox имеет перегрузку сигнатуры сигнала,
7. // то нам необходимо скастовать сигнал к нужной сигнатуре.
8. // В данном случае будем использовать название пункта при его изменении
9. connect(ui->comboBox, **static\_cast**<**void** (QComboBox::\*)(**const** QString &)>(&QComboBox::currentIndexChanged),
10. [=](**const** QString &str){
11. qtLanguageTranslator.load("QtLanguage\_" + str, ".");   // Загружаем перевод
12. qApp->installTranslator(&qtLanguageTranslator);        // Устанавливаем перевод в приложение
13. });
15. // Сделаем первоначальную инициализацию перевода для окна прилоежния
16. qtLanguageTranslator.load(QString("QtLanguage\_") + QString("ru\_RU"));
17. qApp->installTranslator(&qtLanguageTranslator);
18. **void** MainWindow::changeEvent(QEvent \*event)
19. {
20. // В случае получения события изменения языка приложения
21. **if** (event->type() == QEvent::LanguageChange) {
22. ui->retranslateUi(**this**);    // переведём окно заново
23. }
24. }

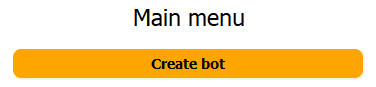
На данной панели расположена кнопка перехода к окну по созданию бота.

Рис. 7 Кнопка перехода к окну по созданию бота

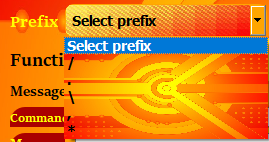
Реализация

1. **void** MainWindow::on\_CreateBotButton\_clicked()
2. {
3. CreateWindow1->show();
4. **this**->close();
5. }

**Окно по созданию бота**



Поля имя и токен бота. Токен связывает код бота с приложением зарегистрированным на Discord Developer Portal. Имя бота придумывает пользователь.



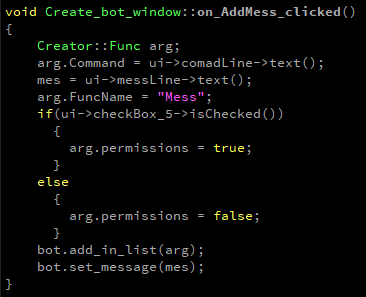
ComboBox содержащий в себе символы которые можно использовать в качестве префикса. Префикс – символ сообщающий боту, что следующие символы являются командой.

Функция ответа на выбранную пользователем команду.



В поле Command пользователь вводит команду (без префикса), В поле Message вводит ответ который должен последовать за ботом. CheckBox Only for admin позволяет сделать функцию доступной для использования только администратору Discord сервера. Кнопка “+” добавляет эту функцию в бота.

Реализация C++



Реализация ,как функция бота Python

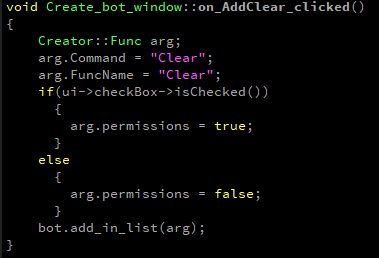


Функция очистки чата.



При вызове команды clear чат Discord канала очищается. CheckBox Only for admin позволяет сделать функцию доступной для использования только администратору Discord сервера. Кнопка “+” добавляет эту функцию в бота.

Реализация C++



Реализация, как функция бота Python

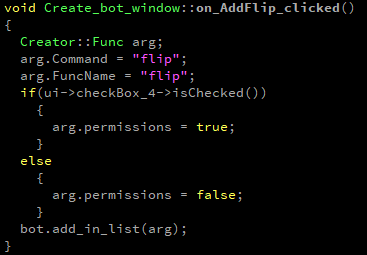


Функция “монетка”

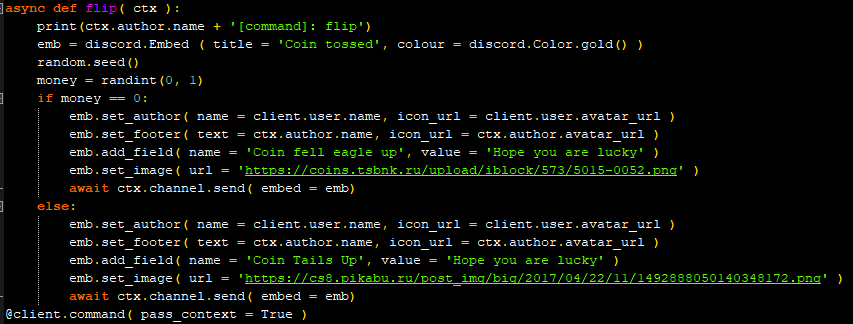


При вызове команды flip в чат Discord канала бот возвращает “Орел” или “Решка”. CheckBox Only for admin позволяет сделать функцию доступной для использования только администратору Discord сервера. Кнопка “+” добавляет эту функцию в бота.

Реализация C++



Реализация, как функция бота Python

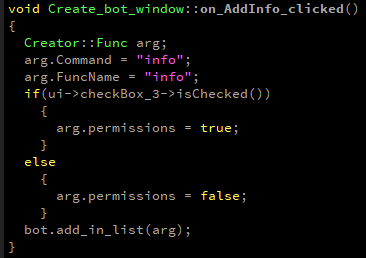


Функция “Обо мне”

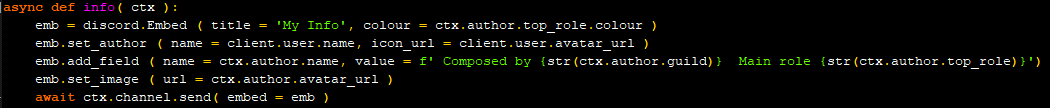


При вызове команды info в чат Discord канала бот возвращает информацию о пользователе, который вызвал эту команду. CheckBox Only for admin позволяет сделать функцию доступной для использования только администратору Discord сервера. Кнопка “+” добавляет эту функцию в бота.

Реализация C++



Реализация, как функция бота Python

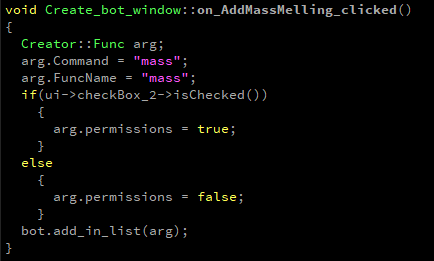


Функция массовой рассылки.

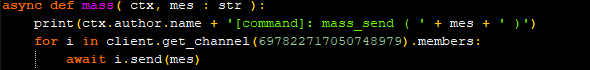


При вызове команды mass и сообщения бот рассылает это сообщение всем участникам Discord сервера. CheckBox Only for admin позволяет сделать функцию доступной для использования только администратору Discord сервера. Кнопка “+” добавляет эту функцию в бота.

Реализация C++



Реализация, как функция бота Python



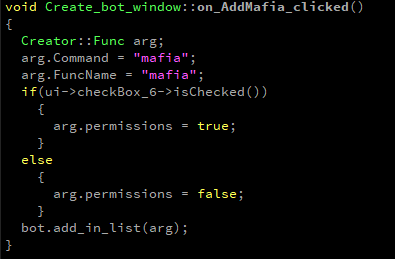
Функция “игра Мафия”



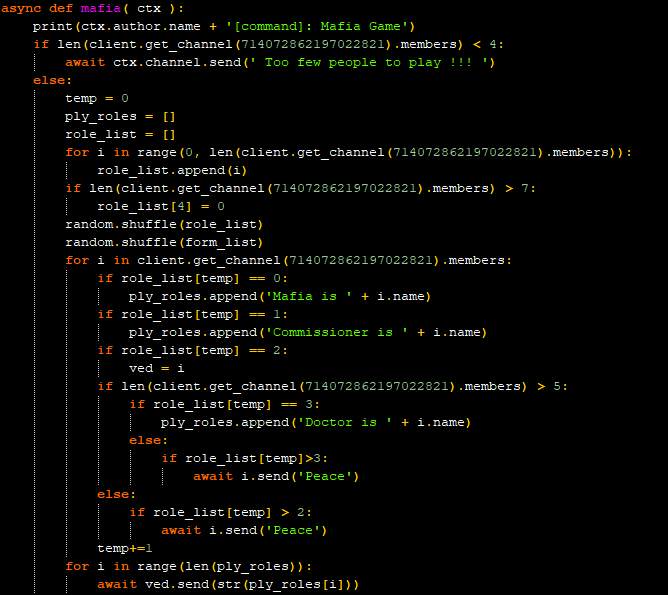
При вызове команды mafia бот рассылает роли для игры в мафию участникам голосового канала(так же высылается роль ведущего, ему будет известно у кого какая роль). CheckBox Only for admin позволяет сделать функцию доступной для использования только администратору Discord сервера. Кнопка “+” добавляет эту функцию в бота.

\*id канала отведенного для игры требуется вставить в файл бота в 58 строку вместо “**714072862197022821**”

Реализация C++



Реализация, как функция бота Python

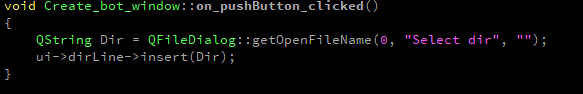


Поле выбора директории



При нажатии на кнопу “шестеренка” открывается окно выбора \*.py файла куда будет записан код бота.

Реализация C++

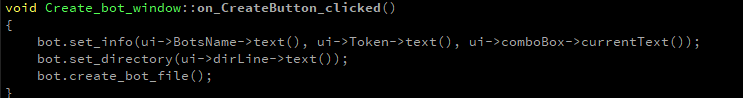


Кнопка Create



Генерирует код бота с выбранными функциями.

Реализация C++

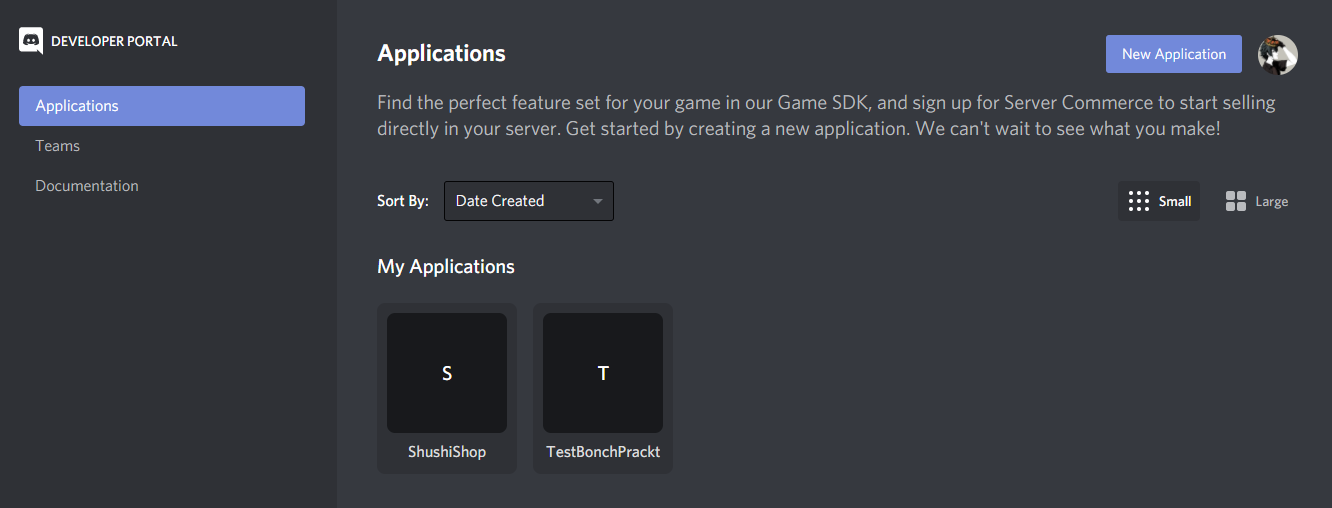


Так же в боте по умолчанию предусмотрена функция отслеживания ошибок при наборе команды.

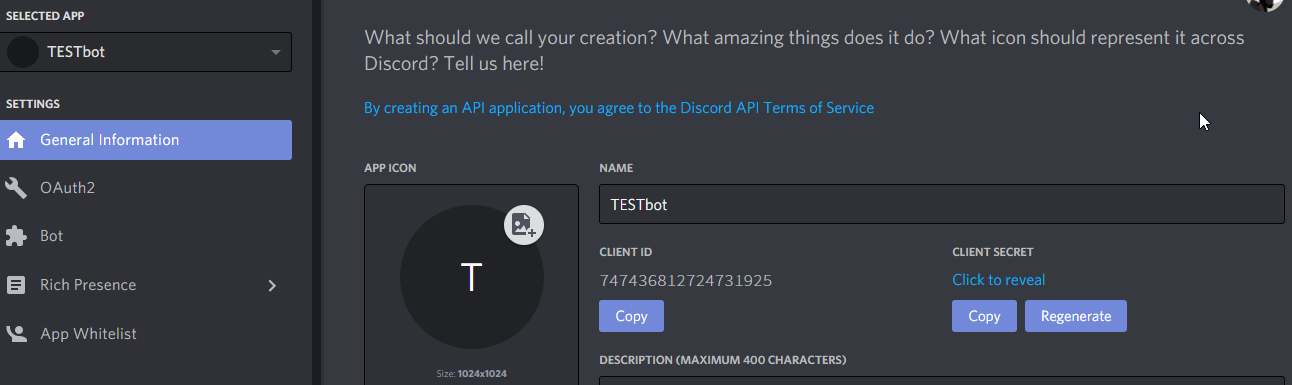
**2.3.Внедрение приложения в платформу**

Пример создания бота при помощи программы.

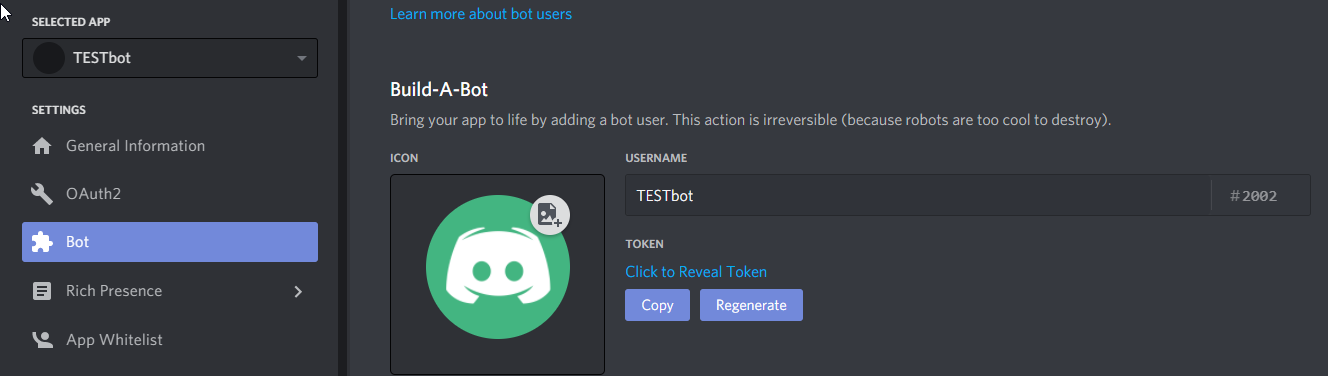
Для начала переходим на сайт <https://discord.com/developers/applications> и создаем новое приложение (New Applications) .



После ввода имени приложения переходим во вкладку Bot.

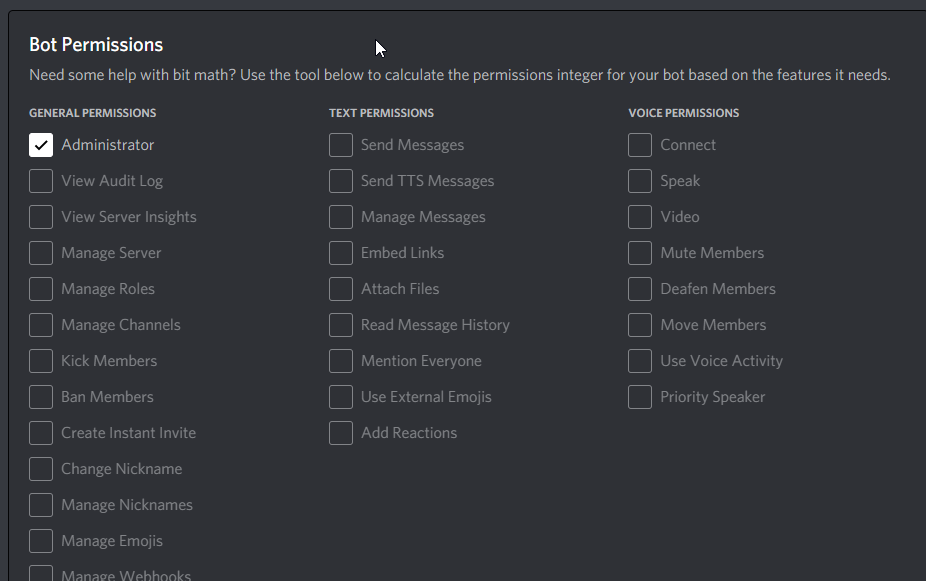


И нажимаем Add bot.

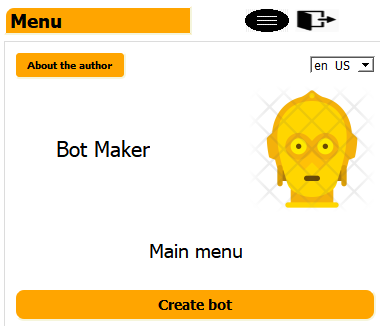


В пункте TOKEN нажимаем Copy.

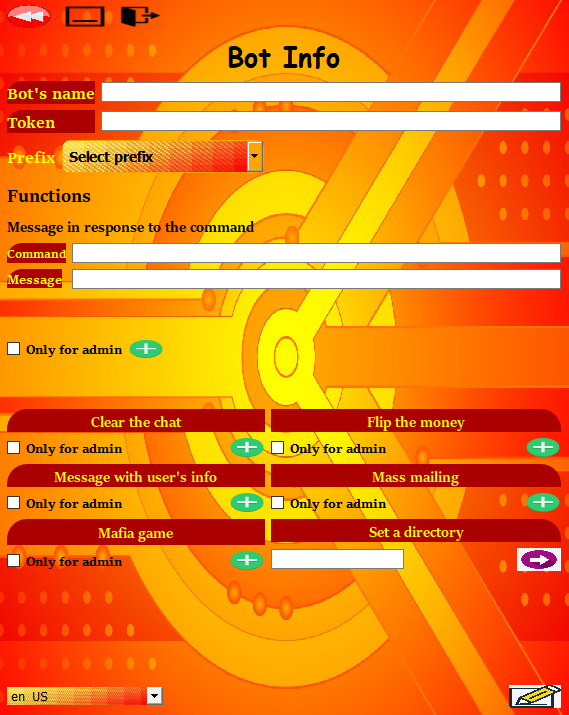
Внизу страницы выбираем пункт Admin.



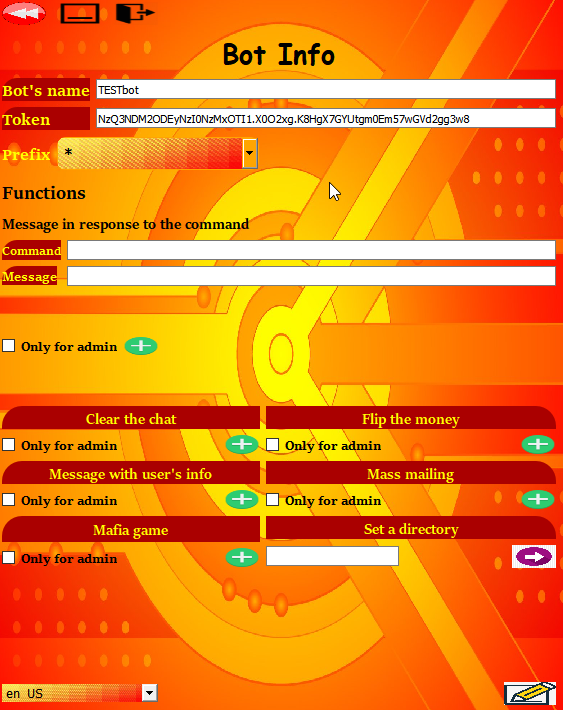
Теперь переходим в программу.



Нажимаем Create bot.

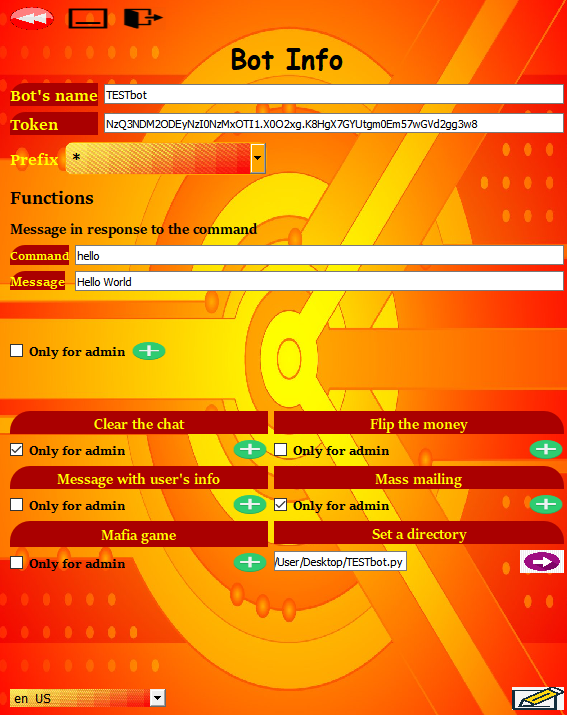


В пункт имя вводим имя бота , а в поле токен вставляем токен который мы скопировали на сайте. И так же выбираем префикс.



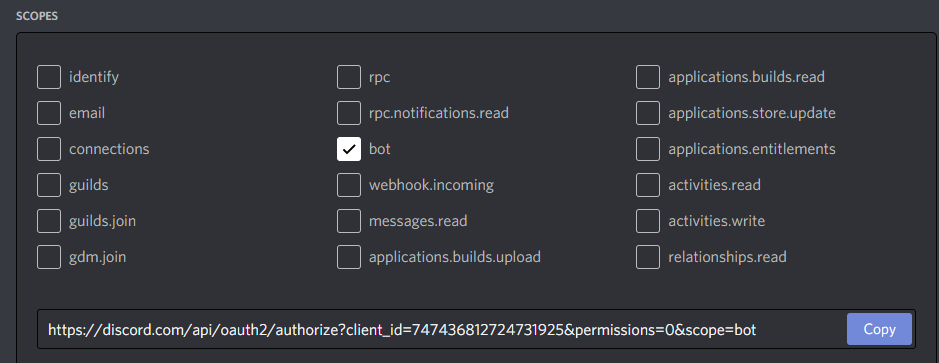
Теперь добавим функции. Начнем с функции ответа на сообщение. Пусть при отправке сообщения hello бот будет отвечать Hello world. Вводим эти сообщения в поля и нажимаем “+”.

Теперь добавим остальные функции ,но функции очистки чата и массовой рассылки разрешим только для администратора. И выберем файл бота.

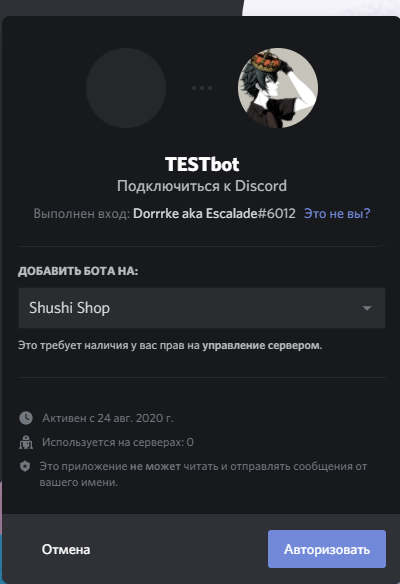


Теперь нажимаем на кнопку Create.

Теперь осталось добавить бота в канал. Для этого возвращаемся на сайт где мы его создавали и переходим в пункт OAuth2 и снизу страницы в таблице scopes ставим галочку напротив bot.

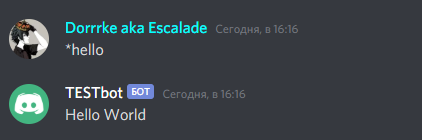


Переходим по ссылке которая появилась и добавляем бота к каналу.

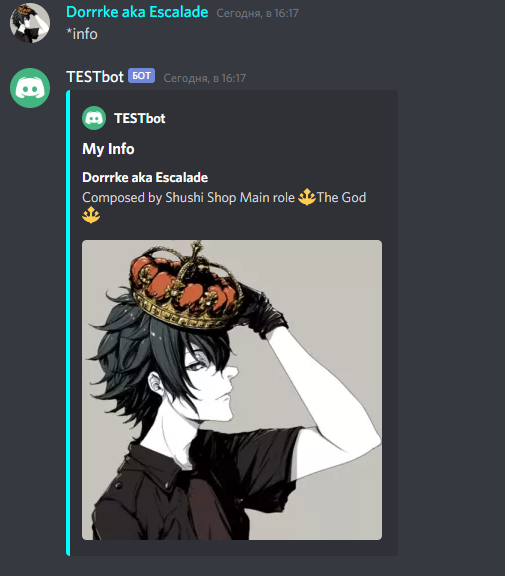


Теперь запускаем нашего бота через py файл и тестируем в канале.

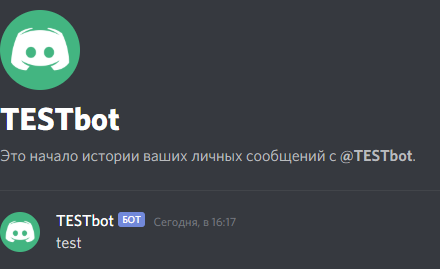
Проверим все функции по очереди.

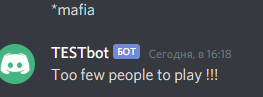




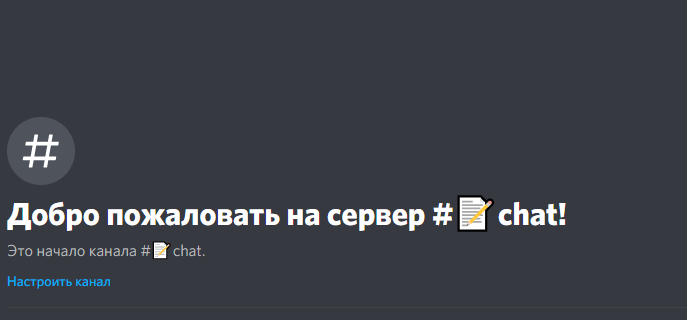






Команда \*Clear



**Заключение**

* 1. Приложение было написано в кроссплатформенной среде разработки Qt Creator на языке C++, позволяющее создать бота для платформы discord.
  2. Была изучена библиотека discord api python для описания функций и задач, которые должен выполнять бот.
  3. Улучшены навыки в кроссплатформенной среде разработки Qt Creator.
  4. Приобрели опыт работы с discord api python.

**Приложение**

**Bot.h**

1. #pragma once
2. #include <iostream>
3. #include<QString>
4. **using** **namespace** std;
6. **class** Bot
7. {
8. **private**:
9. QString Name;  //Bot's name
10. QString Token; //Bot's token
11. QString Prefix;  //Bot's commands prefix
12. **public**:
13. Bot();
14. **void** set\_name(QString name);      //Set bot's name
15. **void** set\_token(QString token);    //Set bot's token
16. **void** set\_prefix(QString prefix);    //Set bot's commands prefix
17. QString get\_name();               //Return bot's name
18. QString get\_token();              //Return bot's token
19. QString get\_prefix();               //Return bot's commands prefix
20. };

**Create\_bot\_window.h**

1. #ifndef CREATE\_BOT\_WINDOW\_H
2. #define CREATE\_BOT\_WINDOW\_H
4. #include <QWidget>
5. #include<QFileDialog>
6. #include<creator.h>
7. #include <QTranslator>
8. #include <QEvent>
10. **namespace** Ui {
11. **class** Create\_bot\_window;
12. }
14. **class** Create\_bot\_window : **public** QWidget
15. {
16. Q\_OBJECT
18. **public**:
19. **explicit** Create\_bot\_window(QWidget \*parent = nullptr);
20. ~Create\_bot\_window();
21. signals:
22. **void** menuW();

25. **private** slots:

28. **void** mouseMoveEvent(QMouseEvent \*);
29. **void** mousePressEvent(QMouseEvent \*);
30. **void** mouseReleaseEvent(QMouseEvent \*);

33. **void** on\_CreateBotButton\_clicked();
35. **void** on\_AddMess\_clicked();
37. **void** on\_comadLine\_textEdited(**const** QString &arg1);
39. **void** on\_messLine\_textEdited(**const** QString &arg1);
41. **void** on\_checkBox\_5\_stateChanged(**int** arg1);
43. **void** on\_AddClear\_clicked();
45. **void** on\_checkBox\_4\_stateChanged(**int** arg1);
47. **void** on\_AddFlip\_clicked();
49. **void** on\_checkBox\_3\_stateChanged(**int** arg1);
51. **void** on\_AddInfo\_clicked();
53. **void** on\_checkBox\_2\_stateChanged(**int** arg1);
55. **void** on\_AddMassMelling\_clicked();
57. **void** on\_checkBox\_6\_stateChanged(**int** arg1);
59. **void** on\_AddMafia\_clicked();


63. **void** on\_pushButton\_clicked();
65. **void** on\_CreateButton\_clicked();
67. **void** on\_pushButton\_2\_clicked();


71. **void** on\_Back\_clicked();
72. **void** on\_pushButton\_3\_clicked();
74. **protected**:
75. // Метод получения событий в главном окне приложения
76. // В нём будет производиться проверка события смены перевода приложения
77. **void** changeEvent(QEvent \* event) override;
78. **private**:
79. Ui::Create\_bot\_window \*ui;
80. QTranslator qtLanguageTranslator1;   // Выделяем перевод в отдельном поле, иначе ничего работать не будет
81. QPoint myPos1;
82. Creator bot;
83. **bool** mes\_permiss;
84. **bool** clear\_permiss;
85. **bool** flip\_permiss;
86. **bool** info\_permiss;
87. **bool** melling\_permiss;
88. **bool** game\_permiss;
89. QString mes, com;
91. };
93. #endif // CREATE\_BOT\_WINDOW\_H

**Creator.h**

1. #pragma once
2. #include<iostream>
3. #include<Bot.h>
4. #include<vector>
5. #include<QString>
6. #include<QFile>
7. **using** **namespace** std;
9. **class** Creator
10. {
11. **public**:
12. Creator()=**default**;
13. Creator(**const** QString Name, **const** QString token, QString prefix)
14. {
15. bot.set\_name(Name);
16. bot.set\_token(token);
17. bot.set\_prefix(prefix);
18. }
19. **struct** Func
20. {
21. QString FuncName;
22. QString Command;
23. **bool** permissions;
24. };
25. **void** set\_info(**const** QString Name, **const** QString token, QString prefix);
26. **void** set\_directory(QString dir);
27. **void** set\_func\_list(vector<Func> List);
28. **void** add\_in\_list(Func arg);
29. **void** set\_message(QString mess);
30. QString get\_directory();
31. **void** create\_bot\_file();
32. vector<Func> get\_func\_list();
33. **private**:
34. Bot bot;
35. vector<Func> FuncList;
36. QString directory;
37. QString message;
38. };

**Info.h**

1. #ifndef INFO\_H
2. #define INFO\_H
4. #include <QDialog>
6. **namespace** Ui {
7. **class** info;
8. }
10. **class** info : **public** QDialog
11. {
12. Q\_OBJECT
14. **public**:
15. **explicit** info(QWidget \*parent = nullptr);
16. ~info();
17. signals:
18. **void** infoWindow();
20. **private** slots:
21. **void** on\_info\_clicked();
23. **void** mouseMoveEvent(QMouseEvent \*);
24. **void** mousePressEvent(QMouseEvent \*);
25. **void** mouseReleaseEvent(QMouseEvent \*);
27. **void** on\_pushButton\_clicked();
29. **private**:
30. Ui::info \*ui;
31. QPoint myPos2;
32. };
34. #endif // INFO\_H

**Mainwindow.h**

1. #ifndef MAINWINDOW\_H
2. #define MAINWINDOW\_H
3. #include <QMainWindow>
4. #include<QWidget>
5. #include <create\_bot\_window.h>
6. #include <info.h>
7. #include <QTranslator>
9. QT\_BEGIN\_NAMESPACE
10. **namespace** Ui
11. {
12. **class** MainWindow;
13. }
14. QT\_END\_NAMESPACE
16. **class** MainWindow : **public** QMainWindow
17. {
18. Q\_OBJECT
20. **public**:
21. MainWindow(QWidget \*parent = 0);
22. ~MainWindow();

25. **private** slots:
26. **void** on\_CreateBotButton\_clicked();
27. **void** on\_info\_clicked();
28. **void** on\_pushButton\_2\_clicked();
29. **void** on\_pushButton\_clicked();
31. **void** mouseMoveEvent(QMouseEvent \*);
32. **void** mousePressEvent(QMouseEvent \*);
33. **void** mouseReleaseEvent(QMouseEvent \*);
35. **protected**:
36. **void** changeEvent(QEvent \* event) override;
37. **private**:
38. Ui::MainWindow \*ui;
39. QTranslator qtLanguageTranslator;
41. QPoint myPos;
42. Create\_bot\_window \*CreateWindow1;
43. info \*Info;
45. };
46. #endif // MAINWINDOW\_H

**Bot.cpp**

1. #include <iostream>
2. #include "Bot.h"
3. **using** **namespace** std;

6. Bot::Bot() : Name("bot"),Token("token"),Prefix('.')
7. {
8. }
10. **void** Bot::set\_name(QString name)     //Set bot's name
11. {
12. Bot::Name = name;
13. }
14. **void** Bot::set\_token(QString token)   //Set bot's token
15. {
16. Bot::Token = token;
17. }
18. **void** Bot::set\_prefix(QString prefix)   //Set bot's commands prefix
19. {
20. Bot::Prefix = prefix;
21. }
22. QString Bot::get\_name()  //Return bot's name
23. {
24. **return** Bot::Name;
25. }
26. QString Bot::get\_token() //Return bot's token
27. {
28. **return** Bot::Token;
29. }
30. QString Bot::get\_prefix()  //Return bot's commands prefix
31. {
32. **return** Bot::Prefix;
33. }

**Create\_bot\_window .cpp**

1. #include "create\_bot\_window.h"
2. #include "ui\_create\_bot\_window.h"
3. #include <QtGui>
4. #include <QTranslator>
5. Create\_bot\_window::Create\_bot\_window(QWidget \*parent) :
6. QWidget(parent),
7. ui(**new** Ui::Create\_bot\_window)
8. {
9. ui->setupUi(**this**);
11. QPixmap bkgnd(":/resourec/img/1579284620\_32-89.jpg");
12. bkgnd = bkgnd.scaled(**this**->size(), Qt::IgnoreAspectRatio);
13. QPalette palette;
14. palette.setBrush(QPalette::Background, bkgnd);
15. **this**->setPalette(palette);
16. **this**->setWindowFlags(Qt::FramelessWindowHint);
18. // Задаём два пункта с текстом локалей в комбобоксе
19. ui->comboBox\_2->addItems(QStringList() << "en\_US" << "ru\_RU");
21. // подключаем к сигналу изменения пункта комбобокса лямбда функцию,
22. // в которой будет изменяться перевод приложения
23. // Здесь имеется интересный момент. Поскольку QComboBox имеет перегрузку сигнатуры сигнала,
24. // то нам необходимо скастовать сигнал к нужной сигнатуре.
25. // В данном случае будем использовать название пункта при его изменении
26. connect(ui->comboBox\_2, **static\_cast**<**void** (QComboBox::\*)(**const** QString &)>(&QComboBox::currentIndexChanged),
27. [=](**const** QString &str){
28. qtLanguageTranslator1.load("QtLanguage\_" + str, ".");   // Загружаем перевод
29. qApp->installTranslator(&qtLanguageTranslator1);        // Устанавливаем перевод в приложение
30. });
32. // Сделаем первоначальную инициализацию перевода для окна прилоежния
33. qtLanguageTranslator1.load(QString("QtLanguage\_") + QString("en\_US"));
34. qApp->installTranslator(&qtLanguageTranslator1);
35. }
37. Create\_bot\_window::~Create\_bot\_window()
38. {
39. **delete** ui;
40. }
42. **void** Create\_bot\_window::on\_CreateBotButton\_clicked()
43. {
44. **this**->close();
45. emit menuW();
46. }
48. **void** Create\_bot\_window::on\_comadLine\_textEdited(**const** QString &arg1)
49. {
50. **this**->com = arg1;
51. }
53. **void** Create\_bot\_window::on\_messLine\_textEdited(**const** QString &arg1)
54. {
55. **this**->mes = arg1;
56. }
58. **void** Create\_bot\_window::on\_checkBox\_5\_stateChanged(**int** arg1)
59. {
60. **if** (arg1 == 1)
61. mes\_permiss = **true**;
62. **else**
63. mes\_permiss = **false**;
64. }
66. **void** Create\_bot\_window::on\_AddMess\_clicked()
67. {
68. Creator::Func arg;
69. arg.Command = ui->comadLine->text();
70. mes = ui->messLine->text();
71. arg.FuncName = "Mess";
72. **if**(ui->checkBox\_5->isChecked())
73. {
74. arg.permissions = **true**;
75. }
76. **else**
77. {
78. arg.permissions = **false**;
79. }
80. bot.add\_in\_list(arg);
81. bot.set\_message(mes);
82. }


86. **void** Create\_bot\_window::on\_AddClear\_clicked()
87. {
88. Creator::Func arg;
89. arg.Command = "Clear";
90. arg.FuncName = "Clear";
91. **if**(ui->checkBox->isChecked())
92. {
93. arg.permissions = **true**;
94. }
95. **else**
96. {
97. arg.permissions = **false**;
98. }
99. bot.add\_in\_list(arg);
100. }
102. **void** Create\_bot\_window::on\_checkBox\_4\_stateChanged(**int** arg1)
103. {
104. **if**(arg1 == 1)
105. flip\_permiss = **true**;
106. **else**
107. flip\_permiss = **false**;
108. }
110. **void** Create\_bot\_window::on\_AddFlip\_clicked()
111. {
112. Creator::Func arg;
113. arg.Command = "flip";
114. arg.FuncName = "flip";
115. **if**(ui->checkBox\_4->isChecked())
116. {
117. arg.permissions = **true**;
118. }
119. **else**
120. {
121. arg.permissions = **false**;
122. }
123. bot.add\_in\_list(arg);
124. }
126. **void** Create\_bot\_window::on\_checkBox\_3\_stateChanged(**int** arg1)
127. {
128. **if**(arg1 == 1)
129. info\_permiss = **true**;
130. **else**
131. info\_permiss = **false**;
132. }
134. **void** Create\_bot\_window::on\_AddInfo\_clicked()
135. {
136. Creator::Func arg;
137. arg.Command = "info";
138. arg.FuncName = "info";
139. **if**(ui->checkBox\_3->isChecked())
140. {
141. arg.permissions = **true**;
142. }
143. **else**
144. {
145. arg.permissions = **false**;
146. }
147. bot.add\_in\_list(arg);
148. }
150. **void** Create\_bot\_window::on\_checkBox\_2\_stateChanged(**int** arg1)
151. {
152. **if**(arg1 == 1)
153. melling\_permiss = **true**;
154. **else**
155. melling\_permiss = **false**;
156. }
158. **void** Create\_bot\_window::on\_AddMassMelling\_clicked()
159. {
160. Creator::Func arg;
161. arg.Command = "mass";
162. arg.FuncName = "mass";
163. **if**(ui->checkBox\_2->isChecked())
164. {
165. arg.permissions = **true**;
166. }
167. **else**
168. {
169. arg.permissions = **false**;
170. }
171. bot.add\_in\_list(arg);
172. }
174. **void** Create\_bot\_window::on\_checkBox\_6\_stateChanged(**int** arg1)
175. {
176. **if**(arg1 == 1)
177. game\_permiss = **true**;
178. **else**
179. game\_permiss = **false**;
180. }
182. **void** Create\_bot\_window::on\_AddMafia\_clicked()
183. {
184. Creator::Func arg;
185. arg.Command = "mafia";
186. arg.FuncName = "mafia";
187. **if**(ui->checkBox\_6->isChecked())
188. {
189. arg.permissions = **true**;
190. }
191. **else**
192. {
193. arg.permissions = **false**;
194. }
195. bot.add\_in\_list(arg);
196. }
198. **void** Create\_bot\_window::on\_pushButton\_clicked()
199. {
200. QString Dir = QFileDialog::getOpenFileName(0, "Select dir", "");
201. ui->dirLine->insert(Dir);
202. }
204. **void** Create\_bot\_window::on\_CreateButton\_clicked()
205. {
206. bot.set\_info(ui->BotsName->text(), ui->Token->text(), ui->comboBox->currentText());
207. bot.set\_directory(ui->dirLine->text());
208. bot.create\_bot\_file();
209. }
211. **void** Create\_bot\_window::mouseMoveEvent(QMouseEvent \*e)
212. {
213. **if**(myPos1.x()>=0 && e->buttons()&& Qt::LeftButton)
214. {
215. QPoint diff = e->pos()- myPos1;
216. QPoint newpos = **this**->pos()+diff;
217. **this**->move(newpos);
218. }
219. }
220. **void** Create\_bot\_window::mousePressEvent(QMouseEvent \*e)
221. {
222. myPos1 = e->pos();
223. }
224. **void** Create\_bot\_window::mouseReleaseEvent(QMouseEvent \*e)
225. {
226. myPos1 = QPoint(-1,1);
227. }
229. **void** Create\_bot\_window::on\_pushButton\_2\_clicked()
230. {
231. close();
232. }


236. **void** Create\_bot\_window::on\_Back\_clicked()
237. {
238. **this**->close();
239. emit menuW();
240. }
241. **void** Create\_bot\_window::changeEvent(QEvent \*event)
242. {
243. // В случае получения события изменения языка приложения
244. **if** (event->type() == QEvent::LanguageChange) {
245. ui->retranslateUi(**this**);    // переведём окно заново
246. }
247. }
249. **void** Create\_bot\_window::on\_pushButton\_3\_clicked()
250. {
251. showMinimized ();
252. }

**Creator .cpp**

1. #include<iostream>
2. #include<vector>
3. #include "creator.h"
4. #include<QTextStream>
5. **using** **namespace** std;
6. QTextStream out(stdout);
8. **void** Creator::set\_info(**const** QString Name, **const** QString token, QString prefix)
9. {
10. bot.set\_name(Name);
11. bot.set\_token(token);
12. bot.set\_prefix(prefix);
13. }
14. **void** Creator::set\_directory(QString dir)
15. {
16. Creator::directory = dir;
17. }
18. **void** Creator::set\_func\_list(vector<Func> list)
19. {
20. Creator::FuncList.reserve(list.size());
21. **for** (unsigned **int** i = 0; i < list.size(); i++)
22. {
23. **this**->FuncList[i] = list[i];
24. }
25. }
26. **void** Creator::add\_in\_list(Func arg)
27. {
28. FuncList.push\_back(arg);
29. }
30. **void** Creator::set\_message(QString mess)
31. {
32. **this**->message = mess;
33. }
34. QString Creator::get\_directory()
35. {
36. **return** Creator::directory;
37. }
38. vector<Creator::Func> Creator::get\_func\_list()
39. {
40. **return** Creator::FuncList;
41. }
42. **void** Creator::create\_bot\_file()
43. {
44. QFile botF(get\_directory());
45. **if**(botF.open(QIODevice::WriteOnly))
46. {
47. QTextStream out(&botF);
48. botF.write("import discord\n");
49. botF.write("from discord.ext import commands\n");
50. botF.write("import random\n");
51. botF.write("from random import randint\n");
52. botF.write("import os\n");
53. botF.write("TOKEN = "); out << bot.get\_token() << endl;
54. botF.write("PREFIX = "); out << "'" << bot.get\_prefix() << "'" << endl;
55. botF.write("client = commands.Bot( command\_prefix = PREFIX )\n");
56. botF.write("@client.event\n");
57. botF.write("async def on\_ready():\n");
58. botF.write("\tprint( ' Im connected ' )\n");
59. botF.write("@client.event\n");
60. botF.write("async def on\_command\_error(ctx, error):\n");
61. botF.write("\tif isinstance(error, commands.CommandNotFound ):\n");
62. botF.write("\t\tawait ctx.send(embed = discord.Embed(description = f'\*\* {ctx.author.name}, this command does not exist. Existing teams can be viewed using {PREFIX}help .\*\*', color=discord.Color.red()))\n");
63. **for**(unsigned **int** i =0; i<FuncList.size(); i++)
64. {
65. out << "@client.command( pass\_context = True )" << endl;
66. **if**(FuncList[i].permissions == **true**)
67. out << "@commands.has\_permissions( administrator = True )" << endl;
68. out << "async def " << FuncList[i].Command;
69. **if**(FuncList[i].FuncName == "Mess")
70. {
71. out <<"( ctx ):" << endl;
72. out << "\tawait ctx.send( '" << **this**->message << "')" << endl;
73. }
74. **if**(FuncList[i].FuncName == "Clear")
75. {
76. out << "( ctx, amount = 100):" << endl;
77. out << "\tprint(ctx.author.name + '[command]: clear')" <<endl;
78. out << "\tawait ctx.channel.purge( limit = amount )" << endl;
79. }
80. **if**(FuncList[i].FuncName == "flip" )
81. {
82. out << "( ctx ):" << endl;
83. out << "\tprint(ctx.author.name + '[command]: flip')" << endl;
84. out << "\temb = discord.Embed ( title = 'Coin tossed', colour = discord.Color.gold() )" << endl;
85. out << "\trandom.seed()" << endl;
86. out << "\tmoney = randint(0, 1)" << endl;
87. out << "\tif money == 0:" << endl;
88. out << "\t\temb.set\_author( name = client.user.name, icon\_url = client.user.avatar\_url )" << endl;
89. out << "\t\temb.set\_footer( text = ctx.author.name, icon\_url = ctx.author.avatar\_url )" << endl;
90. out << "\t\temb.add\_field( name = 'Coin fell eagle up', value = 'Hope you are lucky' )" << endl;
91. out << "\t\temb.set\_image( url = 'https://coins.tsbnk.ru/upload/iblock/573/5015-0052.png' )" << endl;
92. out << "\t\tawait ctx.channel.send( embed = emb)" << endl;
93. out << "\telse:" << endl;
94. out << "\t\temb.set\_author( name = client.user.name, icon\_url = client.user.avatar\_url )" << endl;
95. out << "\t\temb.set\_footer( text = ctx.author.name, icon\_url = ctx.author.avatar\_url )" << endl;
96. out << "\t\temb.add\_field( name = 'Coin Tails Up', value = 'Hope you are lucky' )" << endl;
97. out << "\t\temb.set\_image( url = 'https://cs8.pikabu.ru/post\_img/big/2017/04/22/11/1492888050140348172.png' )" << endl;
98. out << "\t\tawait ctx.channel.send( embed = emb)" << endl;
99. }
100. **if**(FuncList[i].FuncName == "info")
101. {
102. out << "( ctx ):" << endl;
103. out << "\temb = discord.Embed ( title = 'My Info', colour = ctx.author.top\_role.colour )" << endl;
104. out << "\temb.set\_author ( name = client.user.name, icon\_url = client.user.avatar\_url )" << endl;
105. out << "\temb.add\_field ( name = ctx.author.name, value = f' Composed by {str(ctx.author.guild)}  Main role {str(ctx.author.top\_role)}')" << endl;
106. out << "\temb.set\_image ( url = ctx.author.avatar\_url )" << endl;
107. out << "\tawait ctx.channel.send( embed = emb )" << endl;
108. }
109. **if**(FuncList[i].FuncName == "mass")
110. {
111. out << "( ctx, mes : str ):" << endl;
112. out << "\tprint(ctx.author.name + '[command]: mass\_send ( ' + mes + ' )')" << endl;
113. out << "\tfor i in client.get\_channel(697822717050748979).members:" <<endl;
114. out << "\t\tawait i.send(mes)" << endl;
115. }
116. **if**(FuncList[i].FuncName == "mafia")
117. {
118. out << "( ctx ):" << endl;
119. out << "\tprint(ctx.author.name + '[command]: Mafia Game')" << endl;
120. out << "\tif len(client.get\_channel(714072862197022821).members) < 4:" <<endl;
121. out << "\t\tawait ctx.channel.send(' Too few people to play !!! ')" <<endl;
122. out << "\telse:" <<endl;
123. out << "\t\ttemp = 0" <<endl;
124. out << "\t\tply\_roles = []" <<endl;
125. out << "\t\trole\_list = []" <<endl;
126. out << "\t\tfor i in range(0, len(client.get\_channel(714072862197022821).members)):" <<endl;
127. out << "\t\t\trole\_list.append(i)" <<endl;
128. out << "\t\tif len(client.get\_channel(714072862197022821).members) > 7:" <<endl;
129. out << "\t\t\trole\_list[4] = 0" <<endl;
130. out << "\t\trandom.shuffle(role\_list)" <<endl;
131. out << "\t\trandom.shuffle(form\_list)" <<endl;
132. out << "\t\tfor i in client.get\_channel(714072862197022821).members:" <<endl;
133. out << "\t\t\tif role\_list[temp] == 0:" <<endl;
134. out << "\t\t\t\tply\_roles.append('Mafia is ' + i.name)" <<endl;
135. out << "\t\t\tif role\_list[temp] == 1:" <<endl;
136. out << "\t\t\t\tply\_roles.append('Commissioner is ' + i.name)" <<endl;
137. out << "\t\t\tif role\_list[temp] == 2:" <<endl;
138. out << "\t\t\t\tved = i" <<endl;
139. out << "\t\t\tif len(client.get\_channel(714072862197022821).members) > 5:" <<endl;
140. out << "\t\t\t\tif role\_list[temp] == 3:" <<endl;
141. out << "\t\t\t\t\tply\_roles.append('Doctor is ' + i.name)" <<endl;
142. out << "\t\t\t\telse:" <<endl;
143. out << "\t\t\t\t\tif role\_list[temp]>3:" <<endl;
144. out << "\t\t\t\t\t\tawait i.send('Peace')" <<endl;
145. out << "\t\t\telse:" <<endl;
146. out << "\t\t\t\tif role\_list[temp] > 2:" <<endl;
147. out << "\t\t\t\t\tawait i.send('Peace') " <<endl;
148. out << "\t\t\ttemp+=1" <<endl;
149. out << "\t\tfor i in range(len(ply\_roles)):" <<endl;
150. out << "\t\t\tawait ved.send(str(ply\_roles[i]))" <<endl;
151. }
153. }
154. botF.write("client.run(TOKEN)\n");
155. }
156. botF.close();
157. }

**Info.cpp**

1. #include "info.h"
2. #include "ui\_info.h"
3. #include <QtGui>
4. info::info(QWidget \*parent) :
5. QDialog(parent),
6. ui(**new** Ui::info)
7. {
8. ui->setupUi(**this**);
9. **this**->setWindowFlags(Qt::FramelessWindowHint);
10. QPixmap pix(":/resourec/img/SpbGUTelekom.png");
11. ui->image2->setPixmap(pix);
13. }
15. info::~info()
16. {
17. **delete** ui;
18. }
20. **void** info::on\_info\_clicked()
21. {
22. **this**->close();
23. emit infoWindow();
24. }
25. **void** info::mouseMoveEvent(QMouseEvent \*e)
26. {
27. **if**(myPos2.x()>=0 && e->buttons()&& Qt::LeftButton)
28. {
29. QPoint diff = e->pos()- myPos2;
30. QPoint newpos = **this**->pos()+diff;
31. **this**->move(newpos);
32. }
33. }
34. **void** info::mousePressEvent(QMouseEvent \*e)
35. {
36. myPos2 = e->pos();
37. }
38. **void** info::mouseReleaseEvent(QMouseEvent \*e)
39. {
40. myPos2 = QPoint(-1,1);
41. }
43. **void** info::on\_pushButton\_clicked()
44. {
45. close();
46. }

**Main.cpp**

1. #include "mainwindow.h"
2. #include <QApplication>

5. **int** main(**int** argc, **char** \*argv[])
6. {
7. QApplication a(argc, argv);
8. MainWindow w;
9. w.show();
10. **return** a.exec();
11. }

**Mainwindow.cpp**

1. #include "mainwindow.h"
2. #include "ui\_mainwindow.h"
3. #include <QApplication>
4. #include <QTranslator>
5. #include <QLibraryInfo>
6. #include <QDebug>
7. #include <QPixmap>
8. #include <QtGui>
10. MainWindow::MainWindow(QWidget \*parent):
11. QMainWindow(parent),
12. ui(**new** Ui::MainWindow)
13. {
14. ui->setupUi(**this**);
16. ui->comboBox->addItems(QStringList() << "en\_US" << "ru\_RU");
18. connect(ui->comboBox, **static\_cast**<**void** (QComboBox::\*)(**const** QString &)>(&QComboBox::currentIndexChanged),
19. [=](**const** QString &str){
20. qtLanguageTranslator.load("QtLanguage\_" + str, ".");   // Загружаем перевод
21. qApp->installTranslator(&qtLanguageTranslator);        // Устанавливаем перевод в приложение
22. });
23. // Сделаем первоначальную инициализацию перевода для окна прилоежния
24. qtLanguageTranslator.load(QString("QtLanguage\_") + QString("en\_US"));
25. qApp->installTranslator(&qtLanguageTranslator);
27. **this**->setWindowFlags(Qt::FramelessWindowHint);
29. QPixmap pix(":/resourec/img/character-16-563887.png");
30. ui->image->setPixmap(pix);

33. CreateWindow1 = **new** Create\_bot\_window();
34. connect(CreateWindow1, &Create\_bot\_window::menuW, **this**, &MainWindow::show);
36. Info = **new** info();
37. connect(Info, &info::infoWindow, **this**, &MainWindow::show);

40. }
42. MainWindow::~MainWindow()
43. {
44. **delete** ui;
45. }

48. **void** MainWindow::on\_CreateBotButton\_clicked()
49. {
50. CreateWindow1->show();
51. **this**->close();
52. }

55. **void** MainWindow::on\_info\_clicked()
56. {
57. Info ->show();
59. }

62. **void** MainWindow::on\_pushButton\_2\_clicked()
63. {
64. close();
65. }
67. **void** MainWindow::on\_pushButton\_clicked()
68. {
69. showMinimized ();
71. }
73. **void** MainWindow::mouseMoveEvent(QMouseEvent \*e)
74. {
75. **if**(myPos.x()>=0 && e->buttons()&& Qt::LeftButton)
76. {
77. QPoint diff = e->pos()- myPos;
78. QPoint newpos = **this**->pos()+diff;
79. **this**->move(newpos);
80. }
81. }
82. **void** MainWindow::mousePressEvent(QMouseEvent \*e)
83. {
84. myPos = e->pos();
85. }
86. **void** MainWindow::mouseReleaseEvent(QMouseEvent \*e)
87. {
88. myPos = QPoint(-1,1);
89. }
90. **void** MainWindow::changeEvent(QEvent \*event)
91. {
92. // В случае получения события изменения языка приложения
93. **if** (event->type() == QEvent::LanguageChange) {
94. ui->retranslateUi(**this**);    // переведём окно заново
95. }
96. }

**Код бота на python**

1. **import** discord
2. **from** discord.ext **import** commands
3. **import** random
4. **from** random **import** randint
5. **import** os
6. TOKEN = 'NzQ3NDM2ODEyNzI0NzMxOTI1.X0O2xg.K8HgX7GYUtgm0Em57wGVd2gg3w8'
7. PREFIX = '\*'
8. client = commands.Bot( command\_prefix = PREFIX )
9. @client.event
10. async **def** on\_ready():
11. **print**( ' Im connected ' )
12. @client.event
13. async **def** on\_command\_error(ctx, error):
14. **if** isinstance(error, commands.CommandNotFound ):
15. await ctx.send(embed = discord.Embed(description = f'\*\* {ctx.author.name}, this command does not exist. Existing teams can be viewed using {PREFIX}help .\*\*', color=discord.Color.red()))
16. @client.command( pass\_context = True )
17. async **def** hello( ctx ):
18. await ctx.send( 'Hello World')
20. @client.command( pass\_context = True )
21. @commands.has\_permissions( administrator = True )
22. async **def** Clear( ctx, amount = 100):
23. **print**(ctx.author.name + '[command]: clear')
24. await ctx.channel.purge( limit = amount )
26. @client.command( pass\_context = True )
27. async **def** flip( ctx ):
28. **print**(ctx.author.name + '[command]: flip')
29. emb = discord.Embed ( title = 'Coin tossed', colour = discord.Color.gold() )
30. random.seed()
31. money = randint(0, 1)
32. **if** money == 0:
33. emb.set\_author( name = client.user.name, icon\_url = client.user.avatar\_url )
34. emb.set\_footer( text = ctx.author.name, icon\_url = ctx.author.avatar\_url )
35. emb.add\_field( name = 'Coin fell eagle up', value = 'Hope you are lucky' )
36. emb.set\_image( url = 'https://coins.tsbnk.ru/upload/iblock/573/5015-0052.png' )
37. await ctx.channel.send( embed = emb)
38. **else**:
39. emb.set\_author( name = client.user.name, icon\_url = client.user.avatar\_url )
40. emb.set\_footer( text = ctx.author.name, icon\_url = ctx.author.avatar\_url )
41. emb.add\_field( name = 'Coin Tails Up', value = 'Hope you are lucky' )
42. emb.set\_image( url = 'https://cs8.pikabu.ru/post\_img/big/2017/04/22/11/1492888050140348172.png' )
43. await ctx.channel.send( embed = emb)
45. @client.command( pass\_context = True )
46. async **def** info( ctx ):
47. emb = discord.Embed ( title = 'My Info', colour = ctx.author.top\_role.colour )
48. emb.set\_author ( name = client.user.name, icon\_url = client.user.avatar\_url )
49. emb.add\_field ( name = ctx.author.name, value = f' Composed by {str(ctx.author.guild)}  Main role {str(ctx.author.top\_role)}')
50. emb.set\_image ( url = ctx.author.avatar\_url )
51. await ctx.channel.send( embed = emb )
53. @client.command( pass\_context = True )
54. @commands.has\_permissions( administrator = True )
55. async **def** mass( ctx, mes : str ):
56. **print**(ctx.author.name + '[command]: mass\_send ( ' + mes + ' )')
57. **for** i **in** client.get\_channel(697822717050748979).members:
58. await i.send(mes)
60. @client.command( pass\_context = True )
61. async **def** mafia( ctx ):
62. **print**(ctx.author.name + '[command]: Mafia Game')
63. **if** len(client.get\_channel(714072862197022821).members) < 4:
64. await ctx.channel.send(' Too few people to play !!! ')
65. **else**:
66. temp = 0
67. ply\_roles = []
68. role\_list = []
69. **for** i **in** range(0, len(client.get\_channel(714072862197022821).members)):
70. role\_list.append(i)
71. **if** len(client.get\_channel(714072862197022821).members) > 7:
72. role\_list[4] = 0
73. random.shuffle(role\_list)
74. random.shuffle(form\_list)
75. **for** i **in** client.get\_channel(714072862197022821).members:
76. **if** role\_list[temp] == 0:
77. ply\_roles.append('Mafia is ' + i.name)
78. **if** role\_list[temp] == 1:
79. ply\_roles.append('Commissioner is ' + i.name)
80. **if** role\_list[temp] == 2:
81. ved = i
82. **if** len(client.get\_channel(714072862197022821).members) > 5:
83. **if** role\_list[temp] == 3:
84. ply\_roles.append('Doctor is ' + i.name)
85. **else**:
86. **if** role\_list[temp]>3:
87. await i.send('Peace')
88. **else**:
89. **if** role\_list[temp] > 2:
90. await i.send('Peace')
91. temp+=1
92. **for** i **in** range(len(ply\_roles)):
93. await ved.send(str(ply\_roles[i]))
94. client.run(TOKEN)