

"Теле бот" Розробка телеграм-бота для ОРС-технології програмування
вбудованої системи управління на базі ПЛК

ЗМІСТ

1. ВСТУП.....	3
2. ОБГРУНТУВАННЯ РОЗРОБКИ.....	4-5
3. РОЗРОБКА БОТА.....	6-16
3.1 Чат-боти.....	6
3.2 Персональний тек.....	6-7
3.3 Початок розробки чат-бота.....	7-10
3.4 Реалізація кнопок.....	10-13
3.4.1 Створення кнопок з вибором відповіді користувача.....	10-11
3.4.2 Обробка результату Так/Ні.....	12
3.4.3 Розробка інтегрованих кнопок.....	12-13
3.5 Отримання даних з ОРС.....	13-16
3.5.1 Підключення боту до ОРС.....	13-14
3.5.2 Зчитування та видача інформації.....	14-16
4. РОЗРОБКА БАЗИ ДЛЯ ТЕСТУВАННЯ.....	17-25
4.1 Створення проекту.....	17-18
4.2 Оголошення змінних для ОРС серверу.....	19
5. КОНФІГУРАЦІЯ ОРС.....	20-22
6. ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ.....	23-25
7. ВИСНОВКИ.....	26
8. ЛІТЕРАТУРА.....	27
9. АНОТАЦІЯ.....	28
10. ДОДАТОК 1.....	29-31

1. ВСТУП

Вбудована система – спеціалізована комп'ютерна система або обчислювальний пристрій, призначений для виконання обмеженої кількості функцій, часто, з обмеженнями реального часу[6].

Актуальною є розробка програмування вбудованої системи управління яка дозволяє системі реального часу змінювати та фактично програмувати вбудовану систему управління на базі ПЛК.

Чат-бот – комп'ютерна програма, розроблена на основі нейромереж та технологій машинного навчання, за допомогою якої можливо здійснювати комунікацію в аудіо або текстовому форматі[8], але чат-бот також може використовуватися для програмування вбудованих систем без зміни самої програми ПЛК.

Більш складні боти надають можливість вводу тексту і отримання інформації згідно до запиту користувача. Одними з найпопулярніших месенджерів, які підтримують чат-бот програми є месенджери: Telegram та Viber[14].

Система має виконувати задачу по програмуванню вбудованої системи управління в такій полідовності:

Бот>сервер>телеграм-мережа>сервер OPCUA>мережа-ПЛК

2. ОБГРУНТУВАННЯ РОЗРОБКИ

Метою роботи є розробка чат-боту у сервісі Telegram, який може бути використаний для отримання інформації з вбудованих систем на базі OPC сервера та відправляти по запиті користувача необхідну інформацію.

Серед інших представників чат-ботів ця розробка є унікальною, серед інтернету у вільному доступі не було виявлено схожої реалізації запрограмованих для цього чат-ботів, які передавали інформацію з ПЛК через OPC сервер[2].

Далі додається наступний алгоритм обміном даних. Обмін даними вбудованих систем на базі OPC серверу відбувається наступним чином:

1. Налаштовується сам конфігуратор.
2. Інформація з контролерів надходить на OPC сервер.
3. SCADA-система приймає дані з контролерів за допомогою читання даних через OPC.
4. Телеграм бот підключається безпосередньо до OPC сервера і по команді отримує данні, переписує необхідні змінні, контролює весь процес роботи.
5. Телеграм бот відправляє дані користувачеві, та відправляє результати змін.

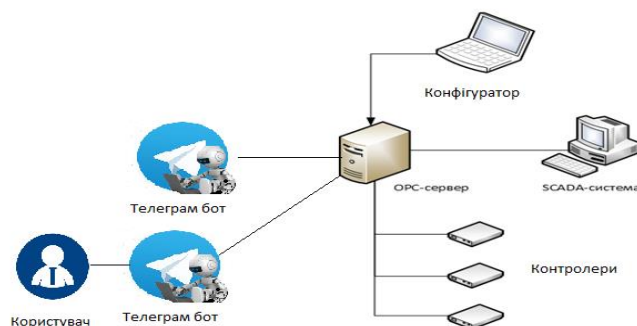


Рисунок 1 – схема обміну даних через OPC сервер.

Розробка телеграм-бота для ОРС-технології програмування вбудованої системи управління на базі ПЛК. Таким чином є доцільно розробка системи програмування вбудованої системи управління на базі ПЛК.

3. РОЗРОБКА ЧАТ-БОТУ

У наш час ми все частіше користуємося чат-ботами для різних цілей, які можуть відрізнитися від користувача, але самі чат-боти працюють однаково[3]. Протягом даної роботи ми розробимо чат-бота, який буде обмінюватися даними вбудованих систем на базі OPC сервера.

3.1 Чат боти

Зазвичай, чат-бот пропонує користувачу обрати варіант серед запропонованих[1]. Тобто отримання інформації або виконання інших задач відбувається шляхом вибору запропонованих відповідей, або запропонованих категорій, і користувач не вводить текст до вікна чату. Більш складні боти надають можливість вводу тексту і отримання інформації згідно до запиту користувача[10].

Віртуальні чат-боти дозволяють робити замовлення або спілкуватися в режимі он-лайн[7].

За сферою застосування чат-ботів поділяють на:

- р2р – персональні комунікації (для особистого спілкування);
- b2c – споживчі (підтримка клієнтів компанії на корпоративному сайті та в мобільних додатках).

3.2 Персональний тег

Першим кроком у програмуванні чат-боту є отримання його персонального токена. Персональний токен – це унікальний номер, за яким працює чат-бот, без нього програма не дізнається через кого будуть оброблятися запити[12]. Щоб отримати персональний тег для бота у сервісі Telegram, необхідно поспілкуватись з «BotFather». У результаті ми отримаємо нашого чат-бота, його токен, задамо йому ім'я та оберемо картинку.

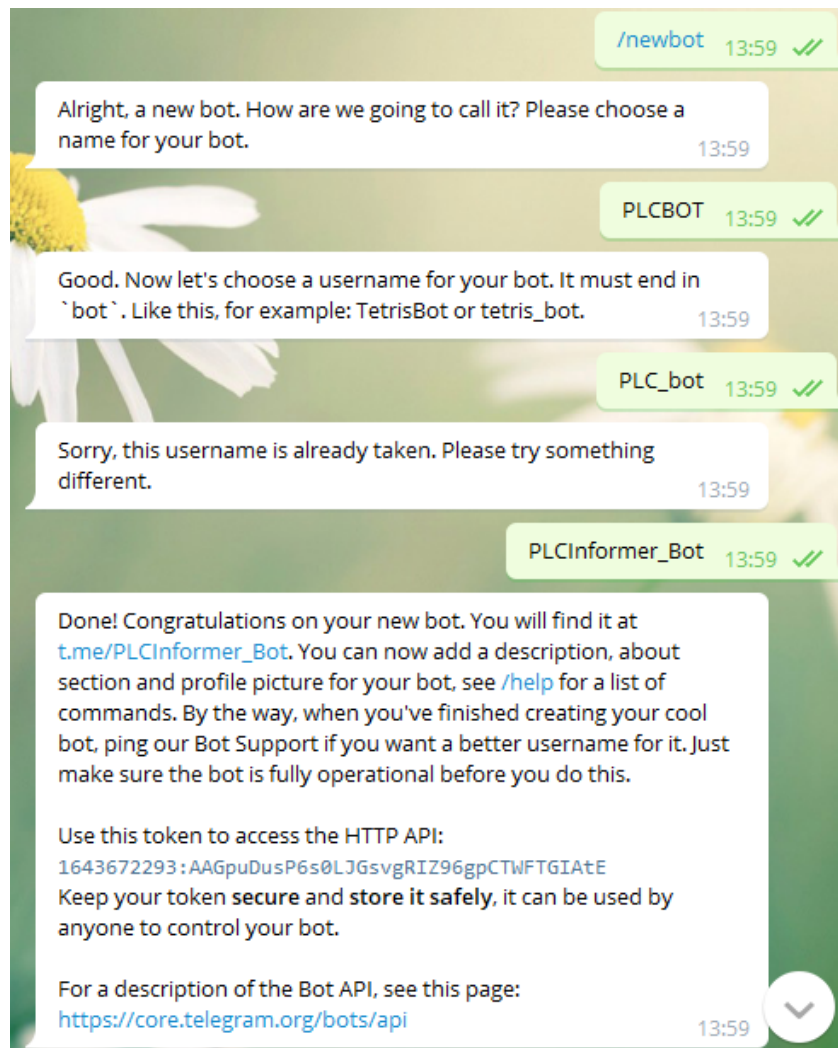


Рисунок 2 – Приклад бесіди з «BotFather».

3.3 Початок розробки чат-бота

Першим етапом є вибір середі програмування у якій ми будемо робити чат-бота та мову написання, для прикладу я використовую середу «Atom», а мовою є «Python»[11].

Atom — розроблений компанією «GitHub» вільний текстовий редактор і редактор коду, який може використовуватися як самодостатнє рішення, так і у ролі технологічного стека для побудови різних спеціалізованих рішень.

Python - високорівнева мова програмування загального призначення з динамічною типізацією і автоматичним управлінням пам'яттю, орієнтований на

підвищення продуктивності розробника, читання коду і його якості, а також на забезпечення переносимості написаних на ньому програм[13].

Вигляд середи програмування Atom представлено на Рисунку 3.

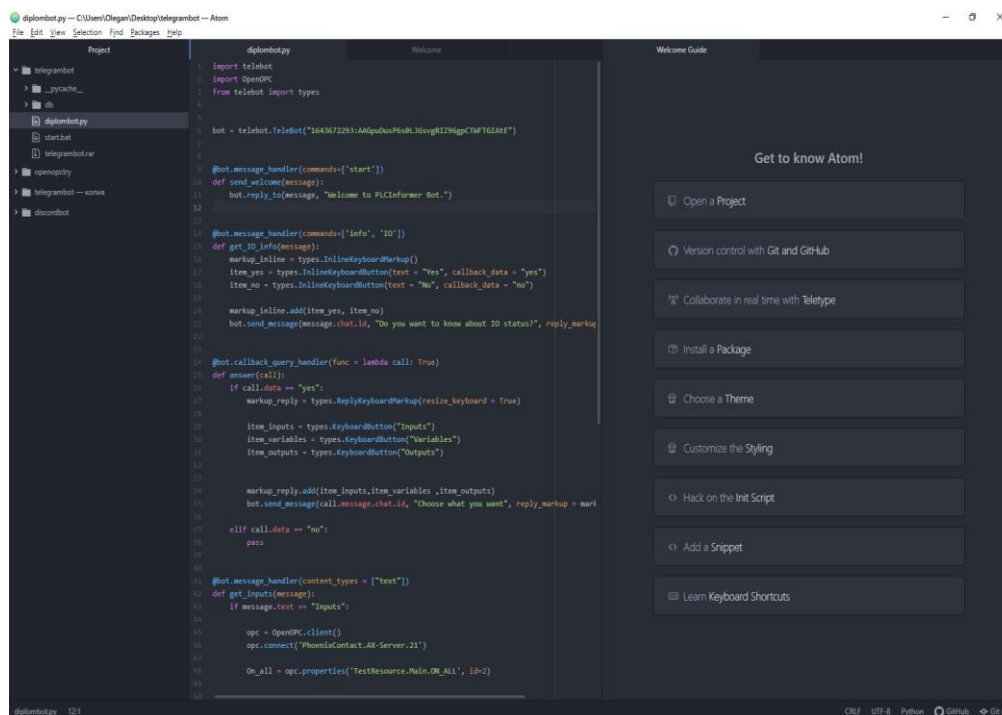


Рисунок 3 – Вигляд середи Atom.

Після вибору середи для написання та мови, наступним кроком стане під'єднання бібліотек. Під'єднання у «Python» проводиться через команду «import».

Щоб наш «Python» розумів, що ми хочемо робити з "незрозумілими" для нього командами, ми прописуємо список бібліотек, які він буде використовувати.

«Import telebot» – бібліотека для розпізнавання рядків коду пов'язаного зі створенням телеграм бота.

«Import OpenOPC» – бібліотека для роботи з даними, які поступають на OPC сервер з ПЛК.

«From telebot import types» – під-бібліотека «import telebot» для використання більшого функціоналу, в нашому випадку реалізація так званих «кнопок».

Приклад під'єднання бібліотек є на Рисунку 4.

```
1 import telebot
2 import OpenOPC
3 from telebot import types
```

Рисунок 4 – Під'єднання бібліотек.

Пропис токєну здійснюється за командою «bot = telebot.TeleBot("*")», де "*" – ваш токен. Даною командою задається токен для нашого телеграм бота, без нього він не буде функціонувати.

```
bot = telebot.TeleBot("1643672293:AAGpuDusP6s0LJGsvgRIZ96gpCTWFTGIAtE")
```

Рисунок 5 – Пропис токєну

Першою командою, яка запустить нашого бота буде команда «/start».

```
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.reply_to(message, "Welcome to PLCInformer Bot.")
```

Рисунок 6 – Пропис команди «/start».

Коли нам щось потрібно від чат-бота, ми прописуємо команду в чаті і нам висвічується результат. «@Bot.message_handler (commands = ['start'])» даною командою ми задаємо необхідна умову для початку роботи з ботом, а саме, коли користувач набирає в чаті «/start». Далі ми задаємо, що буде виконуватися коли користувач введе необхідну команду.

«Def send_welcome (message):» ми створюємо функцію, яка виконає дію по команді, а саме «bot.reply_to (message," Welcome to PLCInformer Bot. ")». Бот при отриманні команди «/start» відповідає користувачеві наступним чином: "Welcome to PLCInformer Bot.».

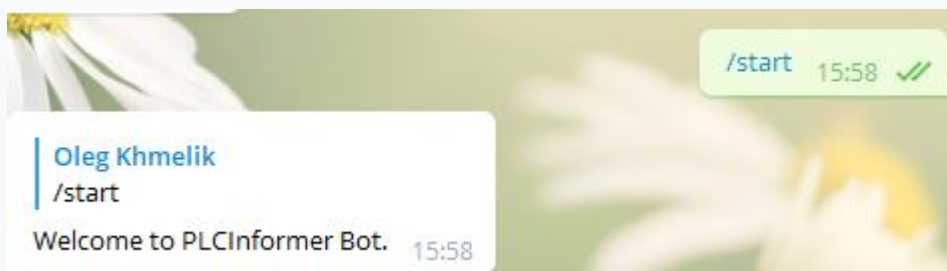


Рисунок 7 – Реакція боту на «/start».

Таким чином з'являється перша та основна робоча версія чат-боту.

3.4 Реалізація кнопок

У телеграм ботах можна створювати «кнопки» які значно полегшують використання ботів, тобто замість введення команд, ви просто натискаєте на кнопку і отримуєте результат.

3.4.1 Створення кнопок з вибором відповіді користувача

```
@bot.message_handler(commands=['info', 'IO'])
def get_IO_info(message):
    markup_inline = types.InlineKeyboardMarkup()
    item_yes = types.InlineKeyboardButton(text = "Yes", callback_data = "yes")
    item_no = types.InlineKeyboardButton(text = "No", callback_data = "no")

    markup_inline.add(item_yes, item_no)
    bot.send_message(message.chat.id, "Do you want to know about IO status?", reply_markup = markup_inline )
```

Рисунок 8 – Пропис «кнопок».

Для початку ми знову таки ставимо умову, того, що бот визначить потрібну команду «@bot.message_handler (commands = ['info', 'IO'])» тут бот буде реагувати тільки на «/info» або «/IO». Також задається функція «def get_IO_info (message):», у якій створюється повідомлення з вибором з кнопок і

їх відповіддю на натискання. «Markup_inline = types.InlineKeyboardMarkup()» задаємо, що під повідомленням у нас будуть кнопки.

Функції:

```
«Item_yes = types.InlineKeyboardButton (text = " Yes ", callback_data = " yes ")»
```

```
«item_no = types.InlineKeyboardButton (text = " No ", callback_data = " no ")»
```

Використовуються для створення кнопок з реакцією «callback_data». Після додаємо ці кнопки під повідомленням командою: «markup_inline.add (item_yes, item_no)». Та виводимо саме повідомлення з прив'язкою кнопок «bot.send_message (message.chat.id," Do you want to know about IO status?", Reply_markup = markup_inline)», де «reply_markup = markup_inline» прив'язка кнопок під повідомленням.

Реакція боту на команду «/info» або «/IO» виглядає наступним чином:

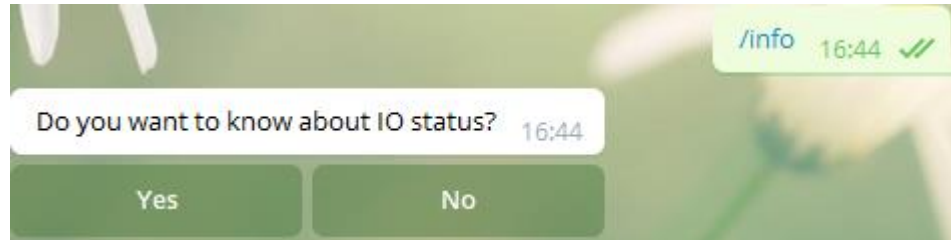


Рисунок 9 – Повідомлення з кнопками.

3.4.2 Обробка результату Так/Ні

Наступним кроком буде розробка обробки результату на кнопки.

```
@bot.callback_query_handler(func = lambda call: True)
def answer(call):
    if call.data == "yes":
        markup_reply = types.ReplyKeyboardMarkup(resize_keyboard = True)

        item_inputs = types.KeyboardButton("Inputs")
        item_variables = types.KeyboardButton("Variables")
        item_outputs = types.KeyboardButton("Outputs")

        markup_reply.add(item_inputs, item_variables, item_outputs)
        bot.send_message(call.message.chat.id, "Choose what you want", reply_markup = markup_reply )

    elif call.data == "no":
        pass
```

Рисунок 10 – Пропис обробки результату та додавання кнопок.

Спочатку задаємо умову, а саме: «@ bot.callback_query_handler (func = lambda call: True)», але це функція на реакцію натискання кнопок «Yes» / «No». Далі йде оголошення функції «def answer (call):»

Після йде опис нашої умови: «if call.data == " yes":», якщо ми натиснули «Yes» відбувається наступне: «markup_reply = types.ReplyKeyboardMarkup (resize_keyboard = True)» під нашим письмовим чатом будуть додаватися 3 нові кнопки, які будуть використовуватися в подальшому для полегшення набору команд.

3.4.3 Розробка інтегрованих кнопок

Першим кроком перед додаванням кнопок буде уникнення проблем з масштабуванням. «Resize_keyboard = True» – необхідно прописати щоб уникнути проблем з масштабуванням кнопок.

Функції:

«Item_inputs = types.KeyboardButton ("Inputs")»

«item_outputs = types.KeyboardButton (" Outputs ")»

«item_variables = types.KeyboardButton("Variables")»

Відповідають за оголошення того, що у нас є кнопки, а «markup_reply.add (item_inputs, item_variables, item_outputs)» додає їх.

«Bot.send_message (call.message.chat.id, "Choose what you want", reply_markup = markup_reply)» далі просто виводиться текстове повідомлення «Choose what you want». Якщо ми натискаємо «No», то нічого не станеться «elif call.data == " no ":». «Pass» знаходиться тут щоб умова працювала, ця команда просто пропускає дію.

Результат цієї частини виглядає наступним чином:

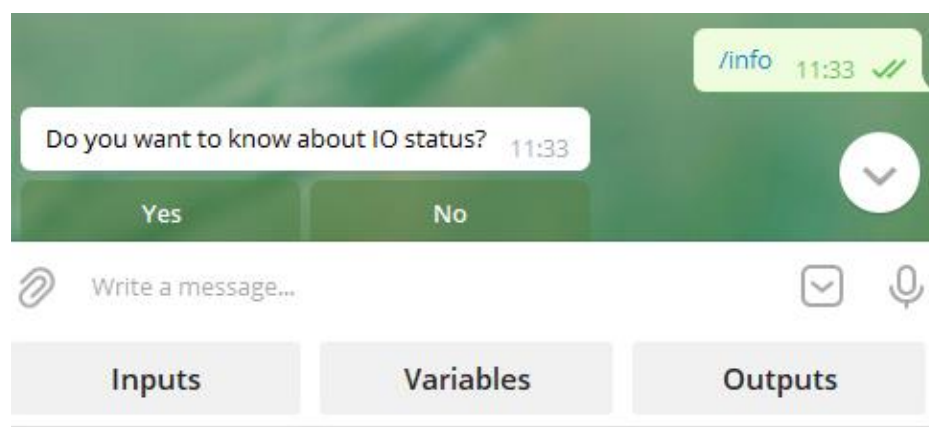


Рисунок 11 – Інтегровані кнопки.

3.5 Отримання даних з OPC

Для отримання даних вбудованих систем на базі OPC серверу ми будемо використовувати емуляцію як і самого серверу так й ПЛК[9]. Емуляція буде проходити з використанням технології PC WORX SRT. PC WORX SRT – це набір програм для емуляції ПЛК[4].

3.5.1 Підключення боту до OPC

Для того, щоб чат-бот підключився до вбудованого серверу OPC його потрібно задати як «Клієнт», виконується це через команду: «opc=OpenOPC.client()». Далі ми підключаємося до доступного серверу за командою «opc.connect("*")», де «*» – адрес серверу.

Повний вид пропису підключення:

```
opc = OpenOPC.client()
opc.connect('PhoenixContact.AX-Server.21')
```

Рисунок 12 – Підключення чат-боту до OPC серверу.

3.5.2 Зчитування та видача інформації

Реалізація зчитування та видачі інформації з OPC сервера:

```
@bot.message_handler(content_types = ["text"])
def get_inputs(message):
    if message.text == "Inputs":

        opc = OpenOPC.client()
        opc.connect('PhoenixContact.AX-Server.21')

        On_all = opc.properties('TestResource.Main.ON_ALL', id=2)

        bot.send_message (message.chat.id, f" ON_ALL : {On_all}")

    opc.close()
```

Рисунок 13 – Пропис зчитування та видачі інформації за командою «Inputs».

```
elif message.text == "Outputs":
    opc = OpenOPC.client()
    opc.connect('PhoenixContact.AX-Server.21')

    Pump = opc.properties('TestResource.Main.PUMP', id=2)

    bot.send_message (message.chat.id, f" PUMP : {Pump}")

    opc.close()
```

Рисунок 14 – Пропис зчитування та видачі інформації за командою «Outputs».

```

elif message.text == "Variables":
    opc = OpenOPC.client()
    opc.connect('PhoenixContact.AX-Server.21')

    Tin = opc.properties('TestResource.Main.TIN', id=2)
    Tout = opc.properties('TestResource.Main.TOUT', id=2)

    bot.send_message (message.chat.id, f" TIN : {Tin}")
    bot.send_message (message.chat.id, f" TOUT : {Tout}")

    opc.close()

```

Рисунок 15 – Пропис зчитування та видачі інформації за командою «Outputs».

Для читання наших змінних з вбудованої системи на базі OPC існує функція «opc.properties (*)», де «*» назва нашої змінної, яку ми дізнаємося завдяки утиліті OPC Client. Для виведення даних з OPC ми використаємо прив'язку змінної до команди.

Приклад команди «Pump = opc.properties ('TestResource.Main.PUMP', id = 2)», де: «Pump» назва нашої змінної, «opc.properties» наша функція запиту отримання даних, «TestResource.Main.PUMP» назва нашої змінної, «id = 2» то що ми виводимо з неї.

Спочатку виводиться повний список даних про змінну, якщо використовується функція без приписки «id = *», в нашому випадку виводиться тільки значення змінної. Висилання даних через бота відбувається ідентичним чином, як і висилання повідомлення по команді, а саме «bot.send_message (message.chat.id, f" PUMP: {Pump} ")», де «{Pump}» це наше привласнення функції.

Після отримання даних в кінці функції необхідно прописати «opc.close ()» для оптимізації використання з'єднання з OPC сервером, без цієї команди після виконання функції з'єднання з вбудованою системою на базі OPC сервера не припиняється, і повторне виконання команди додасть ще 1 підключення, не вимикаючи минулого. Тому краще закривати канал спілкування.

Таким чином реалізуються інші запити. Результат обробки команд:

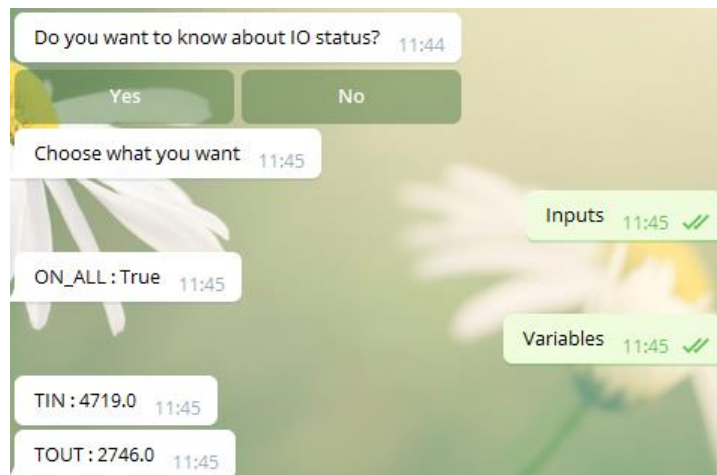


Рисунок 16 – Результат обробки команд та видача інформації.

4. РОЗРОБКА БАЗИ ДЛЯ ТЕСТУВАННЯ

Базою для тестування нашої розробки чат-бота для вбудованих систем на базі OPC буде проект у PC WORX з емуляцією завдяки програмному забезпеченню PC WORX SRT[5].

4.1 Створення проекту

Першим кроком буде створення нового проекту в ПЗ PC WORX. Створення проекту:

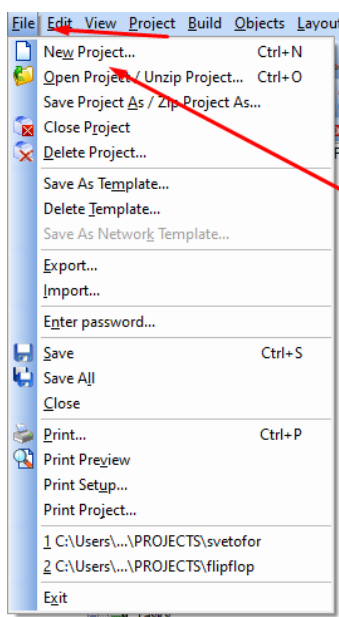


Рисунок 17 – Створення нового проекту у PC WORX.

У наступному вікні нам необхідно обрати наш контролер. У нашому випадку контролером виступає PC WORX SRT V1.1. Меню вибору контролера проілюстровано на Рисунку 17.

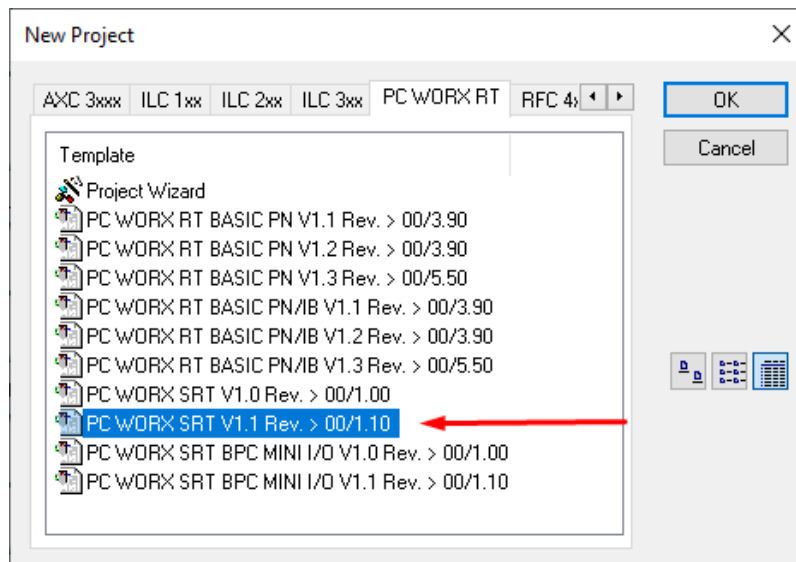


Рисунок 18 – Вікно вибору ПЛК у PC WORX.

Для прикладу був обран простий проект у PC WORX, у якому продемонстрована робота заповнення ємності з різними швидкостями закачки та викачки рідин. Повний проект проілюстрован на Рис. 18.

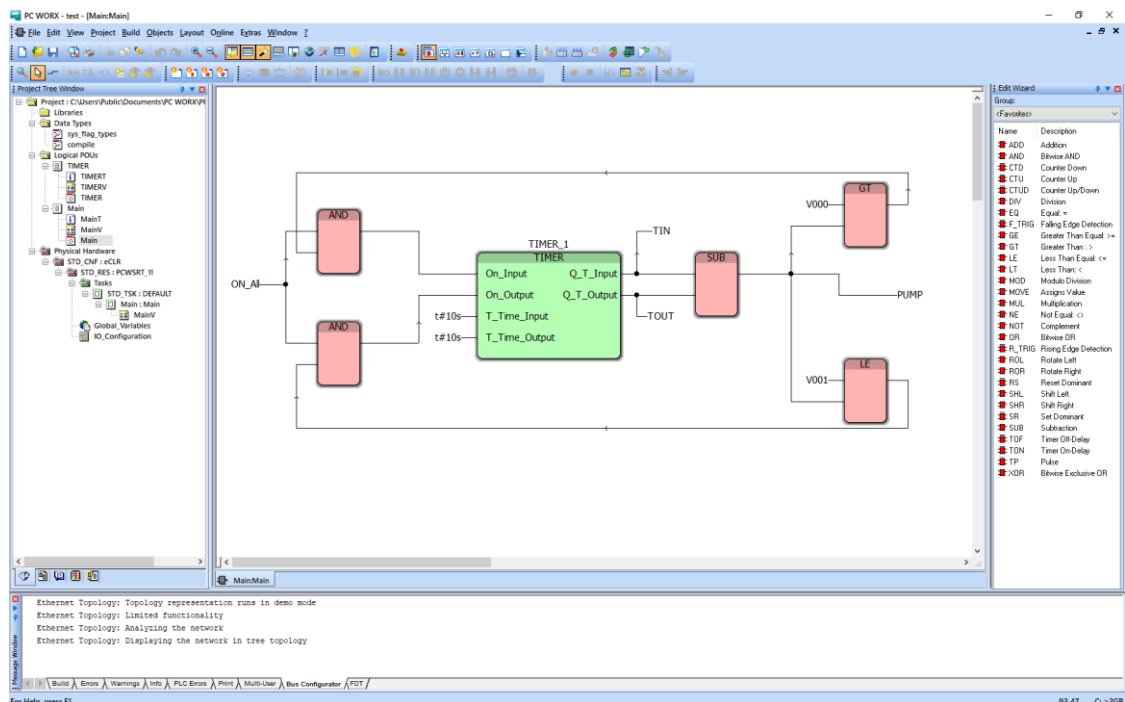


Рисунок 19 – Проект у PC WORX.

Цей проект працює на таймерах, які виконують задачу викачки та закачки рідин у ємність.

4.2 Оголошення змінних для OPC серверу

Щоб данні потрапляли до нашого вбудованого на базі OPC серверу, необхідно задати цю вимогу. Робиться вона через вікно змінних, а саме біля необхідної змінної ставиться галочка OPC. Приклад списку змінних з проекту PC WORX:

Name	Type	Usage	Description	Address	Init	Retain	PDD	OPC	TB	Hid...	Init...	Default Hid...	Re...	Con
[-] Variables														
V000	TIME	VAR			t#0s	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
V001	TIME	VAR			t#2s	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TIN	TIME	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TOUT	TIME	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
[-] Else														
TIMER_1	TIMER	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
[-] Output														
PUMP	TIME	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
[-] Input														
ON_A1	BOOL	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

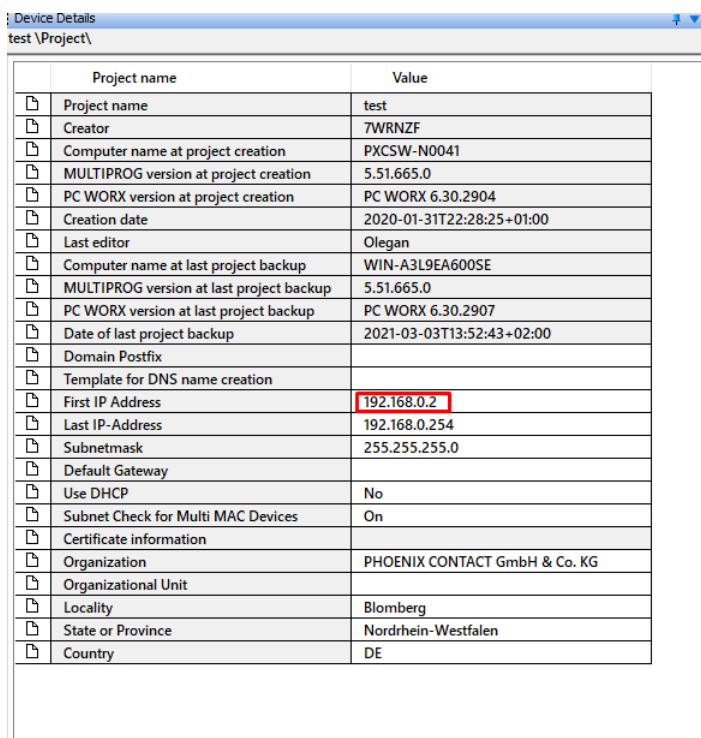
Рисунок 20 – Список змінних PC WORX.

5. КОНФІГУРАЦІЯ OPC

OPC - сімейство програмних технологій, що надають єдиний інтерфейс для управління об'єктами автоматизації і технологічними процесами. Багато з OPC протоколів базуються на Windows-технологіях: OLE, ActiveX, COM / DCOM. Такі OPC протоколи, як OPC XML DA і OPC UA є платформозависимі[2].

Конфігурація OPC серверу досить легка справа, щоб наші данні з ПЛК потрапляли до серверу, необхідно по-перше поставити галочку, як показано на Рис. 19, та задати правильний IP-адрес для нашого ПЛК.

Адрес за яким будуть обмінюватися данні у нашому випадку локальний, тобто знаходиться на одній машині. Конфігурація адресу проводиться у пункті «Bus Configuration Workspace», де нам необхідно прописати IP-адрес. Приклад налаштованої конфігурації відображен на Рисунку 20, де червоним виділено IP-адрес машини[].



Project name	Value
Project name	test
Creator	7WRNZF
Computer name at project creation	PXCSW-N0041
MULTIPROG version at project creation	5.51.665.0
PC WORX version at project creation	PC WORX 6.30.2904
Creation date	2020-01-31T22:28:25+01:00
Last editor	Olegan
Computer name at last project backup	WIN-A3L9EA600SE
MULTIPROG version at last project backup	5.51.665.0
PC WORX version at last project backup	PC WORX 6.30.2907
Date of last project backup	2021-03-03T13:52:43+02:00
Domain Postfix	
Template for DNS name creation	
First IP Address	192.168.0.2
Last IP-Address	192.168.0.254
Subnetmask	255.255.255.0
Default Gateway	
Use DHCP	No
Subnet Check for Multi MAC Devices	On
Certificate information	
Organization	PHOENIX CONTACT GmbH & Co. KG
Organizational Unit	
Locality	Blomberg
State or Province	Nordrhein-Westfalen
Country	DE

Рисунок 21 – Конфігурація OPC.

Щоб вивести наші змінні з вбудованої системи на базі OPC сервера, необхідно дізнатися їх правильну назву. Ця процедура виконується завдяки комплекту програмного забезпечення PC WORX SRT, а саме у «OPC Test Client»[15]. Це програмне забезпечення має наступний вигляд:

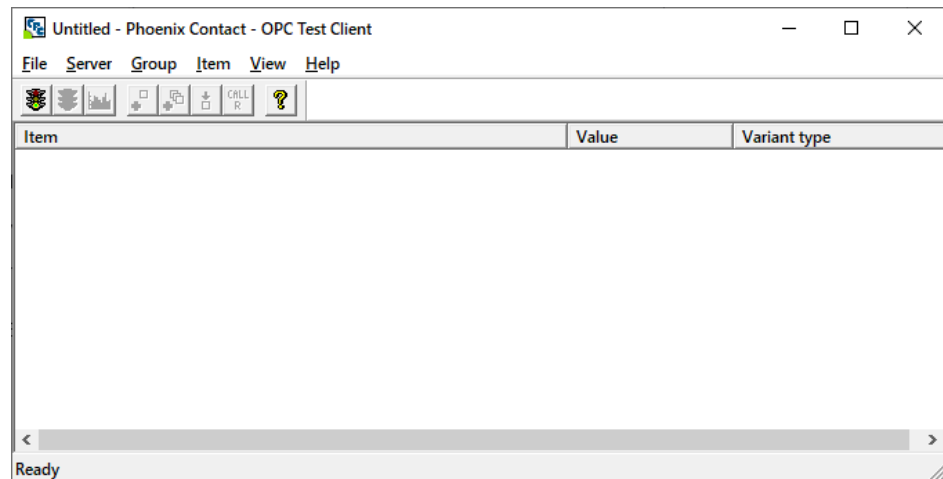


Рисунок 22 – Вигляд ПЗ OPC Test Client.

У цьому ПЗ ми спочатку підключаємося до вбудованого на базі OPC серверу через «Server»>«Connect». Та дізнаємося назви наших змінних через «Group»>«Add Item». У новому вікні обираємо наші змінні та вносимо їх назви до коду чат-бота.

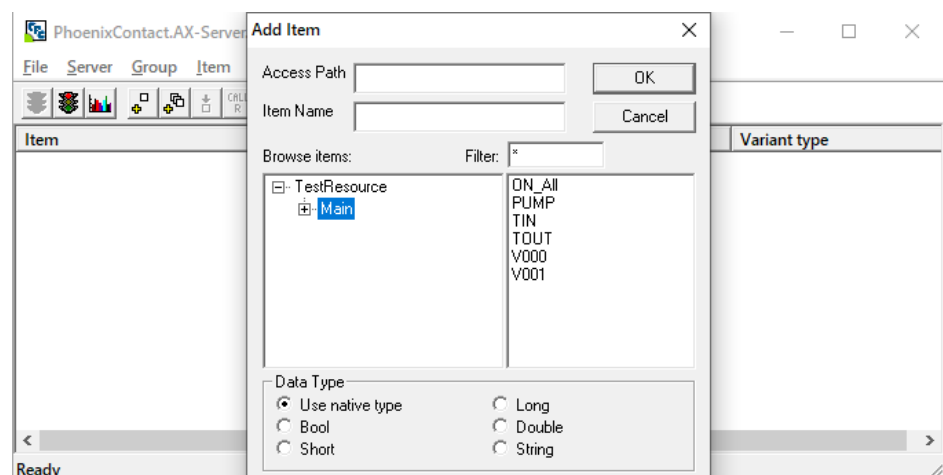


Рисунок 23 – Додавання змінних.

The screenshot shows a window titled "PhoenixContact.AX-Server.21 - Phoenix Contact - OPC Test Client". The window has a menu bar with "File", "Server", "Group", "Item", "View", and "Help". Below the menu bar is a toolbar with several icons, including a tree view, a refresh icon, a call button labeled "CALL", and a help icon. The main area of the window contains a table with three columns: "Item", "Value", and "Variant type". The table lists four items, all with a value of "Bad".

Item	Value	Variant type
TestResource.Main.ON_All	Bad	VT_BOOL
TestResource.Main.PUMP	Bad	VT_UI4
TestResource.Main.TIN	Bad	VT_UI4
TestResource.Main.TOUT	Bad	VT_UI4

At the bottom of the window, there is a status bar that says "Ready".

Рисунок 24 – Назви змінних.

6. ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

Перевіримо написаного нами чат-бота у мережі Telegram для вбудованої системи на базі OPC сервера. Для початку запусимо проект у PC WORX та ввійдемо у режим відладки. Запустимо проект та зупинимо, щоб зберегти деякі данні з змінних.

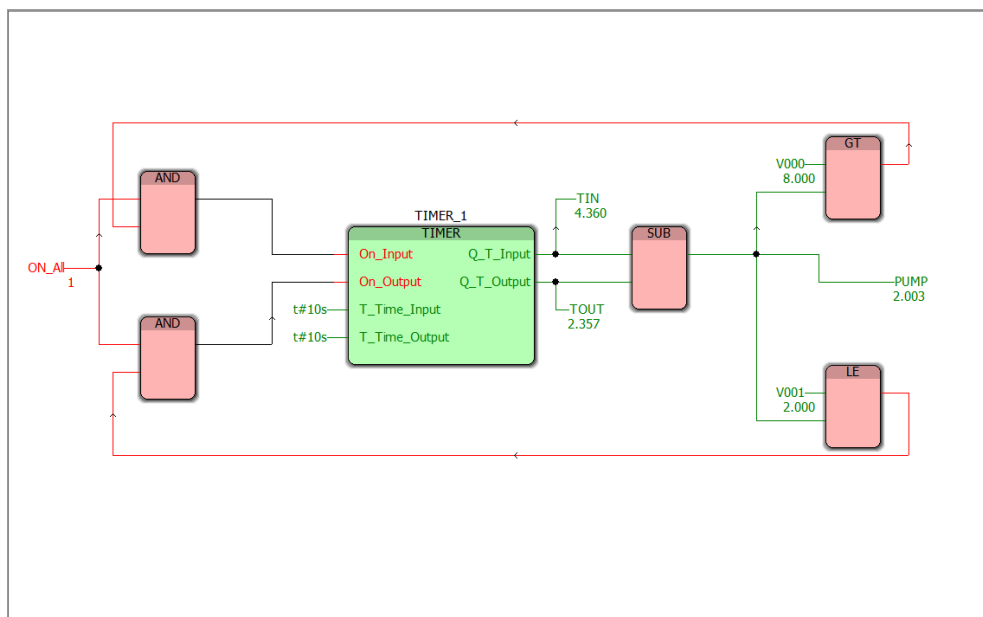


Рисунок 25 – Проект у PC WORX на стадії відладки

Як ми бачимо проект повністю працює та відсилає данні до вбудованої системи на базі OPC серверу. Це ми можемо побачити у OPC Test Client:

Item	Value	Variant type
TestResource.Main.ON_All	-1	VT_BOOL
TestResource.Main.PUMP	2003	VT_UI4
TestResource.Main.TIN	4360	VT_UI4
TestResource.Main.TOUT	2357	VT_UI4
TestResource.Main.V000	8000	VT_UI4
TestResource.Main.V001	2000	VT_UI4

Рисунок 26 – Данні на OPC сервері.

Для перевірки ми запускаємо розробленого чат-бота та далі по запиту у чат-боті ми можемо вивести цю інформацію:

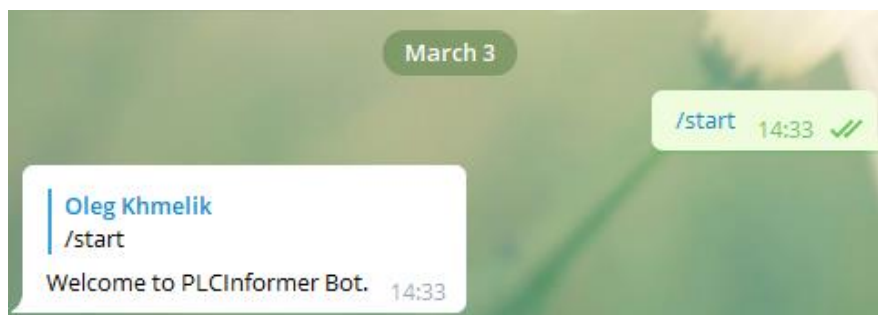


Рисунок 27 – Обробка стартової команди.

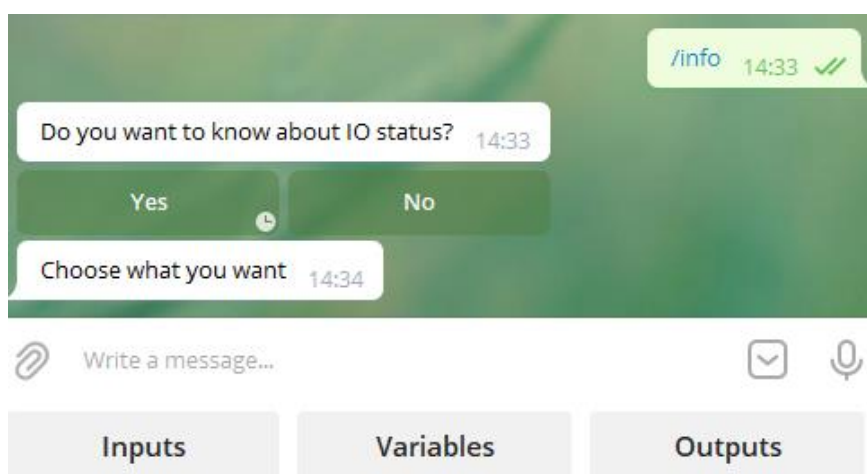


Рисунок 28 – Обробка повідомлень й кнопок.



Рисунок 29 – Вивід інформації по кнопці Inputs.



Рисунок 30 – Вивід інформації по кнопці Outputs.



Рисунок 31 – Вивід інформації по кнопці Variables.

Як ми бачимо написаний чат-бот, який ми розробили, повністю функціонує та виконує поставлену задачу з максимальною ефективністю.

7. ВИСНОВКИ

За виконану роботу можна з упевненістю сказати, що дана розробка є унікальним способом взаємодії користувача по програмуванню вбудованими системам управління та обміном інформації між ними. В майбутньому розвиток даної технології може забезпечити віддалену роботу і користувачам, і розробникам вбудованих систем по їх програмуванню, моніторингу та відладеному відлагоджуванню.

8. ЛІТЕРАТУРА

1. Фортин Т., Хокинсон Б. OPC UA и роль стандартов связи в развитии промышленного Internet вещей // Автоматизация в промышленности. 2016. № 8.
2. Mahnke W., Leitner S.H., Damm M. OPC Unified Architecture. Berlin: Springer, 2009.
3. Веселуха Г.Л. Промышленный Интернет вещей – это легко и интересно! // Автоматизация в промышленности. 2016. № 8.
4. Langmann R. et al. Workshop: The TATU Lab & smart education //2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV). – IEEE, 2016. – С. 400-402.
5. ПВ Галкин, ВВ Гавриленко, АИ Менько Исследование дальности и скорости передачи данных по витой паре в промышленных сетях RS-485 и PROFIBUS // Харків: ХНУРЕ 2016
6. Reinhard Langmann, Yuliya Makarova, Leandro Rojas-Peña, Pavlo Galkin, Igor Klyuchnik, Viktoriya Voropaeva, Valerii Pozepaev, Lyubov Zinyuk, Rostislav Skrypyuk, Elena Shaporina, Volodymyr Shaporin, Vladlen Shapo, Sergii Gorb. 13th International Conference on Remote Engineering and Virtual Instrumentation (REV) – IEEE 2016/2/24 – С 400-402.
7. Pavlo Galkin, Lydmila Golovkina, Igor Klyuchnyk Analysis of single-board computers for IoT and IIoT solutions in embedded control systems // 2018/10/9 IEEE –С. 297-302
8. ВЛ Колосков, ИЮ Павлов, ЕБ Иванов - Системный администратор, 2016
9. SH Leitner, W Mahnke - ABB Corporate Research Center, 2006
10. MH Schwarz, J Börsök - 2013 XXIV International Conference , 2013
11. R Henssen, M Schleipen - Procedia Cirp, 2014 – Elsevier
12. MA Rosid, A Rachmadany, MT Multazam... - IOP Conference , 2018
13. АА Козлов, АВ Батищев - Территория науки, 2017
14. ДР Филонов, ВИ Тупикин - Заметки по информатике и математике, 2017
15. M Schleipen, SS Gilani, T Bischoff, J Pfrommer - Procedia Cirp, 2016

АНОТАЦІЯ

Обрана тема є актуальною, серед інших матеріалів не була виявлена подібна цій розробка, та якщо надалі розвивати її можна добитись повного контролю вбудованої системи зі смартфона.

Актуальною також є розробка програмування вбудованої системи управління яка дозволяє системі реального часу змінювати та фактично програмувати вбудовану систему управління на базі ПЛК.

Метою роботи була розробка чат-боту у мережі Telegram, який буде отримувати інформацію з вбудованої системи на базі OPC серверу та відправляти по запиті користувача необхідну інформацію.

Завданням як і метою виступала розробка чат-боту, який зможе зчитувати та відправляти інформацію користувачу по команді.

Першим, що було зроблено це написання стандартного чат-боту, який пересилав написане йому повідомлення. Наступним кроком була поставлена задача навчити його працювати з базами даних, які постійно оновлюються. Цей етап був нескладнішим попереднього. Після написання цієї версії було проведено тестування, у якому не було виявлено проблем.

Наступним кроком було створення проекту у середовищі PC WORX, та його емолювання. Після цього переписали бота для роботи з OPC сервером. Далі була невелика проблема, пов'язана з зчитуванням даних. Але і вона вирішилася. Після отримання змінних з OPC серверу було модернізовано бота для подальшої роботи. Після проводилося тестування, у якому не було виявлено проблем.

Загально робота була пов'язана зі створення системи по програмуванню вбудованої системи управління в реальному часі, розробки та програмуванням чат-боту, який зчитував та передавав та приймав данні, змінні програми з вбудованої системи певному користувачу, що дозволяло програмувати всю систему.

ДОДАТОК 1

Повний код написаного чат-бота

```
import telebot
import OpenOPC
from telebot import types

bot =telebot.TeleBot("1643672293:AAGpuDusP6s0LJGsvgRIZ96gpCTWFTGIAtE")

@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.reply_to(message, "Welcome to PLCInformer Bot.")

@bot.message_handler(commands=['info', 'IO'])
def get_IO_info(message):
    markup_inline = types.InlineKeyboardMarkup()
    item_yes = types.InlineKeyboardButton(text = "Yes", callback_data = "yes")
    item_no = types.InlineKeyboardButton(text = "No", callback_data = "no")

    markup_inline.add(item_yes, item_no)

    bot.send_message(message.chat.id, "Do you want to know about IO status?",
reply_markup = markup_inline )

@bot.callback_query_handler(func = lambda call: True)
def answer(call):
    if call.data == "yes":
        markup_reply = types.ReplyKeyboardMarkup(resize_keyboard = True)

        item_inputs = types.KeyboardButton("Inputs")
        item_variables = types.KeyboardButton("Variables")
        item_outputs = types.KeyboardButton("Outputs")
```

```
        markup_reply.add(item_inputs,item_variables ,item_outputs)
        bot.send_message(call.message.chat.id, "Choose what you want",
reply_markup = markup_reply )
```

```
    elif call.data == "no":
```

```
        pass
```

```
@bot.message_handler(content_types = ["text"])
```

```
def get_inputs(message):
```

```
    if message.text == "Inputs":
```

```
        opc = OpenOPC.client()
```

```
        opc.connect('PhoenixContact.AX-Server.21')
```

```
        On_all = opc.properties('TestResource.Main.ON_ALL', id=2)
```

```
        bot.send_message (message.chat.id, f" ON_ALL : {On_all}")
```

```
        opc.close()
```

```
    elif message.text == "Outputs":
```

```
        opc = OpenOPC.client()
```

```
        opc.connect('PhoenixContact.AX-Server.21')
```

```
        Pump = opc.properties('TestResource.Main.PUMP', id=2)
```

```
        bot.send_message (message.chat.id, f" PUMP : {Pump}")
```

```
        opc.close()
```

```
elif message.text == "Variables":
    opc = OpenOPC.client()
    opc.connect('PhoenixContact.AX-Server.21')

    Tin = opc.properties('TestResource.Main.TIN', id=2)
    Tout = opc.properties('TestResource.Main.TOUT', id=2)

    bot.send_message (message.chat.id, f" TIN : {Tin}")
    bot.send_message (message.chat.id, f" TOUT : {Tout}")

    opc.close()

bot.polling()
```