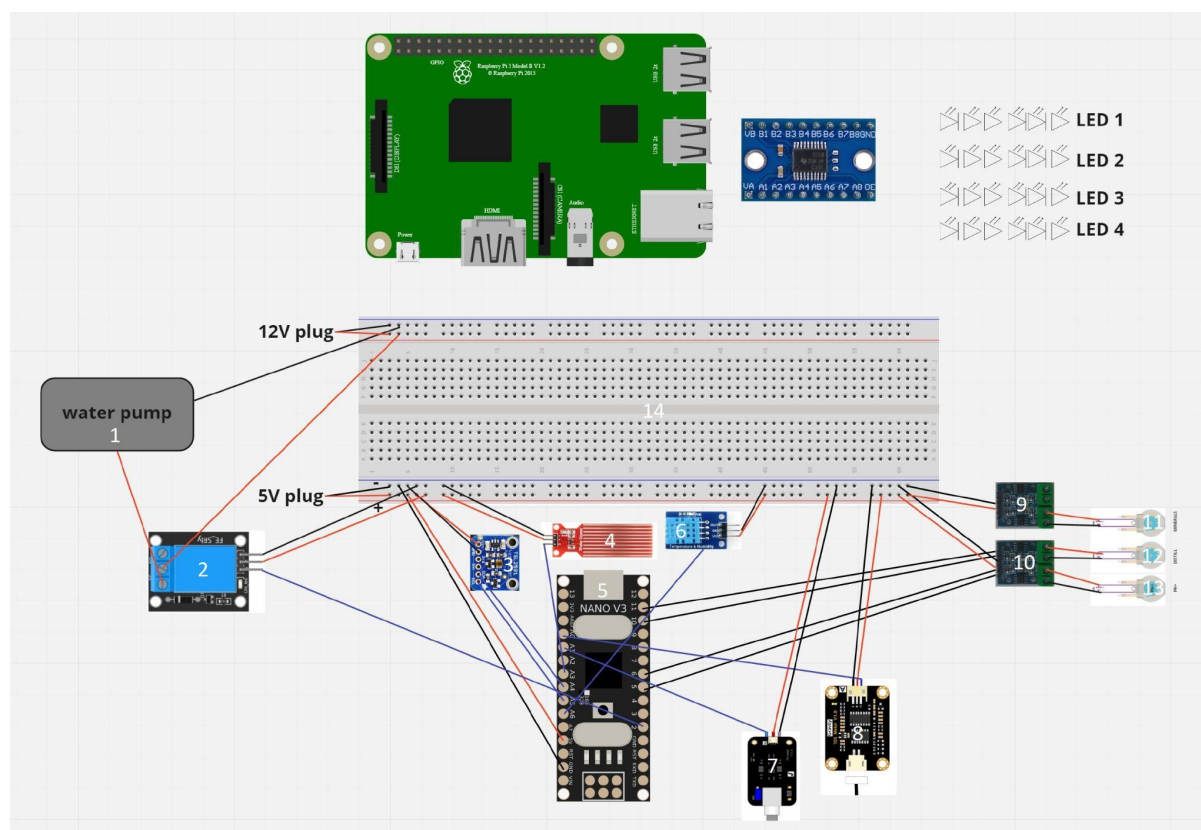


# Документация

## Содержание

1.	<a href="#">Обзор схемы</a>	1
2.	<a href="#">Руководство по программному обеспечению</a>	2
3.	<a href="#">Помпа</a>	10
3.1.	<a href="#">Схема подключения</a>	10
3.2.	<a href="#">Обзор управляющего класса</a>	11
4.	<a href="#">Насосы</a>	12
4.1.	<a href="#">Схема подключения</a>	12
4.2.	<a href="#">Обзор управляющего класса</a>	13
5.	<a href="#">ПИД регулятор</a>	14
6.	<a href="#">Датчик Tds</a>	15
6.1.	<a href="#">Схема подключения</a>	15
6.2.	<a href="#">Обзор управляющего класса</a>	16
7.	<a href="#">Датчик света (TSL)</a>	17
7.1.	<a href="#">Схема подключения</a>	17
7.2.	<a href="#">Обзор управляющего класса</a>	18
8.	<a href="#">Датчик Ph</a>	20
8.1.	<a href="#">Схема подключения</a>	20
8.2.	<a href="#">Обзор управляющего класса</a>	21
9.	<a href="#">Прочее</a>	22

# 1. Обзор схемы



На схеме цифрами пронумерованы основные элементы:

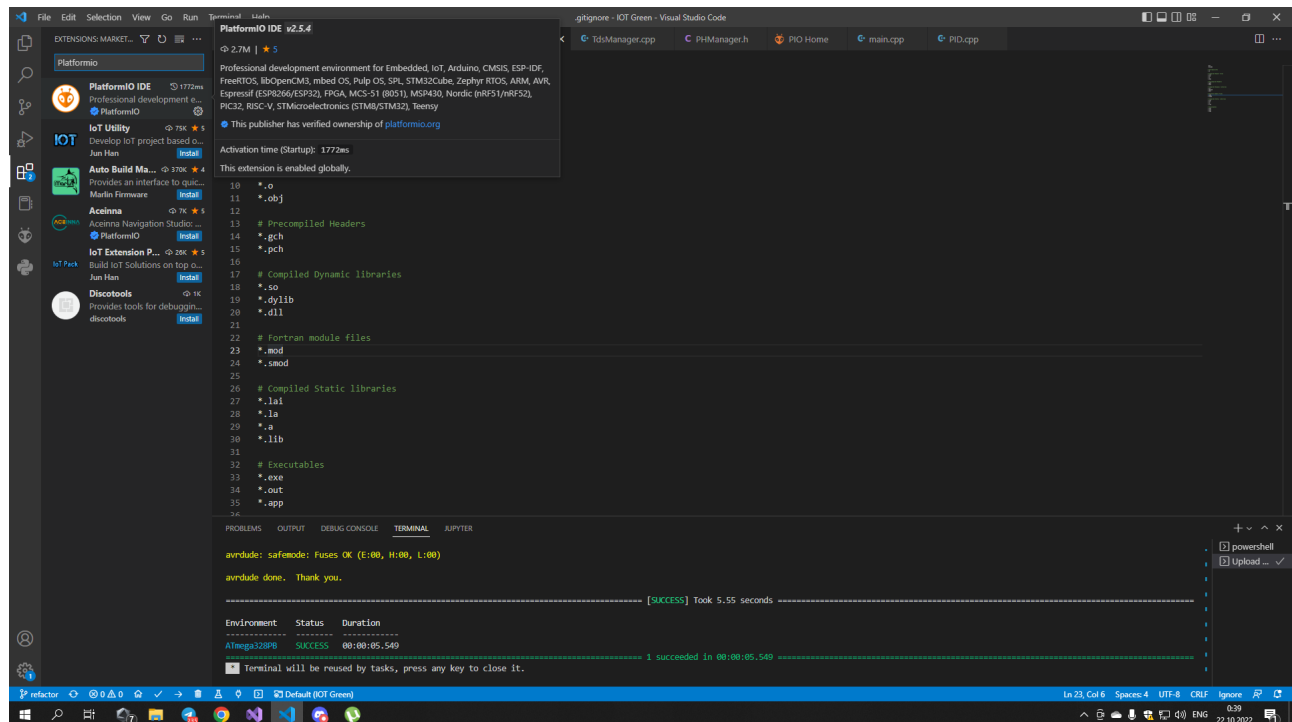
- 1 - насос для полива
- 2 - реле для водяного насоса
- 3 - датчик света tsl
- 4 - датчик уровня воды
- 5 - микроконтроллер
- 6 - датчик влажности и температуры
- 7 - датчик Ph уровня
- 8 - датчик индуктивности воды tds
- 9, 10 - реле для дозаторов
- 11 - дозатор для удобрений
- 12 - дозатор с дистиллированной водой
- 13 - дозатор с Ph+ водой
- 14 - макетная плата с включенными блоками питания на 5 и 12 вольт

В следующих разделах содержится подробное руководство по включению отдельных элементов и управляющие ими классы.

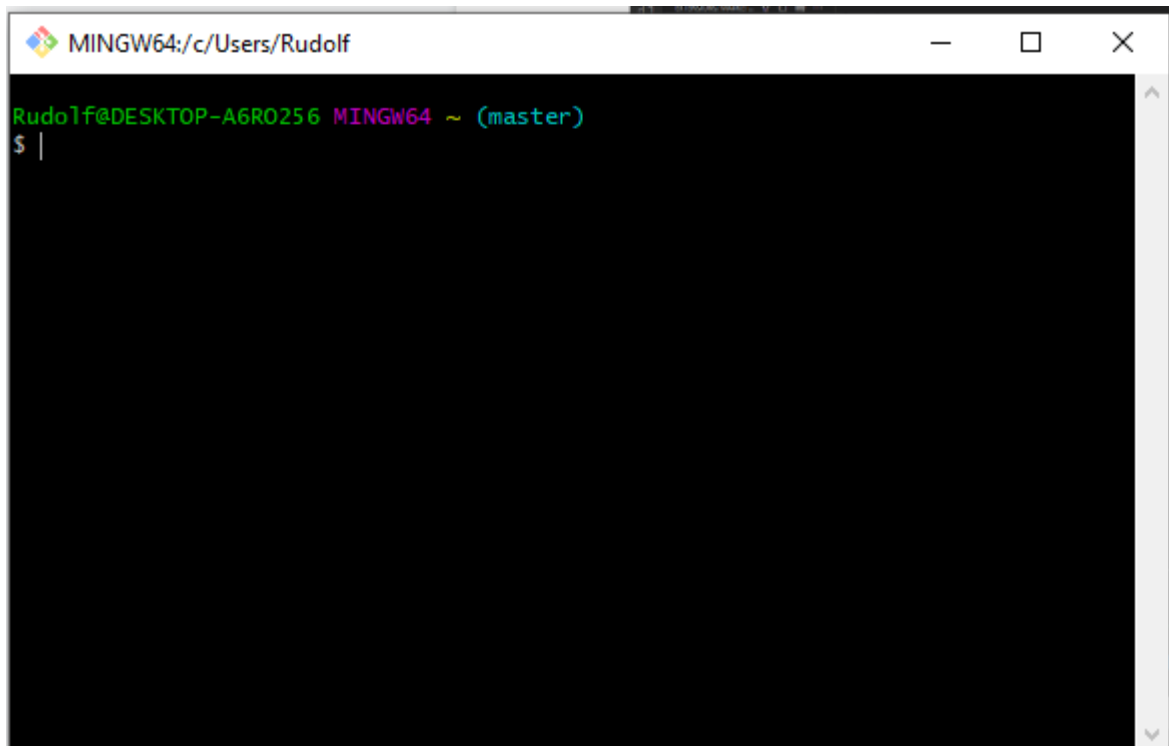
## 2. Руководство по программному обеспечению

В данном разделе описаны программы и их настройки, необходимые для запуска и работы кода.

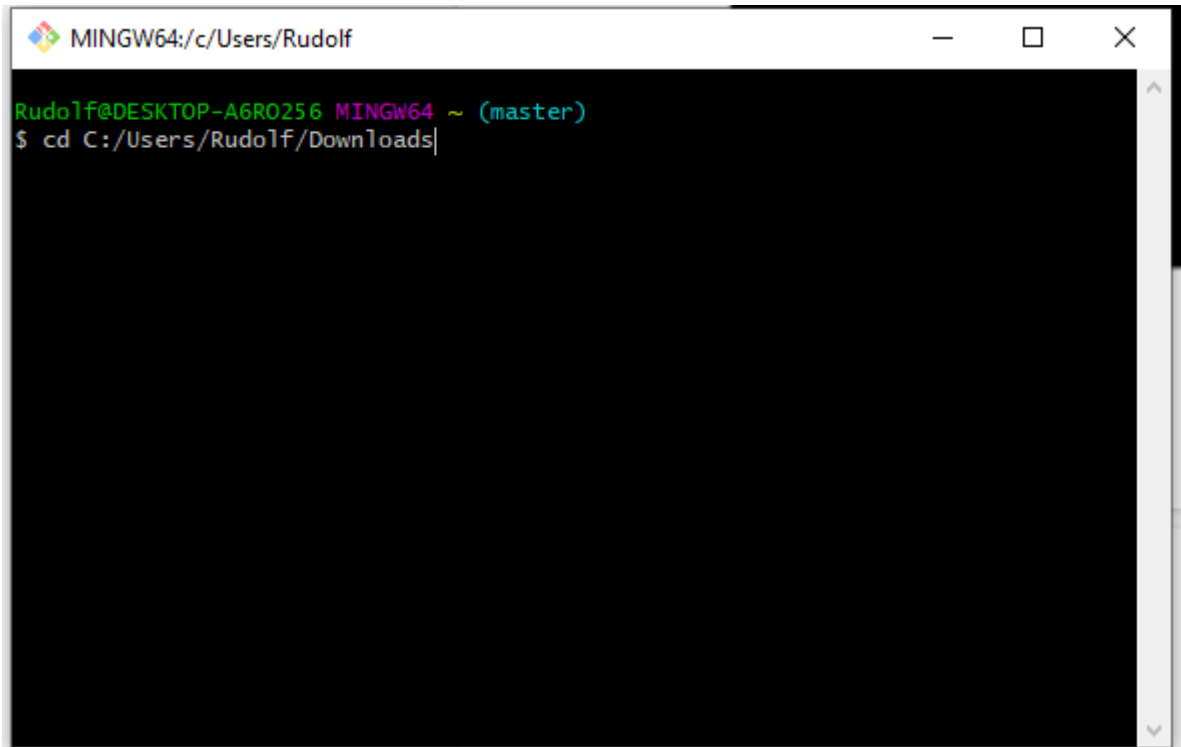
1. Скачать и установить VS Code.
2. Установить расширение Platformio (см. картинку)



3. Скачиваем и устанавливаем Git Bash:

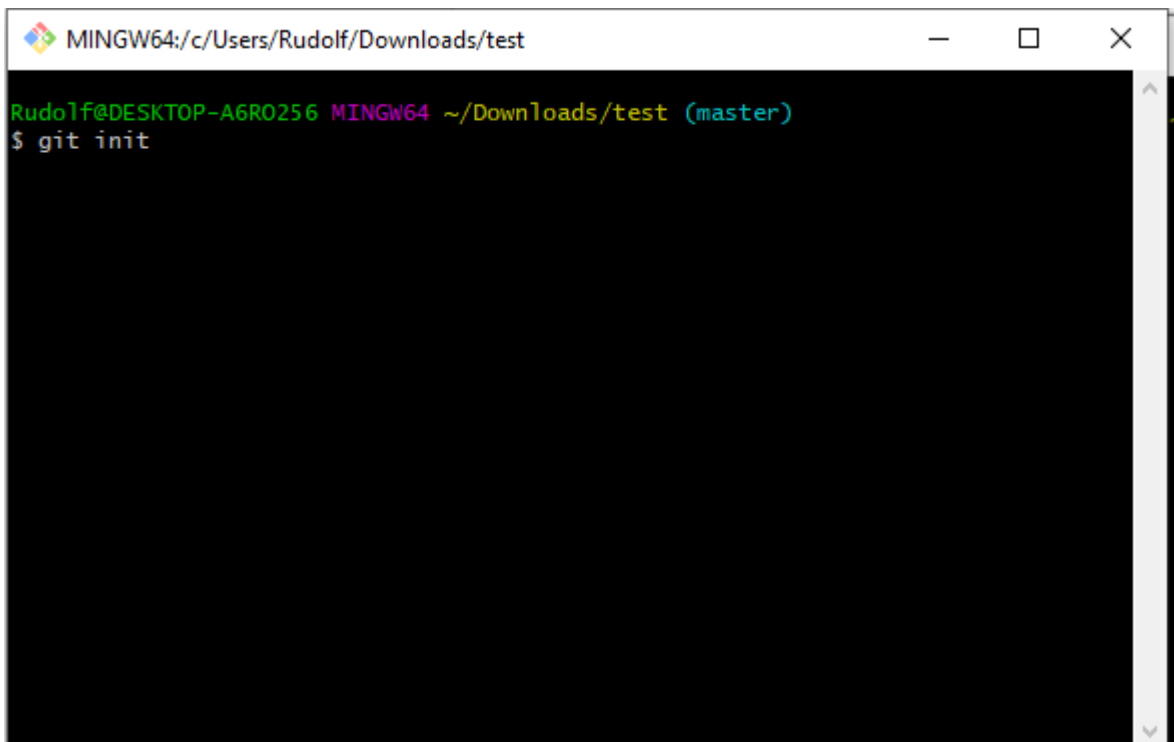


4. Теперь нужно подключить проект из Гитлаба. Регистрируемся на нем, если еще не, и начинаем генерировать ssh ключ. Для этого сначала локально на компьютере создаем папку, в которой будет лежать проект. Затем открываем Git Bash и вводим команду `cd <путь до директории с проектом>`:



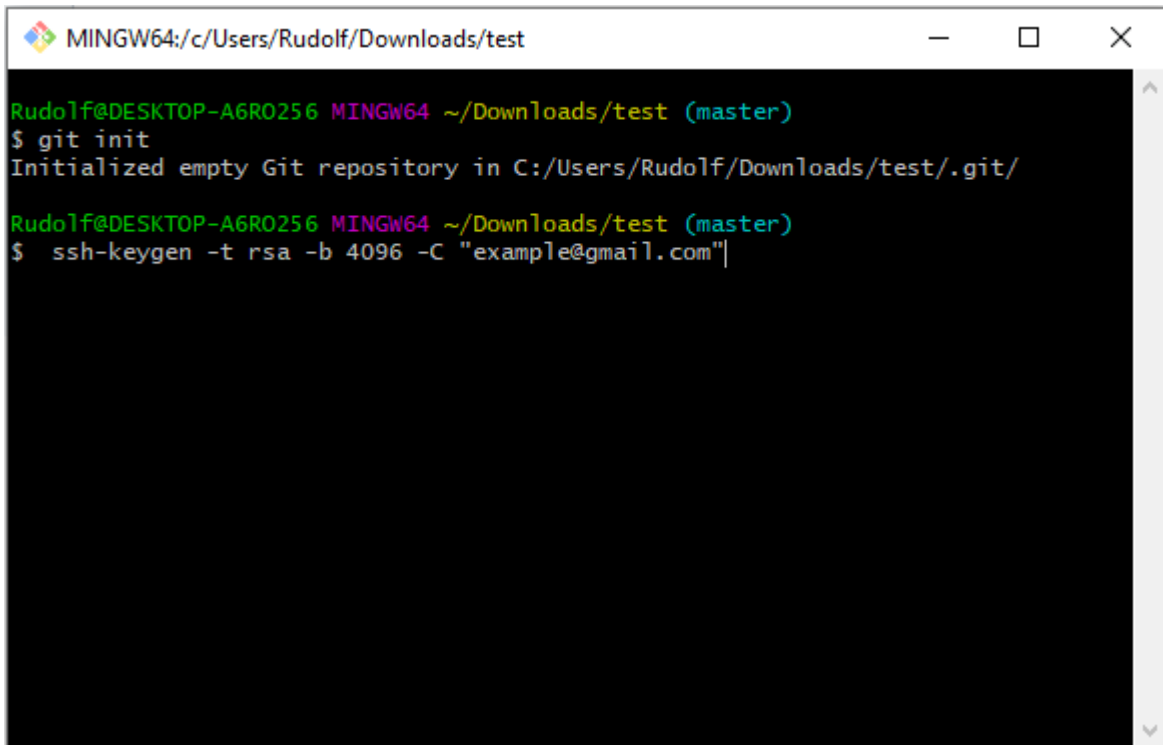
```
MINGW64:/c/Users/Rudolf  
Rudolf@DESKTOP-A6R0256 MINGW64 ~ (master)  
$ cd C:/Users/Rudolf/Downloads|
```

Далее вводим команду `git init`:



```
MINGW64:/c/Users/Rudolf/Downloads/test  
Rudolf@DESKTOP-A6R0256 MINGW64 ~/Downloads/test (master)  
$ git init
```

Таким образом, мы проинициализировали репозиторий гита, в котором будет храниться наш проект. Осталось сгенерировать ssh ключ и добавить его в настройки гитлаба. Для этого вводим следующую команду, в которой указываем почту, привязанную к аккаунту в Gitlab:



```
MINGW64:/c:/Users/Rudolf/Downloads/test
Rudolf@DESKTOP-A6R0256 MINGW64 ~/Downloads/test (master)
$ git init
Initialized empty Git repository in C:/Users/Rudolf/Downloads/test/.git/
Rudolf@DESKTOP-A6R0256 MINGW64 ~/Downloads/test (master)
$ ssh-keygen -t rsa -b 4096 -C "example@gmail.com"
```

После ввода команды система попросит ввести имя файла, в котором будет храниться ключ (можно оставить по умолчанию), затем попросит придумать пароль (passphrase). Придумываем и запоминаем его, он будет нужен всегда при внесении изменений.

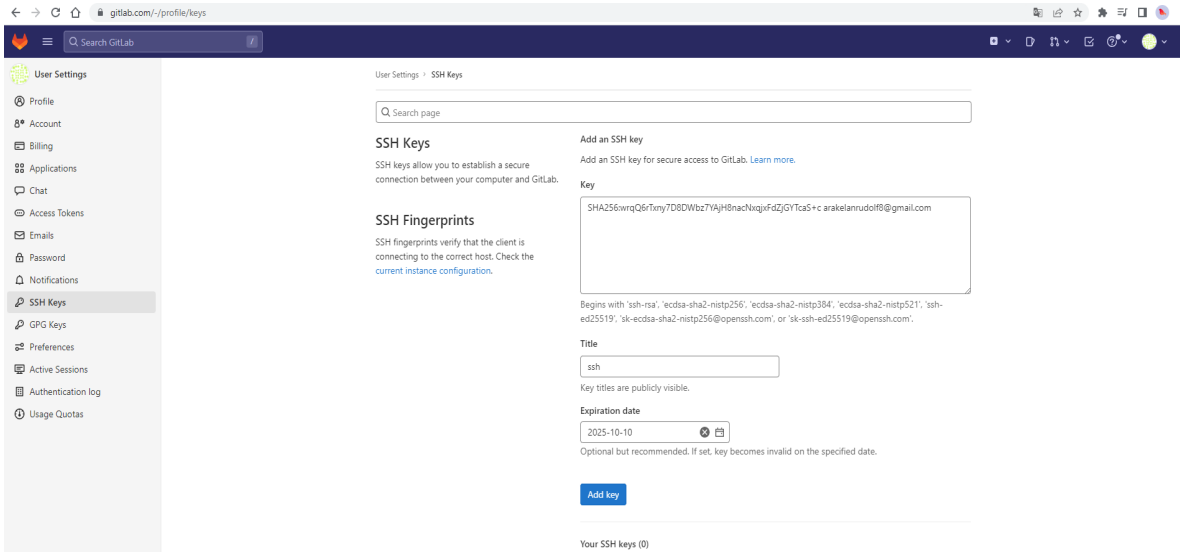
5. Осталось добавить ключ в аккаунт GitLab. После выполнения предыдущей команды, git выведет сгенерированный ключ:

```
MINGW64:/c:/Users/Rudolf/Downloads/test
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in test
Your public key has been saved in test.pub
The key's fingerprint is:
SHA256:wrqQ6rTxny7D8DWbz7YAjH8nacNxqjxFdZjGYTcaS+c arake1anrudolf8@gmail.com
The key's randomart image is:
+---[RSA 4096]---+
|      .==+      |
|      o*B..      |
|      oo.E       |
|   o  o         |
|  . o..o.S       |
|  ...o==.        |
|  o*..+X+.       |
|  ..=B++B.       |
| oo +B+o+.       |
+---[SHA256]-----+

Rudolf@DESKTOP-A6R0256 MINGW64 ~/Downloads/test (master)
$ ^C

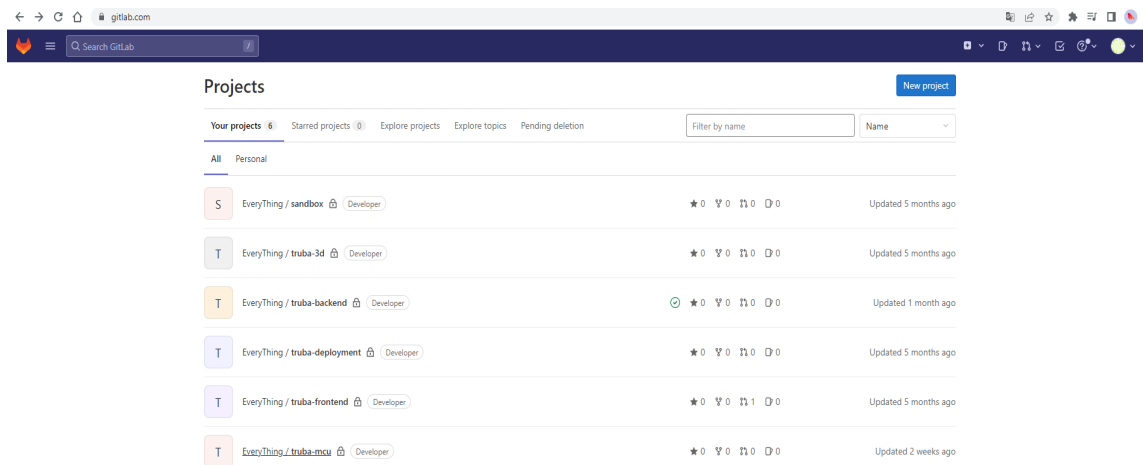
Rudolf@DESKTOP-A6R0256 MINGW64 ~/Downloads/test (master)
$ |
```

Копируем его и вставляем в специальное окошко в настройках Gitlab:

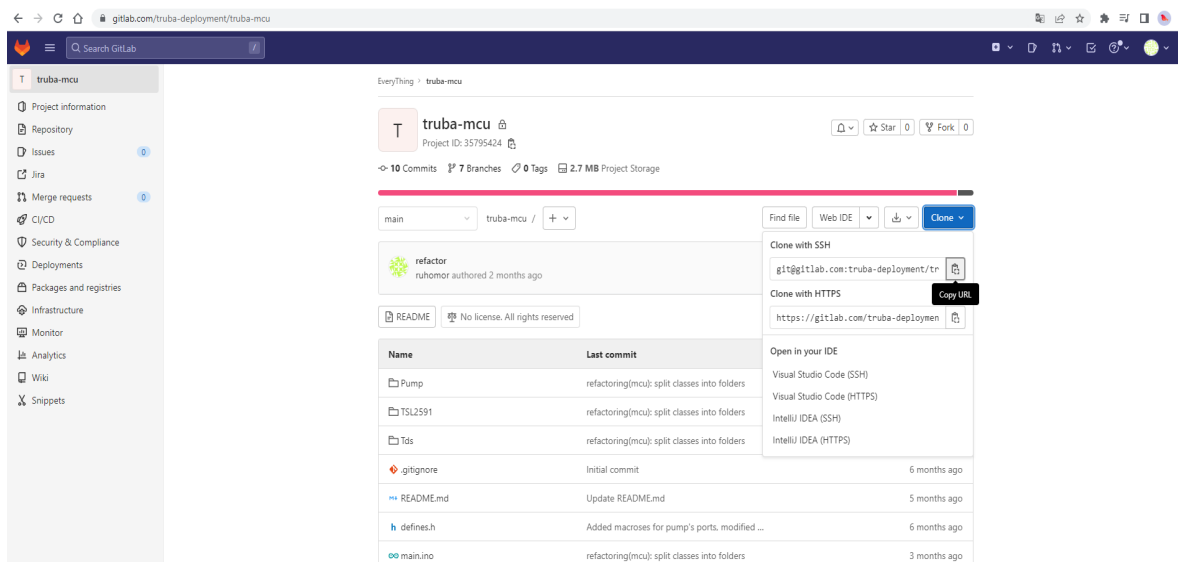


Придумываем название, переставляем expiration date на пару лет вперед и жмем “add key”.

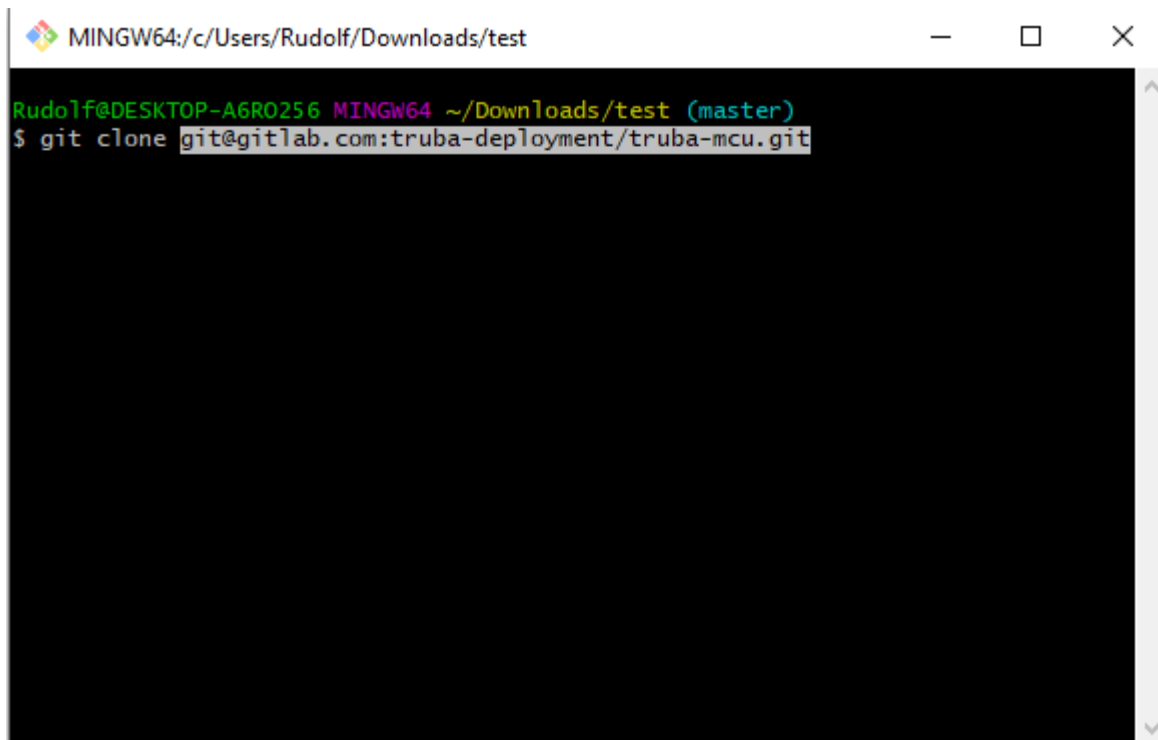
6. Таким образом, мы получили доступ к проекту на Гитлабе и можем скачать его локально. Заходим в папку truba-тси в Гитлабе:



Нажимаем на кнопку “Clone” и копируем ссылку ssh:



Теперь, находясь в git bash всё в той же папке под проект, вводим команду `git clone` и вставляем скопированную ссылку:

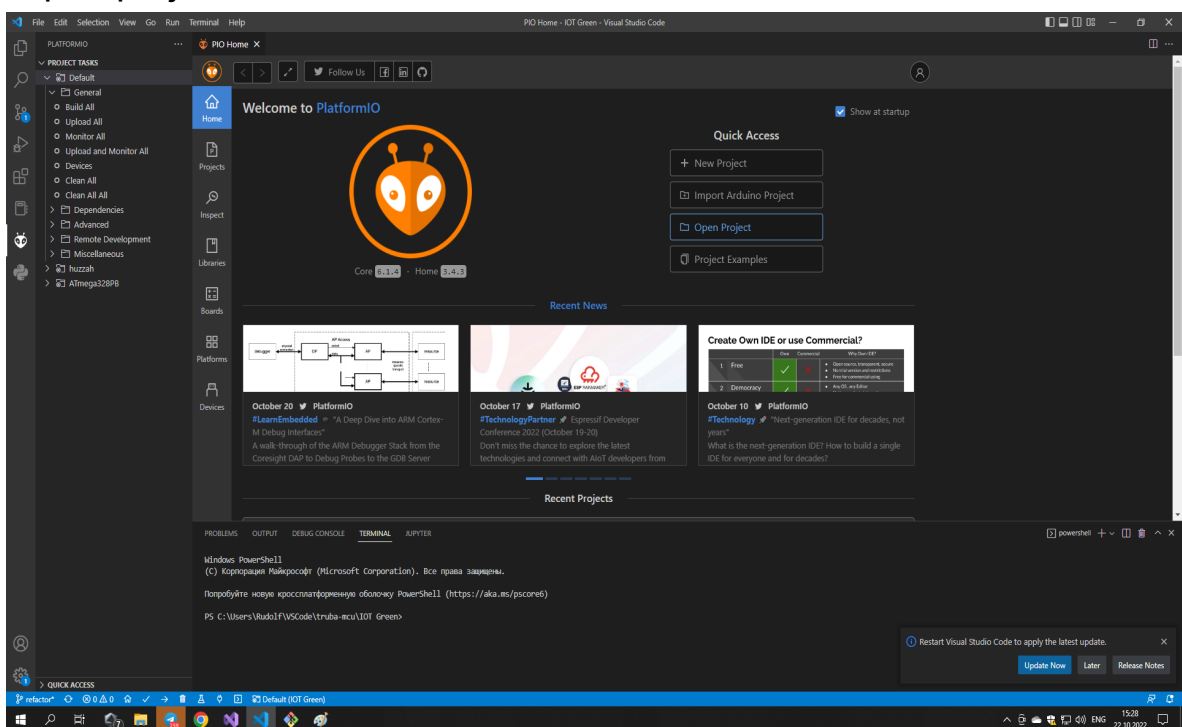


```
MINGW64:/c/Users/Rudolf/Downloads/test

Rudolf@DESKTOP-A6R0256 MINGW64 ~/Downloads/test (master)
$ git clone git@gitlab.com:truba-deployment/truba-mcu.git
```

Затем вводим придуманный ранее пароль, и получаем проект локально на компьютере.

7. Следующий шаг - открытие проекта в VS Code. Заходим на домашнюю страницу расширения Platformio и жмем кнопку "Open project":

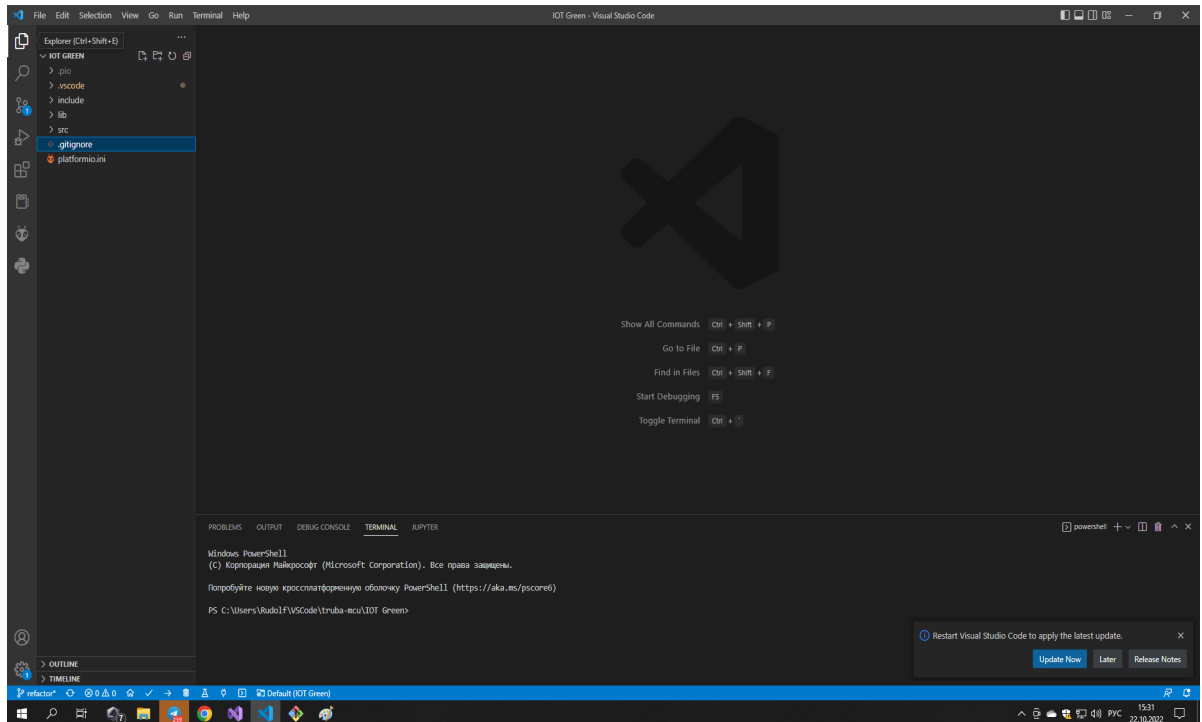


Здесь указываем путь до скачанного проекта и открываем его. В



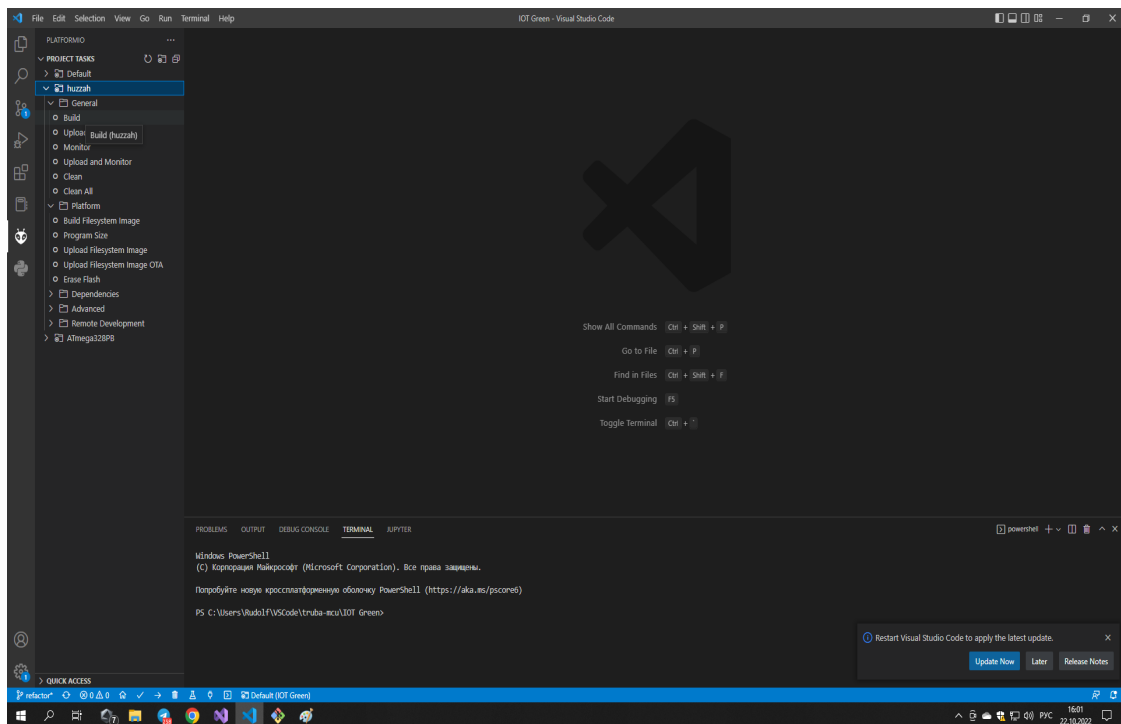
первый раз нужно будет немного подождать открытия, затем перезапустить VS Code.

8. Готово, наш проект открыт. Посмотрим, как с ним работать. Переходим слева на иконку “Explorer” и видим все файлы и папки проекта:



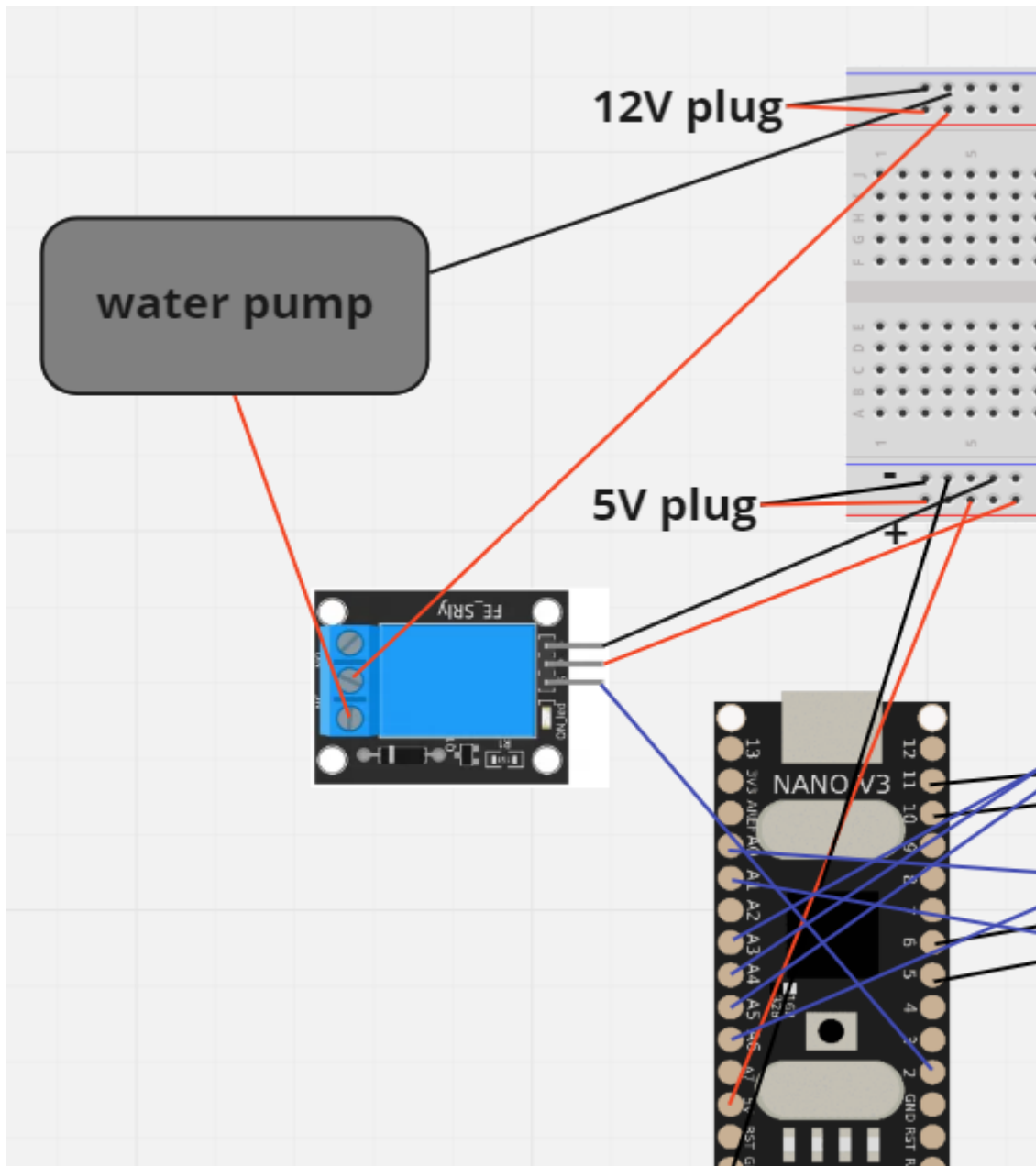
Внизу слева на голубой панели можно переключаться между ветками проекта. Для того чтобы скомпилировать весь проект, можно на этой же панели нажать галочку. Чтобы загрузить его на микроконтроллер, подключаем его по usb к компьютеру и нажимаем на значок стрелки вправо. Для компиляции и загрузки кода на конкретный микроконтроллер, нужно перейти на значок Platformio слева, развернуть нужный список и нажать кнопку

build или upload соответственно:



## 3. Помпа

### 3.1. Схема подключения



Минусовой контакт насоса (1) вставляется напрямую в макетную плату в минус 12-вольтового блока питания. Плюсовой контакт вставляется одно из гнезд реле (2), от которого из соседнего гнезда идет провод в плюс на макетной плате. Таким образом, когда ключ в реле соединяет провода, схема замыкается и насос работает.

Плюс и минус реле соединены с соответствующими плюсом и минусом 5-вольтового блока питания. Третий провод, отвечающий за управление подачей напряжения, подключен к порту 2 микроконтроллера.

### 3.2. Обзор управляющего класса

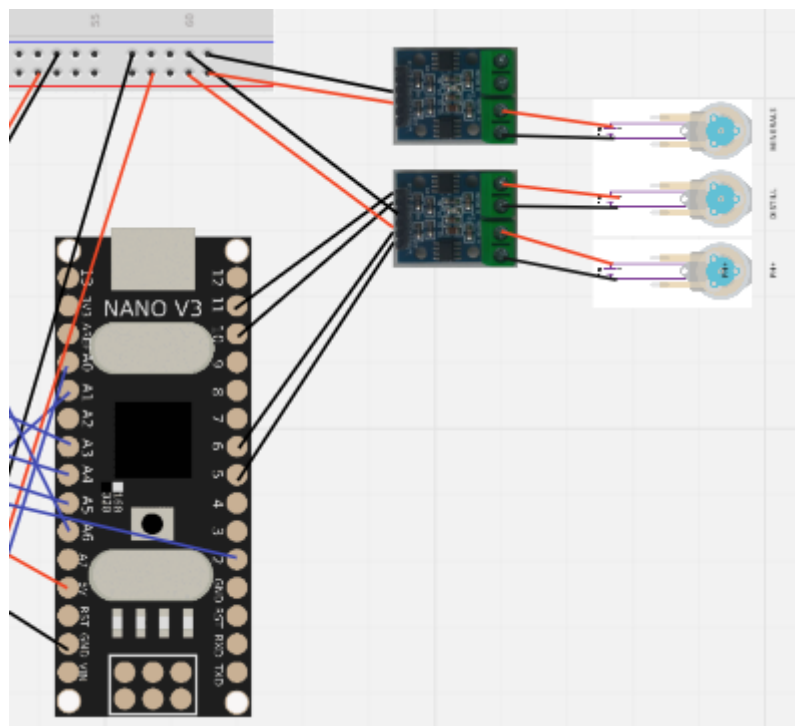
```
4  class Pomp
5  {
6  public:
7      Pomp(uint8_t port);
8
9      void Execute(uint32_t restTime, uint32_t workTime); // time in seconds
10
11 private:
12     uint8_t _port;
13 };
```

Конструктор **Pomp** запоминает порт, к которому подключен насос и открывает его.

Функция **Execute** в качестве параметров принимает время простоя (restTime) и время работы (workTime) в секундах. Соответственно, при выполнении функции насос качает воду workTime секунд, затем не работает restTime секунд.

## 4. Насосы

### 4.1. Схема подключения



12 и 13 насосы регулируют уровень Ph растения. Они подключаются к двойному реле 10 как показано на рисунке. От реле для каждого насоса идут по 2 провода с управляющим напряжением. Их мы подключаем к портам 10, 11 и 5, 6 соответственно. Эти порты обозначены символом “~”, означающим возможность регулировать величину напряжения, и, следовательно, скорость работы насосов. Так же от реле идут провода + - к соответствующим разъемам 5-вольтового блока питания на макетной плате.

Насос 11 нужен для подкачки удобрений. Он подключается аналогично насосам 12, 13 к отдельному реле (9). Порты, управляющие напряжением 11 насоса - 3 и 9.

## 4.2. Обзор управляющего класса

```
4 class Pump
5 {
6 public:
7     Pump(uint8_t port1, uint8_t port2);
8
9     void SetSpeed(float); // Устанавливает скорость вращения мотора числом [0; 165]
10    // (ниже 90 насос не качает, поэтому значения из [90; 255] смещены к нулю)
11
12 private:
13     const uint8_t _port1;
14     const uint8_t _port2;
15 };
```

Конструктор **Pump** запоминает порты, к которым подключен насос и открывает их.

Функция **SetSpeed** устанавливает скорость вращения мотора в диапазоне [0; 165].

## 5. ПИД регулятор

```
3  struct PID
4  {
5      Pump _saltPump;
6      Pump _freshPump;
7
8      PID(Pump pump1, Pump pump2) : _saltPump(pump1),
9      | | | | | | | | _freshPump(pump2){};
10
11     float _coeff = 1.0;
12     float _derive = 0.0;
13
14     void PIDSetup(float coeff = 1);
15
16     void PIDLoop(float value, float target, float eps);
17 };
```

ПИД регулятор - устройство для плавного регулирования целевого показателя датчика (например Ph или Tds уровня) в реальном времени.

За его реализацию в нашем проекте отвечает структура PID (на рисунке).

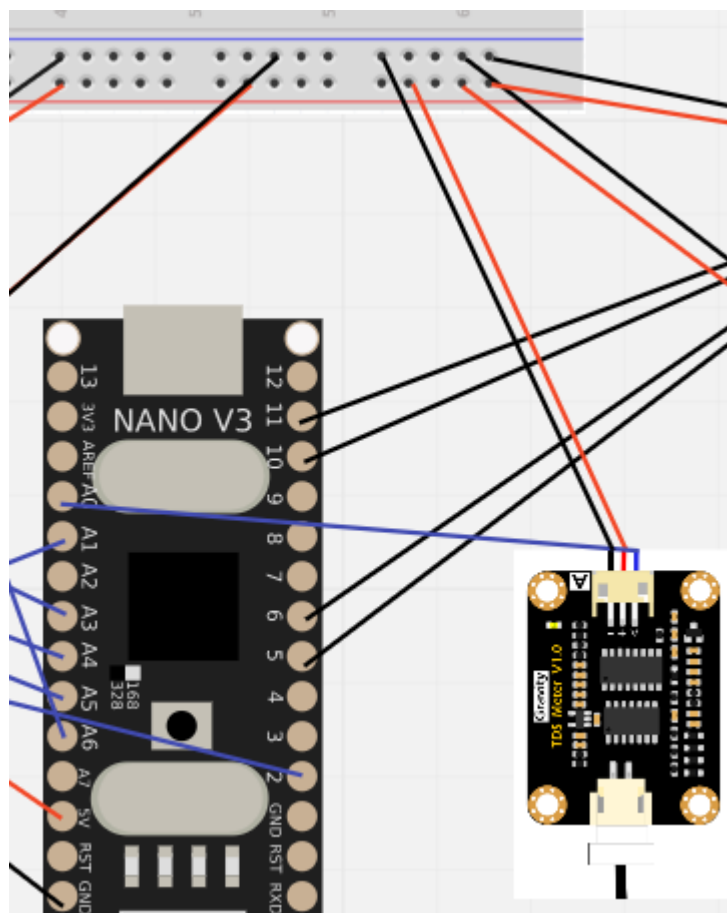
Конструктор **PID** принимает в качестве аргументов 2 насоса (**\_saltPump** и **\_frashPump**): один отвечает за понижение отслеживаемого показателя, второй - за повышение.

Метод **PIDSetup** вызывается единожды и устанавливает коэффициент пид регулятора.

Метод **PIDLoop** вызывается в цикле, принимает 3 аргумента: текущее значение с сенсора (**value**), целевое значение (**target**) и допустимую погрешность (**eps**) достижения целевого значения. Внутри метода при каждом вызове вычисляется скорость и направление подкачки вещества в емкость с растением.

## 6. Датчик Tds

### 6.1. Схема подключения



На картинке изображена схема подключения датчика индуктивности (солёности) Tds. 2 разъёма предназначены для тока и земли, идут в соответствующие узлы 5-вольтового источника питания на макетной плате.

Третий разъём, предназначенный для передачи данных, идет в порт A0 микроконтроллера.



## 6.2. Обзор управляющего класса

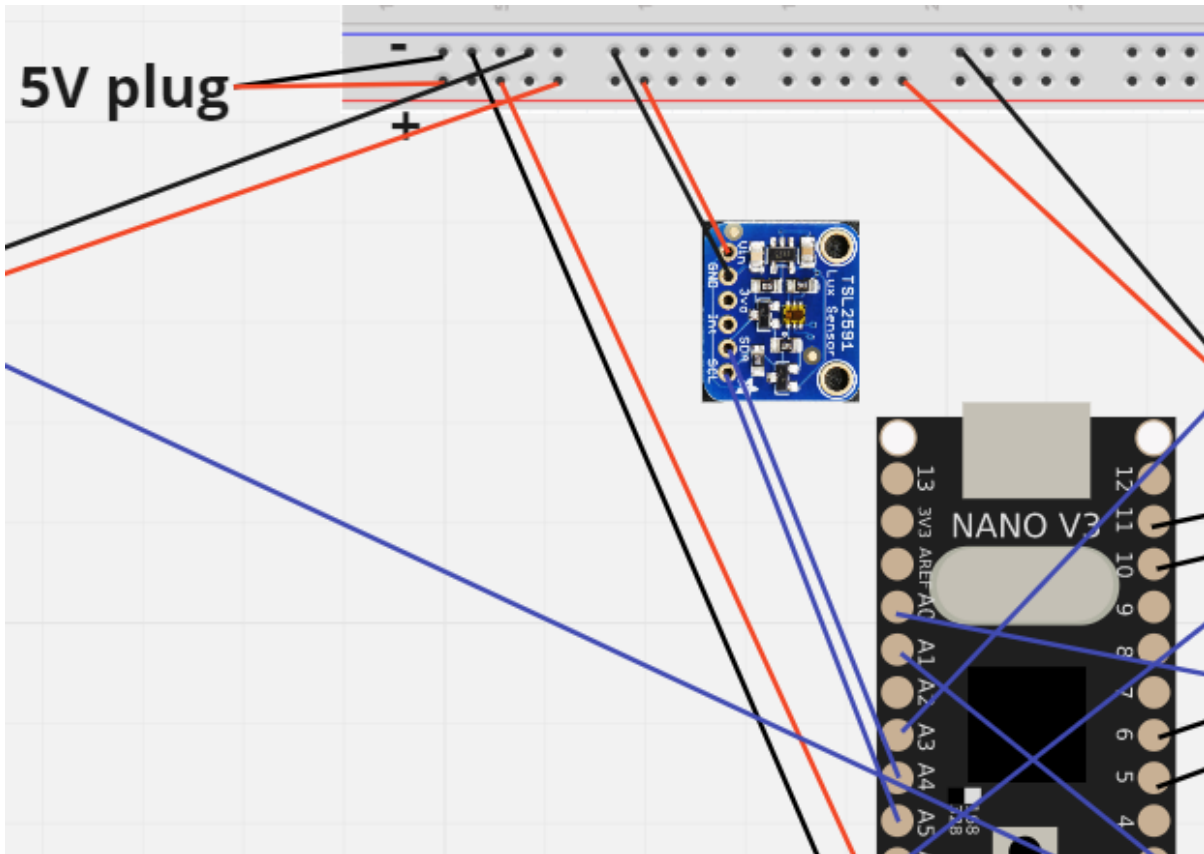
```
12 class Tds
13 {
14 public:
15     explicit Tds(uint8_t port);
16
17     float GetValue() const; // Считывает значение напряжения с сенсора и переводит в значение tds
18
19 private:
20     PID _pid = { Pump(SALT_PUMP_PORT1, SALT_PUMP_PORT2), Pump(FRESH_PUMP_PORT1, FRESH_PUMP_PORT2) };
21
22     const uint8_t _port;
23 };
24
```

Управляющий класс для Tds датчика представлен на рисунке. Конструктор принимает порт для считывания данных и сохраняет его.

Метод ***GetValue*** считывает данные с датчика и возвращает их. Поле класса ***PID*** нужно для применения ПИД регулятора при контролировании значения tds.

## 7. Датчик света (TSL)

### 7.1. Схема подключения



На картинке представлена схема подключения датчика света TSL2591. Провода от разъемов Vin и gnd подключаются к + и - 5-вольтового источника питания соответственно.

Датчик работает по i2c протоколу соединения, поэтому разъем на датчике SDA идет в порт A4, выполняющий также функцию SCL, а от SCL - к A5 (SCL).

## 7.2. Обзор управляющего класса

```
6  class TSL
7  {
8  public:
9      explicit TSL(size_t id);
10
11      bool Init();
12      bool Begin();
13
14      void DisplaySensorDetails();
15      void ConfigureSensor(tsl2591Gain_t gain,
16                          tsl2591IntegrationTime_t integrationTime);
17
18      void SimpleRead(uint8_t mode);
19      void AdvancedRead();
20      void UnifiedSensorAPIRead();
21
22 private:
23     void messageOk() noexcept;
24     void messageError() noexcept;
25
26 private:
27     Adafruit_TSL2591 _tsl;
28 };
```

На рисунке представлен класс управления датчиком `tsl`, написанный поверх сторонней библиотеки.

Конструктор ***TSL*** принимает `id`, присваиваемый датчику и запоминает его.

Функция ***Init*** инициализирует датчик и выводит сообщение об успешности операции.

Функция ***Begin*** запускается в цикле. Начинает работу датчика и в случае неисправности выводит сообщение об ошибке.

Функция ***DisplaySensorDetails*** выводит сообщение с информацией о датчике.

***ConfigureSensor*** выставяет коэффициент чувствительности датчика (`gain`) и скорость реагирования на изменение освещённости (`integrationTime`).

***SimpleRead*** в зависимости от `mode` выводит инфракрасный, видимый или полный спектр света.

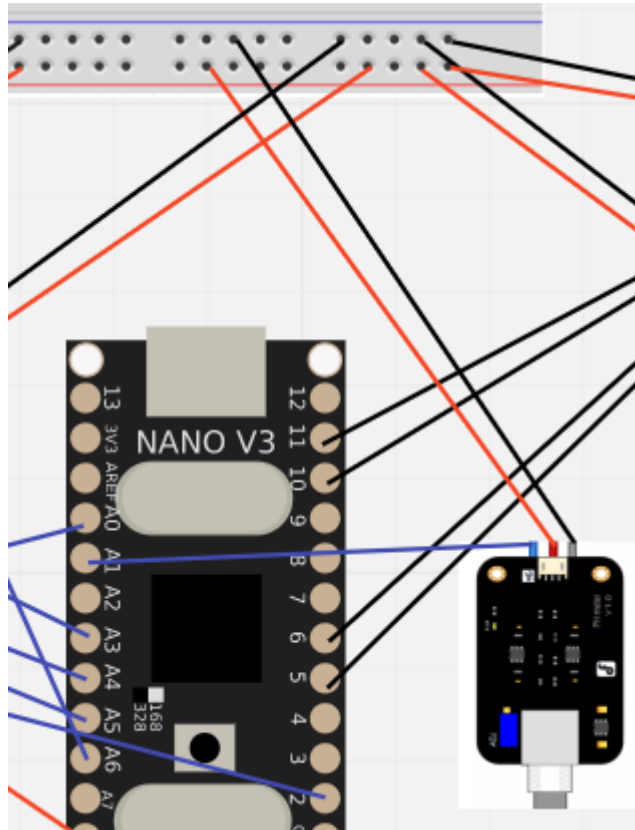
***AdvancedRead*** выводит все спектры света и интенсивность в люксах.

***UnifiedSensorAPIRead*** выводит интенсивность света в люксах.

***messageOK*** и ***messageError*** - функции отладочных сообщений.

## 8. Датчик Ph

### 8.1. Схема подключения



На картинке изображена схема подключения датчика Ph. 2 разъёма предназначены для тока и земли, идут в соответствующие узлы 5-вольтового источника питания на макетной плате.

Третий разъём, предназначенный для передачи данных, идет в порт A1 микроконтроллера.

## 8.2. Обзор управляющего класса

```
5  class PHManager
6  {
7      public:
8          explicit PHManager(uint8_t port);
9
10         float GetValue(int measureCount);
11
12     private:
13         uint8_t _port;
14 };
```

Управляющий класс для Ph датчика представлен на рисунке. Конструктор принимает порт для считывания данных и сохраняет его.

Метод ***GetValue*** считывает данные с датчика и возвращает их.

## 9. Прочее

В файле ***defines.h*** содержатся общие константы, например целевой уровень кислотности pH или уровень Tds.

Файлы ***Esp8266Config.h*** и ***IskraNanoConfig.h*** содержат специфическую информацию для каждого контроллера соответственно, такие как скорость чтения Serial монитора и порты, к которым подключены датчики.