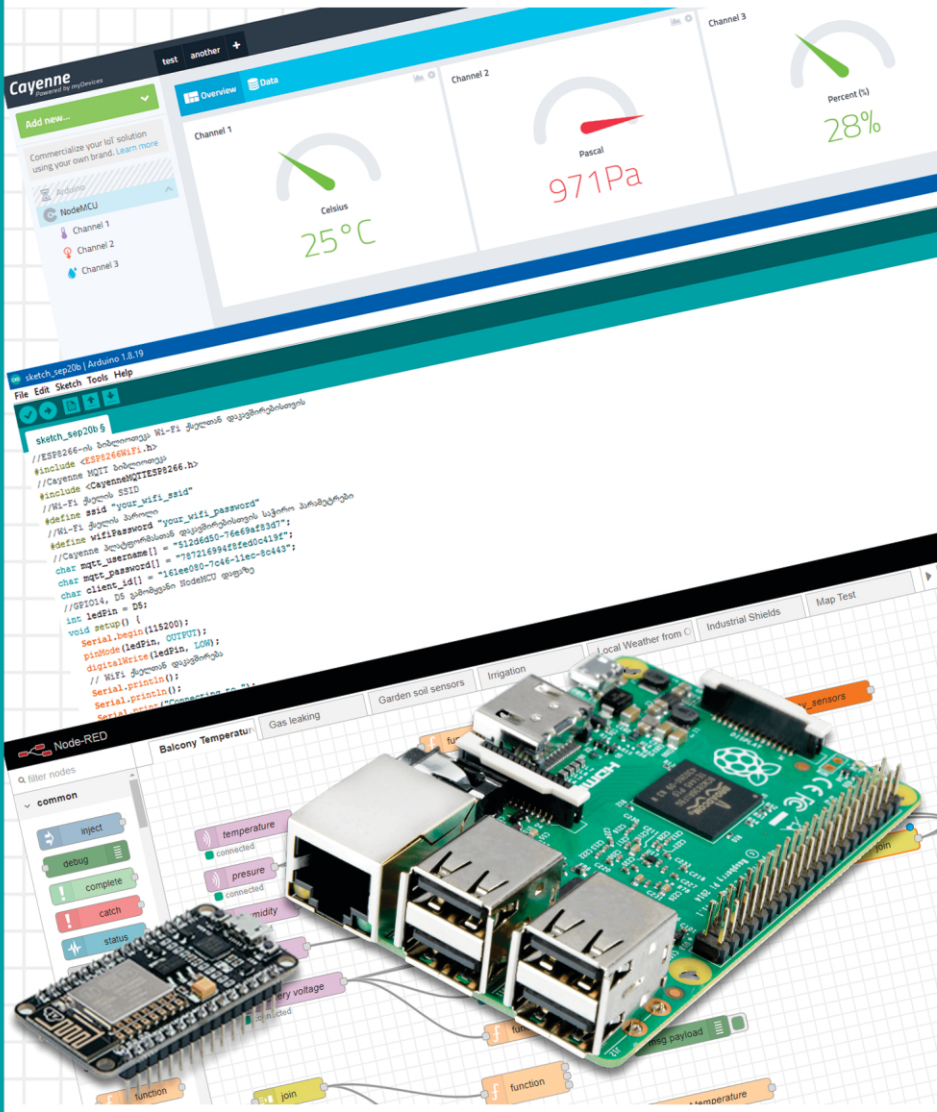
საგნების ინტერნეტი

IoT



მარტივად და გასაგებად

ვასა ტაბატაძე, თეა თოდუა

IBSU Publications

ზაზა ტაბატაძე, თეა თოდუა

საგნების ინტერნეტი
(IoT)
მარტივად და გასაგებად

თბილისი
2022

უაკ 004.4

ნაშრომი წარმოადგენს წიგნების სერიის „მიკროკონტროლერები დამწყებთათვის“ გამოცემის გაგრძელებას. სერიის პირველ წიგნში (არდუინო. პრაქტიკული სახელმძღვანელო დამწყებთათვის) დეტალურად არის განხილული ის საკითხები, რომელთა ცოდნაც აუცილებელია არდუინოს შესწავლის საწყის ეტაპზე. მეორე წიგნი განკუთვნილია მკითხველთა ფართო წრისთვის, ყველა იმ ადამიანისთვის, ვისაც სურს გაეცნოს საგნების ინტერნეტის საინტერესო სამყაროს და განახორციელოს სახლის სრული ავტომატიზაცია შედარებით იაფად, NodeMCU დაფის გამოყენებით. წიგნში მოცემული ამოცანების განხორციელება მკითხველს საშუალებას მისცემს მათ საფუძველზე, კიდევ უფრო დიდი და მასშტაბური პროექტები განახორციელოს.

რეცენზენტები:

საქართველოს ტექნიკური უნივერსიტეტი,
პროფესორი: ლ. იმნაიშვილი

შავი ზღვის საერთაშორისო უნივერსიტეტი,
პროფესორი: ი. როდონაია

საქართველოს ტექნიკური უნივერსიტეტი,
ასოცირებული პროფესორი მ. ბედინეიშვილი

საქართველოს ტექნიკური უნივერსიტეტი,
არჩილ ელიაშვილის სახელობის მართვის
სისტემების ინსტიტუტი, უფროსი მეცნიერ-
მუშაკი, ო. ქართველიშვილი

გამომცემელი: შავი ზღვის საერთაშორისო უნივერსიტეტი

ISBN 978-9941-8-4897-1

წინასიტყვაობა

„რაც კი რამ დაუბადია
უფალს სულიერ-უსულო, -
ყველასაც თურმე ენა აქვს,
არა ყოფილა ურჯულო.“

ვაჟა-ფშაველა. „გველისმჭამელი“

შვეიცარიელმა ინჟინერმა გუსტავ აიხელბერგმა XX საუკუნის 50-იან წლებში გამოცემულ წიგნში „ადამიანი და ტექნიკა“ კაცობრიობის განვითარება, სამეცნიერო-ტექნიკური ევოლუციის თვალსაზრისით, ხატოვნად წარმოადგინა 60 კილომეტრიანი რბოლის სახით, სადაც თითოეული კილომეტრი 10 000 წელს აღნიშნავს: ამ წარმოსახვითი რბოლის უმეტესი ნაწილი უსიერ ტყეებზე გადის, მხოლოდ ბოლოს, 58-ე-59-ე კილომეტრებზე გამოქვაბულებში პირველყოფილი ადამიანების იარაღების გვერდით ჩანს ნახატებიც. დისტანციის უკანასკნელ კილომეტრზე ჩნდება მიწათმოქმედების ნიშნები. ფინიშამდე სამას მეტრში ქვის ფილებით მოგებულ გზას ეგვიპტის პირამიდებისა და ძველი რომაული ციხესიმაგრეებისკენ მივყავართ. ჩვენი მორბენ-ლების გზის უკანასკნელ ას მეტრზე მოჩანს შუასაუკუნეების ქალაქური ნაგებობანი. ფინიშამდე კიდევ ორმოცდაათი მეტრი რჩება. დგას ადამიანი და ჭკვიანი, გამჭრიახი თვალებით ადევნებს თვალს რბოლას. ეს ლეონარდო და

ვინჩია. დარჩა მხოლოდ 10 მეტრი, მორბენლები ჩირაღდნებისა და ზეთის ლამპების მბჟუტავ შუქზე მირბიან. გზის უკანასკნელ 5 მეტრზე გამაოგნებელი საოცრება ხდება: შუქი აჩირაღდნებს ღამეულ გზას, ჩნდებიან ავტომობილები, ისმის თვითმფრინავების ხმაური, ელექტრონული გამომთვლელი მანქანების ტაბლოები წამების მეასედებს ითვლიან. ფინიშზე მორბენლებს აბრმავებს ფოტო და ტელეკორესპოდენტების პროჟექტორების შუქი.

ჩვენო ძვირფასო მკითხველებო, გუსტავ აიხელბერგის მიერ დახატული სიუჟეტის გაგრძელებას თქვენს გონებას მივანდობთ, ჩვენ ჩვენებურად გავაგრძელებთ და ამ წიგნში იმ ერთ-ერთ საინტერესო ტექნოლოგიაზე ვისაუბრებთ, რომელიც აიხელბერგისეულ მორბენლებს ჩვენსკენ მომავალ გზაზე და მომავალშიც აუცილებლად შეხვდებათ, ეს საგნების ინტერნეტია (IoT, Internet of Things).

საგნების ინტერნეტის საშუალებით ადამიანებს შეუძლიათ ინფორმაციის მიმოცვლა საგნებთან, თავის მხრივ საგნებსაც შეუძლიათ ერთმანეთთან კომუნიკაცია, „საუბარი“. ადრე ითვლებოდა, რომ ინტერნეტში შეიძლება ჩავრთოთ მხოლოდ კომპიუტერები და სმარტფონები, დღეს უკვე თანამედროვე ჭკვიან მაცივარს შეუძლია მომხმარებელს შეტყობინება გაუგზავნოს ამა თუ იმ პროდუქტის შეძენის საჭიროებასთან დაკავშირებით ან სულაც თვითონ განახორციელოს პროდუქტების შეკვეთა; მცენარის მოვლისთვის აუცილებელი არ არის მასთან ახლოს

ვიმყოფებოდეთ, IoT ტექნოლოგია საშუალებას მოგვცემს მივიღოთ ინფორმაცია ნიადაგის ტენიანობის, გარემოს ტემპერატურისა და ტენიანობის, ზოგადად მცენარის მდგომარეობის შესახებ. ხომ ძალიან უცნაურია ეს ყველაფერი, მაგრამ მსგავსი რამ თითოეული ჩვენგანის გონებაში ჯერ კიდევ სკოლის წლებიდან აღბეჭდილა: შვლის ნუკრი, ხმელი წიფელი, ია, მთის წყარო, მთები... ისინი საუბრობენ, ფიქრობენ, რაღაცას დარდობენ, რაღაც უხარიათ. ვაჟა-ფშაველას ყველასათვის ცნობილი ეს გენიალური ნაწარმოებები მე-20 საუკუნის გარიჟრაჟზეა დაწერილი, 1915 წლამდე. 1926 წელს კი ასევე უდიდესი გენიოსი ნიკოლა ტესლა თავის ერთ-ერთ ინტერვიუში იტყვის, რომ მომავალში ყველა საგანი ერთიან დიდ სისტემაში გაერთიანდება, ხოლო ის მოწყობილობები, რომელთა საშუალებითაც ასეთი გაერთიანება გახდება შესაძლებელი, ასანთის კოლოფის ზომისა იქნება და ადვილად მოთავსდება ჯიბეში. ნიკოლა ტესლამ ზუსტად აღწერა ის, რასაც დღეს მთელი მსოფლიო საგნების ინტერნეტს უწოდებს.

ვისთვის არის ეს წიგნი განკუთვნილი

ამ ნაშრომით ჩვენ ვაგრძელებთ წიგნების სერიის - მიკროკონტროლერები დამწყებთათვის - გამოცემას. იმ დადებითმა გამოხმაურებებმა, რაც წინა წიგნის (ზ. ტაბატაძე. თ. თოდუა. არდუინო. პრაქტიკული სახელმძღვანელო დამწყებთათვის) მკითხველებისგან მივიღეთ, სტიმული

მოგვცა გაგვეგრძელებინა ეს საქმე და კიდევ ერთი საინტერესო ნაშრომი წარგვედგინა მათთვის. ეს წიგნიც წარმოადგენს საჩუქარს ქართველი ახალგაზრდობისთვის და მიზნად ისახავს მათთვის იმ ცოდნისა და გამოცდილების გადაცემას, რომელიც წიგნის ავტორებს გაგვაჩნია. წიგნის ელექტრონული ვერსია ღიად არის ხელმისაწვდომი ინტერნეტ სივრცეში.

წიგნი განკუთვნილია მკითხველთა ფართო წრისთვის, ყველა იმ ადამიანისთვის, ვისაც სურს გაეცნოს საგნების ინტერნეტის საინტერესო სამყაროს და განახორციელოს სახლის სრული ავტომატიზაცია შედარებით იაფად, NodeMCU დაფის გამოყენებით. წიგნში მოცემული ამოცანების განხორციელება მკითხველს საშუალებას მისცემს მათ საფუძველზე, კიდევ უფრო დიდი და მასშტაბური პროექტები განახორციელოს.

მადლიერება

მადლობა გვინდა გადავუხადოთ ყველა იმ ადამიანს, რომლებიც წიგნზე მუშაობის პროცესში დაგვეხმარნენ, მადლობა მათაც, ვინც უბრალოდ ხელი არ შეგვიშალა.

ჩვენი განსაკუთრებული მადლობა, მოწიწება და პატივისცემა მშობლებსა და მასწავლებლებს, რომ არა ისინი, არა თუ წიგნი, ერთი სიმბოლოც კი ვერ დაიწერებოდა.

მადლობა რეცენზენტებს: საქართველოს ტექნიკური უნივერსიტეტის კომპიუტერული ინჟინერიის დეპარტამენტის უფროსს, პროფესორ ლევან იმნაიშვილს, ამავე დეპარტამენტის ასოცირებულ პროფესორს, მაგული ბედინეიშვილს; შავი ზღვის საერთაშორისო უნივერსიტეტის პროფესორს ირაკლი როდონაიას; საქართველოს ტექნიკური უნივერსიტეტის არჩილ ელიაშვილის სახელობის მართვის სისტემების ინსტიტუტის უფროს მეცნიერ-მუშაკს ოთარ ქართველიშვილს, რომელთათვისაც საყვარელი საქმე და პროფესია ერთმანეთისგან განუყოფელი ცნებებია.

მადლობა ტექნიკური უნივერსიტეტის მეოთხე კურსის სტუდენტს, გიორგი ახობაძეს, უნიჭიერეს ადამიანს, რომელმაც წიგნის წერის პროცესში უამრავი საინტერესო მოსაზრება გაგვიზიარა.

მადლობა შავი ზღვის საერთაშორისო უნივერსიტეტს მხარდაჭერისა და გვერდში დგომისთვის, იმ უდიდესი წვლილისთვის, რომელიც წიგნის გამოცემაში შეიტანა.

ზაზა ტაბატაძე
თეა თოდუა

შესავალი

თანამედროვე ინფორმაციული საზოგადოება აქტიურად იყენებს ინფორმაციულ ტექნოლოგიებს მისი საქმიანობის თითქმის ყველა სფეროში. დღეისათვის მონაცემების გადაცემის ყველაზე მასშტაბურ ქსელს ინტერნეტი წარმოადგენს. ინტერნეტი გლობალური ინფორმაციული გარემოა, უზარმაზარი საკომუნიკაციო სივრცეა, რომლის საშუალებითაც მილიონობით ადამიანს ინფორმაციის შექმნის, დამუშავებისა და მიმოცვლის საშუალება აქვს. ადრე თუ ინტერნეტი ადამიანზე იყო ორიენტირებული და ის იყო ინფორმაციის წყაროც, მიმღებიც და დამმუშავებელიც, დღეს ადამიანთან ერთად, სხვადასხვა სახის საგანიც (იქნება ეს ჩვეულებრივი საყოფაცხოვრებო ნივთი თუ სხვა რამ) შეიძლება ასრულებდეს მსგავს ფუნქციებს; ხშირ შემთხვევაში, მათ შეიძლება დაეკისროთ მთავარი როლიც და ისინი შეიძლება იყვნენ ფიზიკური და ვირტუალური სამყაროს დამაკავშირებელი რგოლები. სიტუაციის მკვეთრი ცვლილება გასული საუკუნიდან დაიწყო. ტექნოლოგიების განვითარებამ სენსორების გამოჩენა გამოიწვია, სწორედ მათ შეძლეს ფიზიკური სამყაროს აღქმა ადამიანის უშუალო ჩარევის გარეშე. მონაცემთა გადაცემის ტექნოლოგიის განვითარებამ სხვადასხვა სახის სენსორისა და ქსელის გაერთიანების შესაძლებლობა წარმოშვა.

საგნების ინტერნეტი (Internet of Things, IoT) ერთმანეთთან და გარე სამყაროსთან ურთიერთქმედებისთვის ჩაშენებული ტექნოლოგიებით აღჭურვილი ფიზიკური ობიექტების გამოთვლითი ქსელია. ასეთ ქსელს გააჩნია სენსორები გარემომცველ სამყაროზე მონაცემების მისაღებად, იდენტიფიკაციისა და მონაცემთა გადაცემის, მიღებული ინფორმაციის შენახვისა და დამუშავების სხვადასხვა საშუალება.

თუ საკითხის რაოდენობრივ მხარეს განვიხილავთ, „ადამიანების ინტერნეტი“ უკვე საგნების ინტერნეტად გადაიქცა. დღეისათვის ინტერნეტით სარგებლობს მსოფლიოს 2 მილიარდზე მეტი ადამიანი, მაშინ როდესაც ქსელში ჩართული მოწყობილობების რაოდენობა დაახლოებით 40 მილიარდს აღწევს. ცხადია, რაოდენობრივი განსხვავება არ შეიძლება იყოს განმსაზღვრელი. საგნების ინტერნეტის უმნიშვნელოვანეს მახასიათებელს წარმოადგენს არა ჩართული მოწყობილობების რაოდენობა, არამედ მათი უნარი დამოუკიდებლად ან ადამიანის მინიმალური ჩარევით მოახდინონ რეალური სამყაროს იდენტიფიცირება და მის შესახებ ინფორმაციის მიღება.

საგნების ინტერნეტი. ძირითადი ცნებები და განმარტებები

ამა თუ იმ საკითხზე საუბრისას, ყოველთვის მნიშვნელოვანია, რამდენად კარგად აქვთ ადამიანებს წარმოდგენილი განსახილველი საკითხისა თუ ობიექტის არსი და რაობა. როდესაც საუბარია, ვთქვათ, უპილოტო საფრენ აპარატებზე, როგორც მინიმუმი, მოსაუბრეებმა უნდა იცოდნენ, რა იგულისხმება ამ სახელწოდების ქვეშ, მხოლოდ ამის შემდეგ შეძლებენ ისინი გაუგონ ერთმანეთს და გამოთქვან მოცემულ თემაზე თავიანთი მოსაზრებები. კონკრეტული საკითხისა თუ ცნების მაქსიმალურად ზუსტად განმარტება და შესაბამისი ტერმინოლოგიის სწორად გამოყენება ძალიან მნიშვნელოვანია.

საგნების ინტერნეტი ტექნოლოგიაა, რომელიც მოწყობილობებს აერთიანებს გამოთვლით ქსელში და საშუალებას აძლევს მათ შეაგროვონ, გაანალიზონ, დაამუშაონ და გადასცენ მონაცემები სხვა ობიექტებს პროგრამული უზრუნველყოფის და ტექნიკური მოწყობილობების საშუალებით. უმეტესწილად, მოწყობილობები ადამიანის ჩარევის გარეშე მუშაობენ, თუმცა ადამიანებს მათთან ურთიერთქმედება შეუძლიათ.

IoT სისტემები, როგორც წესი, შედგება ჭკვიანი მოწყობილობების ქსელისა და ღრუბლოვანი პლატ-ფორმისგან, რომელთანაც ეს მოწყობილობებია დაკავ-შირებული. თავიდან მოწყობილობები აგროვებენ მონაცემებს

(მაგ. ოთახის ტემპერატურის ან მომხმარებლის გულისცემის შესახებ), შემდეგ ეს მონაცემები გაიგზავნება ღრუბლებში. მოწყობილობები ღრუბლებს სხვადასხვა გზით (ფიჭური ან თანამგზავრული კავშირი, WiFi, Bluetooth ან კავშირის რომელიმე სხვა საშუალება) უკავშირდება. როგორც კი მონაცემები ღრუბელში მოხვდება, შესაბამისი პროგრამული უზრუნველყოფა ამუშავებს მათ. ეს შეიძლება იყოს ძალიან მარტივი პროცესი, მაგალითად, სახლის ტემპერატურის შედარება მომხმარებლის მიერ ტემპერატურის წინასწარ მოცემულ დასაშვებ დიაპაზონთან. შეიძლება გაცილებით რთული ამოცანის წინაშე ვიდგეთ, როგორიცაა, მაგალითად, ვიდეოზე ობიექტების იდენტიფიკაციისთვის კომპიუ-ტერული ხედვის გამოყენება; ეს ობიექტები შეიძლება იყვნენ კრიმინალები, რომლებმაც სახლში შეაღწიეს.

რა ხდება იმ შემთხვევაში, როდესაც ტემპერატურა დასაშვებზე მაღალი აღმოჩნდება ან სახლში მძარცველის არსებობა იქნება გამოვლენილი? სისტემას შეუძლია აცნობოს ამის თაობაზე მომხმარებელს, ეს შეიძლება იყოს ტექსტური შეტყობინება ან სიგნალი; სისტემას ასევე შეუძლია თვითონ განახორციელოს შემდგომი მოქმედებები, თუ მომხმარებელს სისტემისთვის წინასწარ აქვს მიცემული გარკვეული ინსტრუქციები. მაგალითად, ნაცვლად იმისა, რომ დაურეკოს სახლის პატრონს, სისტემას შეუძლია დაუყოვნებლივ გაუგზავნოს პოლიციას შეტყობინება.

საგნების ინტერნეტი ბევრისთვის ჭკვიანი სახლის სისტემასთან ასოცირდება, თუმცა ეს ცნება მხოლოდ ჭკვიან სახლს არ გულისხმობს. საგნების ინტერნეტს მნიშვნელოვანი ადგილი უჭირავს ბიზნესის სფეროშიც. ის კომპანიებს პროცესების ავტომატიზების შესაძლებლობას აძლევს, რითაც დანახარჯები მცირდება.

საგნების ინტერნეტი ეხება ყველა დარგს, ჯანდაცვის, ფინანსების, საცალო ვაჭრობისა და წარმოების ჩათვლით. საგნების ინტერნეტის დანერგვა, დისტანციური მონიტორინგის სახით, ელექტროენერგეტიკაში ქვესადგურებისა და ელექტროგადამცემი ხაზების კონტროლის გაუმჯობესებას იწვევს.

ჯანდაცვაში საგნების ინტერნეტის საშუალებით შესაძლებელია დიაგნოსტიკის გაცილებით მაღალი დონის მიღწევა. „ჭკვიან“ მოწყობილობებს შეუძლიათ პაციენტის ჯანმრთელობის მაჩვენებლები აკონტროლონ. ეს ამცირებს არაგეგმიური ჰოსპიტალიზაციისა და სტაციონარების ზედმეტად დატვირთვის რისკებს.

სოფლის მეურნეობაში ჭკვიანი ფერმები და სათბურები თვითონ ახდენენ სასუქებისა და წყლის დოზირებას. ეს იწვევს მოსავლიანობის გაზრდას, აუმჯობესებს პროდუქციის ხარისხს, ამცირებს სასოფლო-სამეურნეო ტექნიკის საწვავის ხარჯს.

საგნების ინტერნეტი სატრანსპორტო სფეროში უმეტეს შემთხვევაში გულისხმობს ელექტრონულ ტაბლოს, ნავი-გატორებს, უსაფრთხოების სისტემებს, დაკვირვების კამერებს, რომლებიც ერთმანეთთან ურთიერთქმედებენ. მიღებული მონაცემების მონიტორინგი შესაძლებელია მობილური აპლიკაციების საშუალებით. დიდ ქალაქებში არსებული ჭკვიანი პარკირების სისტემები აღჭურვილია სენსორებით, რომლებიც თავისუფალი ადგილების შესახებ ინფორმაციას აწვდიან სერვერს.

საგნების ინტერნეტი და მეოთხე ინდუსტრიული რევოლუცია

სიტყვა „რევოლუცია“ მკვეთრ და კარდინალურ ცვლილებას ნიშნავს. რევოლუციები ხდებოდა კაცობრიობის ისტორიული განვითარების სხვადასხვა საფეხურზე, როდესაც ახალი ტექნოლოგიები და სამყაროს აღქმის ახალი საშუალებები ფუნდამენტურ ცვლილებებს იწვევდა ეკონომიკურ სისტემებსა და სოციალურ სტრუქტურებში.

პირველი კარდინალური ძვრა ადამიანის ცხოვრებაში შემგროვებლობიდან მიწათმოქმედებაზე გადასვლა იყო. ეს დაახლოებით 10000 წლის წინ მოხდა, ადამიანის მიერ ცხოველების მოშინაურების შემდეგ. აგრარული რევოლუცია მოხდა ადამიანებისა და ცხოველების ძალთა გაერთიანების შედეგად, წარმოების, ტრანსპორტირებისა და კომუნიკაციის

გაზრდის მიზნით. თანდათანობით გაიზარდა კვების პროდუქტების წარმოების ეფექტურობა, ამან ხელი შეუწყო მოსახლეობის ზრდასა და სიცოცხლისუნარიანობას. ამ ფაქტორებმა შემდგომში ხელი შეუწყვეს ურბანიზაციის პროცესებს.

აგრარული რევოლუციის შემდეგ, XVIII საუკუნის მეორე ნახევრიდან, მსოფლიოში ინდუსტრიული რევოლუციები იწყება.

პირველი ინდუსტრიული რევოლუცია (Industry 1.0) 1760-იანი წლებიდან 1840-იან წლებამდე პერიოდს მოიცავს. ამ რევოლუციას უკავშირდება რკინიგზების მშენებლობა და ორთქლის ძრავას გამოგონება, რამაც მექანიკური წარმოების განვითარებას შეუწყო ხელი.

მეორე ინდუსტრიული რევოლუცია (Industry 2.0) XIX საუკუნის ბოლოს იწყება და XX საუკუნის დასაწყისამდე გრძელდება. ამ რევოლუციამ განაპირობა მასობრივი წარმოების წარმოშობა ელექტროფიკაციისა და კონვეიერის დანერგვის გამო.

მესამე ინდუსტრიული რევოლუცია (Industry 3.0) XX საუკუნის 60-იან წლებში დაიწყო. ჩვეულებრივ, მას კომპიუტერულ ან ციფრულ რევოლუციას უწოდებენ, რადგანაც ის ნახევარგამტარების, დიდი გამომთვლელი მანქანების (გასული საუკუნის 60-იანი წლები), პერსონალური კომპიუტერების (გასული საუკუნის 70-იანი და 80-იანი

წლები), და ინტერნეტის განვითარებამ (გასული საუკუნის 90-იანი წლები) განაპირობა.

მეოთხე ინდუსტრიული რევოლუცია (Industry 4.0) ახალ ათასწლეულთან ერთად დაიწყო და ციფრული ტექნიკის განვითარებას ეფუძნება. მისთვის დამახასიათებელია ყველგან არსებული და მობილური ინტერნეტი, მინიატურული საწარმოო მოწყობილობები, ხელოვნური ინტელექტი და თვითსწავლადი მანქანები. მეოთხე ინდუსტრიული რევოლუციის ძირითადი მიზანია წარმოების მეთოდების შეცვლა: წარმოების კონტროლი რეალური დროის რეჟიმში, ასევე მაქსიმალურად მჭიდრო ურთიერთკავშირი ტექნოლოგიურ და ბიზნეს პროცესებს შორის.

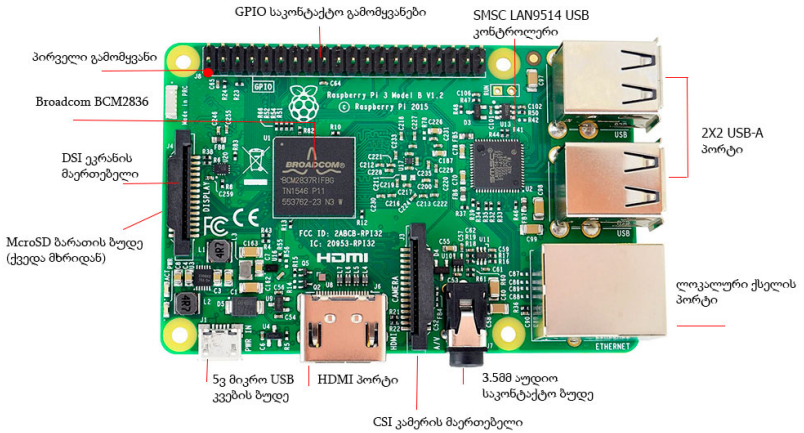
მესამე ინდუსტრიული რევოლუცია გულისხმობს, რომ ვირტუალურ და ფიზიკურ სამყაროს შორის გარკვეული საზღვარი არსებობს. ფიზიკური სენსორები ვირტუალურ მონაცემებს წარმოქმნიან, რომლებიც გადაწყვეტილების ავტომატურად მისაღებად არ გამოიყენება. Industry 4.0 ტექნოლოგია საშუალებას იძლევა შეიქმნას კიბერფიზიკური სისტემები, რომლებიც შლიან ამ საკმაოდ პირობით საზღვარს რეალურ და ვირტუალურ სამყაროებს შორის.

რა თქმა უნდა, Industry 3.0-ის თაობის სისტემები ასევე აგროვებენ მონაცემებს და ახდენენ მათ მიმოცვლას, მაგრამ მხოლოდ Industry 4.0 ხდის შესაძლებელს, რომ ეს მონაცემები რეალურ დროში იქნეს გამოყენებული, გარკვეული საწარმოო

პროცესების დამოუკიდებლად შესასრულებლად და გარე-მოსთან ურთიერთქმედებისთვის. Industry 4.0 ტექნოლოგია გულისხმობს, რომ ტექნიკას თვითონ შეუძლია შეასრულოს საწარმოო ამოცანები ადამიანის ჩარევის გარეშე.

საჭირო ტექნიკური და პროგრამული უზრუნველყოფა

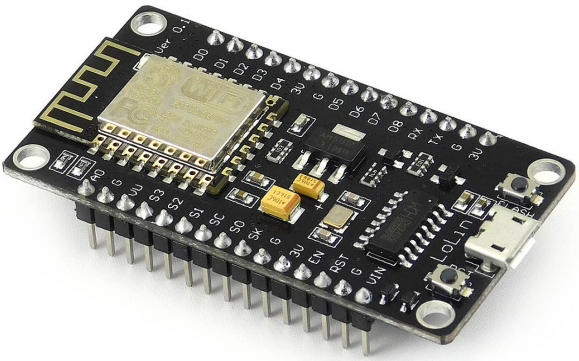
წიგნში მოცემული პროექტების განსახორციელებლად გამოვიყენებთ Raspberry Pi 3 ვერსიას. მას გააჩნია ჩაშენებული Wi-Fi ინტერფეისი და დამატებითი USB პორტები. ერთდაფიანი კომპიუტერების ოჯახის წარმომადგენელი Raspberry Pi წარმოადგენს Raspberry Pi Foundation-ის პროდუქტს (RaspberryPi.org). Raspberry Pi ძირითადად განკუთვნილია Linux-ის ოპერაციული სისტემებისთვის, მათ შორის ყველაზე პოპულარულია Raspberry Pi OS.



სურ. 1. Raspberry Pi 3

Raspberry Pi-ს გააჩნია შესაბამისი დრაივერები უამრავი სხვადასხვა მოწყობილობისა და სენსორისთვის, რაც ძალიან ამარტივებს მისი IoT პროექტებში გამოყენების შესაძლებლობას, განსაკუთრებით საინტერესოა მისი, როგორც სერვერის გამოყენება. მიუხედავად იმისა, რომ Raspberry Pi არ წარმოადგენს დაბალი ენერგომოხმარების მოწყობილობას, რაც ზღუდავს მის გამოყენებას IoT მოწყობილობის სახით, ის მაინც რჩება საუკეთესო სერვერად და პროტოტიპირების საშუალებად.

IoT მოწყობილობის სახით გამოვიყენებთ NodeMCU ESP8266-ს. ის წარმოადგენს იაფ IoT პლატფორმას, ღია საწყისი კოდით. ESP8266-ს აქვს Wi-Fi-ს მიმღებ/გადამცემი, მოდულს შეუძლია გააგზავნოს და მიიღოს ინფორმაცია ლოკალურ ქსელში ან ინტერნეტში Wi-Fi-ს გამოყენებით.



სურ. 2. NodeMCU ESP8266

წიგნში უხვადაა პრაქტიკული მაგალითები, რომელთა განხორციელებისთვის საჭიროა სხვადასხვა სენსორი და შემსრულებელი მექანიზმი (სურ.3)

სურ. 3. სენსორები და შემსრულებელი მექანიზმები

Raspberry Pi-ს პლატფორმა

ყველა მაგალითი, რომელსაც შემდგომში განვიხილავთ, გულისხმობს რაიმე ტიპის ციფრული შესასვლელების და/ან გამოსასვლელების გამოყენებას სენსორებთან და შემსრულებელ მექანიზმებთან ურთიერთქმედებისთვის. მოწყობილობები შესასვლელ/გამოსასვლელებს მიმართავს, როგორც ზოგადი დანიშნულების შესასვლელ/გამოსასვლელებს (GPIO – General-purpose input output). თითოეული მოწყობილობის GPIO-ს აქვს განსხვავებული სპეციფიკაცია მაქსიმალურ ძაბვასა და დენთან მიმართებაში, რისი გათვალისწინებაც აუცილებელია.

სურ. 4-ზე მოცემულია GPIO გამომყვანების განლაგების სქემა Raspberry Pi 3-თვის.

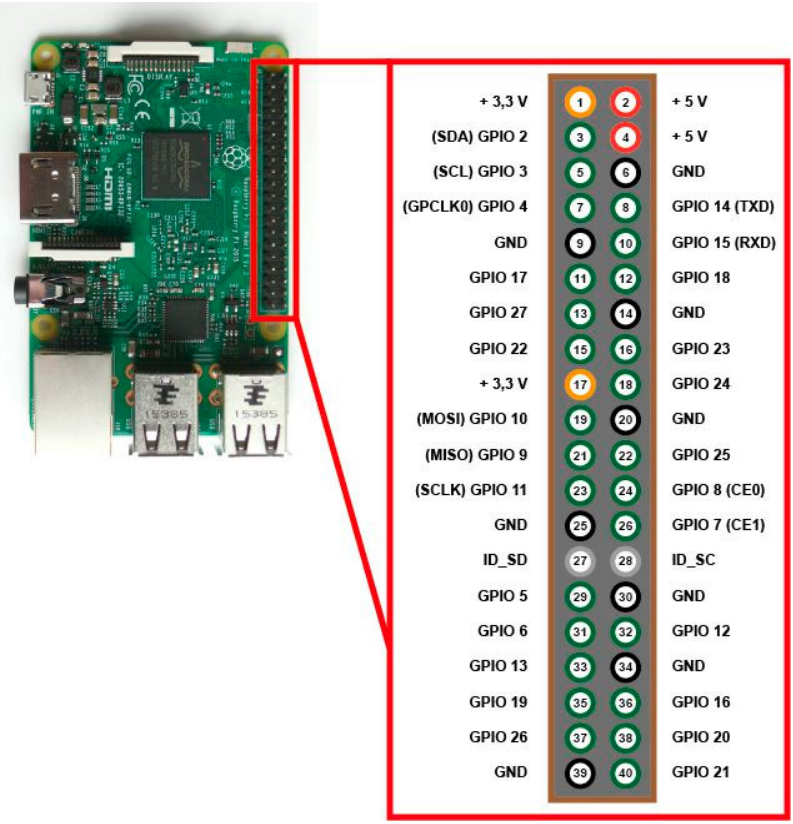
Raspberry Pi 3-ის კვების საკონტაქტო გამომყვანები:

Raspberry Pi 3-ის დაფაზე გვაქვს ორი 5V, ორი 3.3V და 8 Gnd („მიწა“) საკონტაქტო გამომყვანი. როგორც წესი, მათი კონფიგურირება შეუძლებელია.

5V საკონტაქტო გამომყვანი გამოიყენება ქსელის ადაპტერიდან უშუალოდ კვებისთვის. მისი გამოყენება შეიძლება როგორც Raspberry Pi-ს, ასევე სხვა 5V-ზე მომუშავე მოწყობილობების კვებისთვის.

3.3V საკონტაქტო გამომყვანი გამოიყენება 3.3V სტაბილური კვების მისაწოდებლად გარე მოწყობილობებისთვის.

GND საკონტაქტო გამომყვანი გამოიყენება დამიწებისთვის. ყველა ძაბვა იზომება მასთან მიმართებაში.



სურ.4. GPIO გამომყვანების განლაგების სქემა Raspberry Pi 3-თვის

შეტანა-გამოტანის საკონტაქტო გამომყვანები. GPIO გამომყვანი, რომელიც კონფიგურირებულია როგორც შესასვლელი, კითხულობს Raspberry Pi-ს მიერ მიღებულ სიგნალს,

გამოგზავნილს იმ მოწყობილობიდან, რომელიც ამ საკონტაქტო გამომყვანს უკავშირდება. Raspberry Pi-ს მიერ 1.8V-დან 3.3V-მდე ძაბვის მნიშვნელობა წაიკითხება როგორც ლოგიკური 1 (HIGH), 1.8V-მდე ძაბვა კი წაიკითხება, როგორც ლოგიკური 0 (LOW).

GPIO გამომყვანებისთვის არ შეიძლება მოწყობილობების მიერთება 3.3V-ზე მეტი ძაბვით, ეს გამოიწვევს Raspberry Pi-ს გადაწვას.

GPIO გამომყვანი, რომელიც კონფიგურირებულია როგორც გამოსასვლელი, აგზავნის მაღალი (3.3V) ან დაბალი (0V) დონის სიგნალს. როდესაც ეს გამომყვანი არის HIGH მდგომარეობაში, ეს ნიშნავს, რომ მასზე არის 3.3V ძაბვა, როდესაც გამომყვანი არის LOW მდგომარეობაში, ეს ნიშნავს, რომ მასზე არის 0V ძაბვა.

GPIO გამომყვანების უმეტესობას, გარდა შესასვლელ/გამოსასვლელის ფუნქციისა, სხვა დანიშნულებაც აქვს (სურ.4). ქვემოთ მოცემულია მათი აღწერა:

PWM (განივ-იმპულსური მოდულაცია) საკონტაქტო გამომყვანები

- პროგრამული PWM-ის მიღწევა შესაძლებელია ყველა საკონტაქტო გამომყვანზე;
- აპარატურული PWM-ის მიღწევა შესაძლებელია მხოლოდ GPIO 12, GPIO 13, GPIO 18, GPIO 19 გამომყვანებზე.

SPI (Serial Peripheral Interface) საკონტაქტო გამომყვანები. Raspberry Pi SPI პროტოკოლს იყენებს ერთ ან რამდენიმე პერიფერიულ მოწყობილობასთან სწრაფი კომუნიკაციისთვის.

SPI საკონტაქტო გამომყვანები Raspberry Pi 3-ზე:
GPIO 8 (CE0), GPIO 7 (CE1), GPIO 10 (MOSI), GPIO 9 (MISO), GPIO 11 (SCLK).

I2C (Inter-Integrated Circuit) საკონტაქტო გამომყვანები Raspberry Pi 3-ზე. I2C გამომყვანები გამოიყენება იმ მოწყობილობებთან კავშირისთვის, რომლებიც თავსებადია I2C პროტოკოლთან.

12C საკონტაქტო გამომყვანები Raspberry Pi 3-ზე:

- SDA: (GPIO2), SCL (GPIO3)
- EEPROM Data: (საკონტაქტო გამომყვანი 27), EEPROM Clock (საკონტაქტო გამომყვანი 28)

UART (Universal Asynchronous Receiver/Transmitter) გამომყვანები Raspberry Pi 3-ზე. UART გამომყვანები უზრუნველყოფს კავშირს ორ მიკროკონტროლერს ან კომპიუტერს შორის.

- TX (GPIO14)
- RX (GPIO15)

სქემის აგებისას შესაძლებელია Raspberry Pi-ს ნებისმიერი GPIO გამომყვანის გამოყენება, მხოლოდ არ უნდა დაგვავიწყდეს, რომ GPIO გამომყვანზე დატვირთვის დენი არ უნდა აღემატებოდეს 16 მილიამპერს.

Raspberry Pi OS ოპერაციული სისტემის ინსტალაცია

იმისათვის, რომ ვიმუშაოთ Raspberry Pi-თან, საკმარისი არ არის მხოლოდ ამ მოწყობილობის შეძენა. ქვემოთ ჩამოთვლილია საჭირო კომპონენტები, რომლებიც დაგვჭირდება Raspberry Pi-თან მუშაობის დასაწყებად:

- ***კვების წყარო.*** დაგვჭირდება micro USB კაბელი, რომელსაც მივაერთებთ კვების წყაროსთან. Raspberry Pi 3 ვერსიისთვის ოპტიმალური კვების წყაროს მონაცემები უნდა იყოს 5 ვოლტი და 2.5 ამპერი, რაც უზრუნველყოფს საკმარის სიმძლავრეს ყველა იმ პერიფერიული მოწყობილობისთვის, რომლებიც შესაძლებელია მის USB პორტებთან იყვნენ მიერთებული.
- ***HDMI კაბელი.*** Raspberry Pi-სთან სამუშაოდ შეგვიძლია კომპიუტერის მონიტორის გამოყენება. ამისათვის HDMI კაბელი უნდა შევუერთოთ მონიტორის HDMI შესასვლელს.
- ***USB მაუსი და კლავიატურა*** ძალიან მოსახერხებელია Raspberry Pi-თან სამუშაოდ, თუმცა, რა თქმა უნდა, შესაძლებელია უსადენო მაუსის და კლავიატურის გამოყენებაც.
- ***მეხსიერების SD ბარათი.*** ოპერაციული სისტემის ჩასაწერად საჭიროა მინიმუმ 8 გიგაბაიტი მოცულობის ბარათი, თუმცა უკეთესი იქნება 32 გიგაბაიტი მოცულობის ბარათის შეძენა.

- *კომპიუტერი/ნოუთბუქი.* იმისათვის, რომ RaspberryPi-ში ჩავწეროთ ოპერაციული სისტემა, აუცილებლად დაგვჭირდება კომპიუტერი.
- *მეხსიერების Micro SD ბარათის წამკითხველი.* თვითონ Raspberry Pi-ს SD ბარათის წამკითხველი არ სჭირდება, ის დაგვჭირდება იმ შემთხვევაში, თუ ჩვენს კომპიუტერს SD ბარათის წამკითხავი არ აქვს.

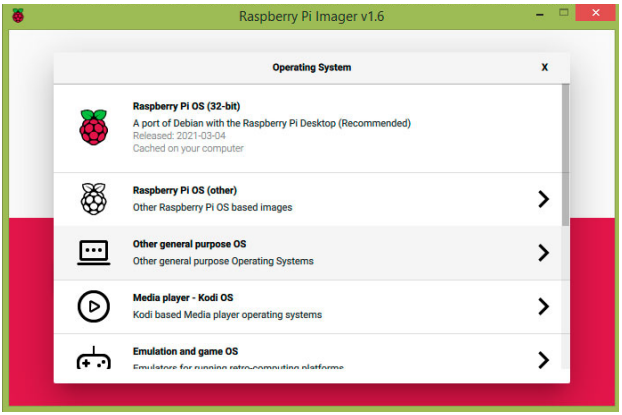
მას შემდეგ, რაც ხელთ გვექნება ყველა საჭირო კომპონენტი, შეგვიძლია შევუდგეთ ოპერაციული სისტემის Raspberry Pi-ში ინსტალაციას. ამისათვის შეგვიძლია გამოვიყენოთ Windows, Mac ან Linux ოპერაციულ სისტემებზე მომუშავე კომპიუტერი. ჩვენ გამოვიყენებთ პერსონალურ კომპიუტერს Windows ოპერაციული სისტემით, თუმცა Raspberry Pi-ზე ოპერაციული სისტემის ინსტალაციის პროცესი მსგავსია Mac ან Linux სისტემებზე მომუშავე კომპიუტერების შემთხვევაში. მივყვეთ ქვემოთ მოცემულ ინსტრუქციას:

1. მოვათავსოთ ბარათი/ბარათის წამკითხველი კომპიუტერში.
2. ჩამოვტვირთოთ და დავაინსტალიროთ Raspberry Pi Imager ფაილი (https://www.raspberrypi.org/software/) კომპიუტერში. ფაილი მისაწვდომია როგორც Windows, ასევე macOS და Linux სისტემებისთვის. ჩვენს შემთხვევაში, ვაჭერთ Download for Windows ღილაკს.

ზემოთ აღნიშნული პროცედურების შესრულების შემდეგ ჩვენი კომპიუტერის ეკრანზე გამოვა Raspberry Pi Imager-ის ფანჯარა.

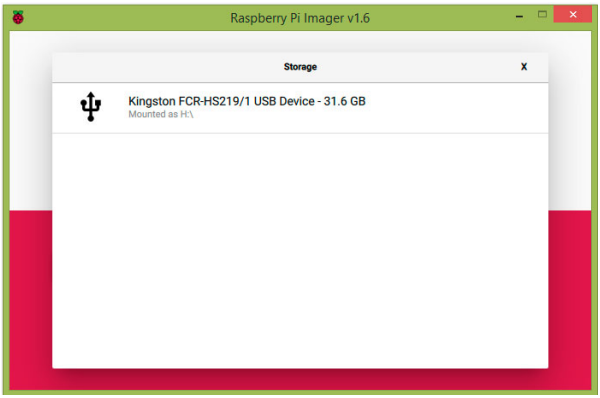
სურ. 5. Raspberry Pi Imager-ის ფანჯარა

Click Choose OS ღილაკზე დაჭერით ვირჩევთ Raspberry Pi OS (32-bit) ოპერაციულ სისტემას.



სურ. 6. Raspberry Pi OS სისტემის არჩევა

Choose SD Card ღილაკზე დაჭერით ვირჩევთ ჩვენს SD ბარათს.

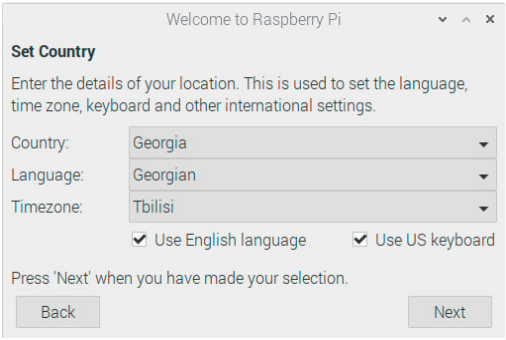


სურ. 7. SD ბარათის არჩევა

SD ბარათის არჩევის შემდეგ დავაჭიროთ Write ღილაკს. Raspberry Pi OS ოპერაციული სისტემის ინსტალაციის დასრულების შემდეგ SD ბარათი ჩავდოთ Raspberry Pi-ში. HDMI კაბელის მეშვეობით Raspberry Pi შევუერთოთ მონიტორს, Micro USB კაბელი შევაერთოთ კვების წყაროსთან. მონიტორზე გამოჩნდება მეხსიერების ბარათზე დაინსტალირებული ოპერაციული სისტემა.

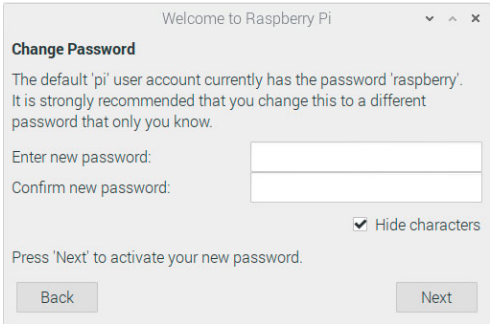
სისტემის პირველი ჩატვირთვისას, ეკრანზე გამოჩნდება „Welcome to the Raspberry Pi Desktop!" დიალოგური ფანჯარა, რომელსაც მივყავართ სისტემის მნიშვნელოვანი პარამეტრების არჩევამდე.

„Welcome to the Raspberry Pi Desktop!" დიალოგურ ფანჯარაში დავაჭიროთ Next ღილაკს, შევარჩიოთ ქვეყანა, ენა, საათობრივი სარტყელი და კლავიატურის ტიპი.



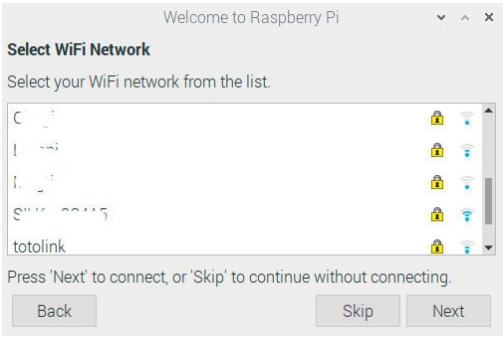
სურ.8. რეგიონული პარამეტრების არჩევა

Next ღილაკზე დაჭერის შედეგად გამოჩნდება მომხმარებლის პაროლის შექმნის დიალოგური ფანჯარა.



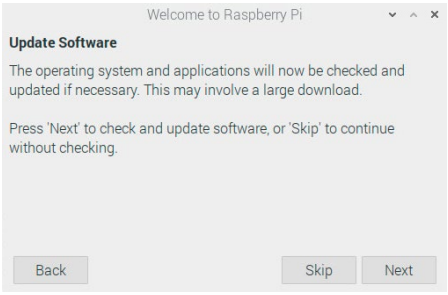
სურ. 9. მომხმარებლის პაროლის შექმნა

ზემოთ აღწერილი მოქმედების შესრულების შემდეგ გამოჩნდება ფანჯარა, რომელიც ჩვენი Wi-Fi ქსელის ამორჩევის საშუალებას მოგვცემს. Wi-Fi ქსელის არქონის ან Ethernet-ის გამოყენების შემთხვევაში, შეგვიძლია გამოვტოვოთ ეს პროცედურა. Next ღილაკზე დაჭერის შემდეგ გამოჩნდება Enter WiFi Password ფანჯარა. პაროლის შეტანის შემდეგ ვაჭერთ Next ღილაკს.



სურ. 10. Wi-Fi ქსელის არჩევა

შემდეგ ეტაპზე გამოჩნდება პროგრამული უზრუნველყოფის განახლების ფანჯარა. ინტერნეტთან კავშირის გარეშე განახლება შეუძლებელია, ასეთ შემთხვევაში Skip ღილაკს უნდა დავაჭიროთ, წინააღმდეგ შემთხვევაში, ვაჭერთ Next ღილაკს.

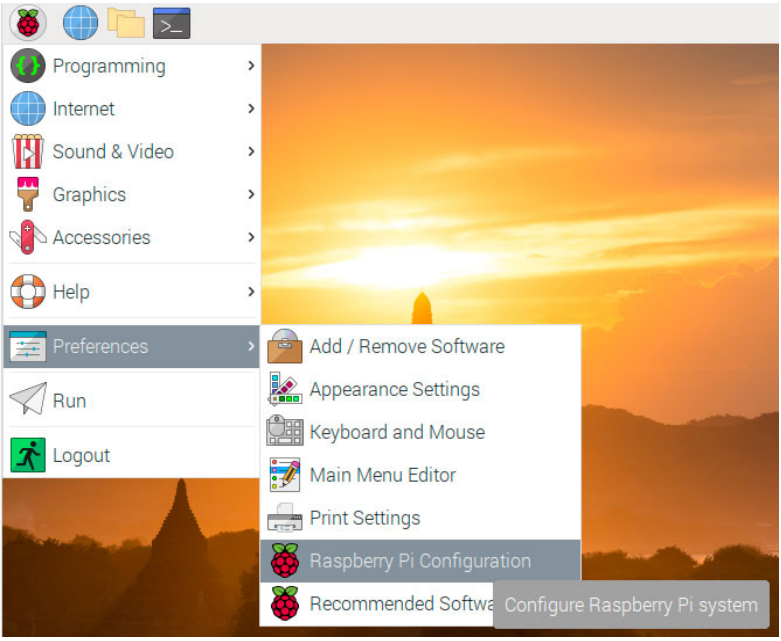


სურ. 11. პროგრამული უზრუნველყოფის განახლება

დავაჭიროთ Done ან Restart-ს (თუ იქნება მოთხოვნილი Restart)

Raspberry Pi OS-ის კონფიგურირება

ოპერაციული სისტემის ინსტალაციის შემდეგ, სისტემის პარამეტრების შეცვლა შესაძლებელია ეკრანის ზედა მარცხენა კუთხიდან Preferences ->Raspberry Pi Configuration მენიუს გამოძახების გზით.

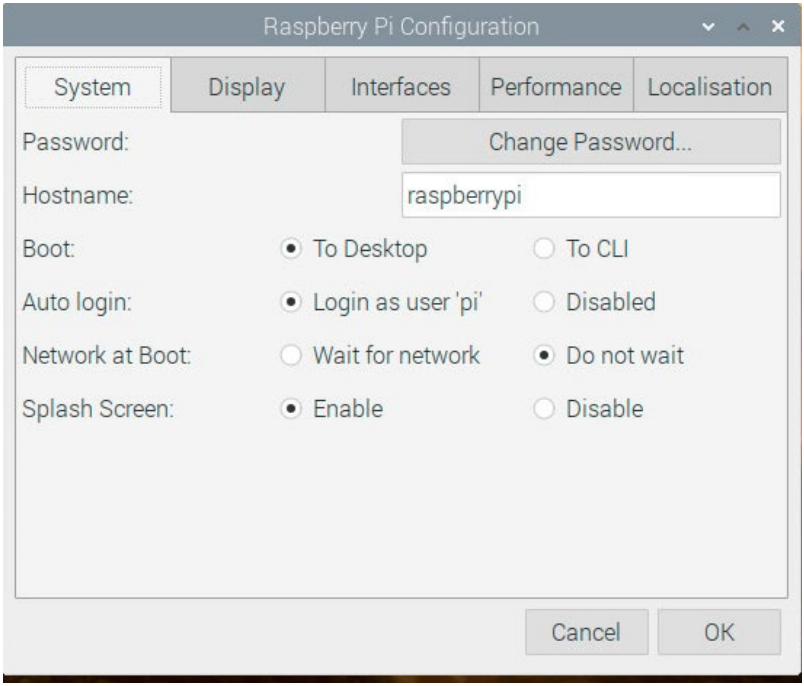


სურ. 12. Raspberry Pi-ს კონფიგურირების მენიუ

Raspberry Pi-ს კონფიგურირების ფანჯრის System ჩანართის (სურ.13) საშუალებით შესაძლებელია შემდეგი პარამეტრების დაყენება:

- მომხმარებლის პაროლის შეცვლა;

- მოწყობილობის სახელის შეცვლა;
- სისტემის ჩატვირთვა საბრძანებო სტრიქონის ან გრაფიკული ინტერფეისის სახით;
- სისტემაში ავტომატურად შესვლა;
- ჩატვირთვა ქსელის ლოდინით და ქსელის გარეშე;
- ჩატვირთვისას ტექსტური ან გრაფიკული ინტერფეისის გამოჩენა



სურ. 13. Raspberry Pi-ს კონფიგურირების ფანჯარა

მოწყობილობის სახელის შეცვლა განსაკუთრებით აქტუალური ხდება იმ შემთხვევაში, თუ ერთზე მეტი Raspberry Pi გვაქვს. მოწყობილობის სახელის შესაცვლელად შესაბამის ველში (hostname) საკმარისია სასურველი სახელის შეტანა.

Boot to desktop ან Boot to CLI პარამეტრები განსაზღვრავს, Raspberry Pi-ს ჩატვირთვის შემდეგ გრაფიკულ ინტერფეისთან ვიმუშავებთ თუ ტერმინალთან. ჩვენს შემთხვევაში, დავაყენეთ desktop, რადგანაც უმეტესწილად გრაფიკულ გარემოში მოგვიწევს მუშაობა.

როგორც წესი, Raspberry Pi ავტომატურად შედის ოპერაციულ სისტემაში მისი ჩატვირთვისას. იმ შემთხვევაში, თუ ჩვენი Raspberry Pi შეიცავს რაიმე სახის კონფიდენციალურ ინფორმაციას ან პირდაპირ უკავშირდება Wi-Fi ქსელს, რომელსაც სხვა მოწყობილობებთან აქვს წვდომა, უმჯობესი იქნება სისტემაში ავტომატურად შესვლის ფუნქცია გამოვრთოთ. ჩვენს შემთხვევაში, დაყენებული გვაქვს ავტომატური შესვლა (სურ. 13).

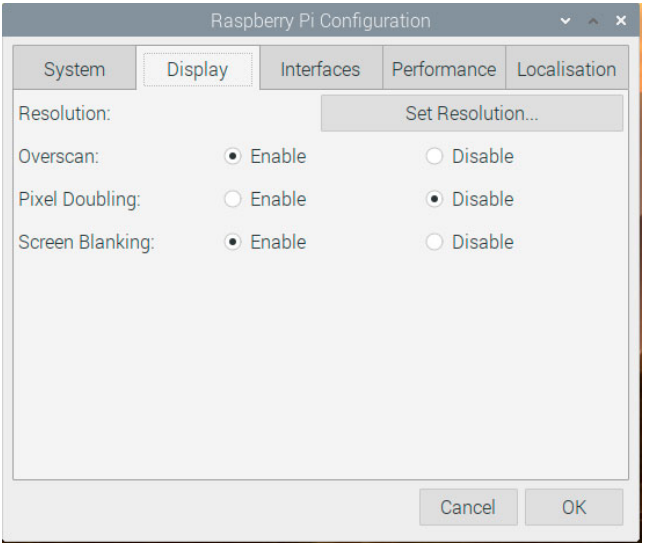
Network at boot პარამეტრის ჩართვა სასარგებლოა მაშინ, როდესაც სხვა მოწყობილობებთან ან საქაღალდეებთან გვაქვს ქსელური რესურსები გასაზიარებელი. ასეთ შემთხვევაში, Raspberry Pi დაელოდება, სანამ ქსელი გახდება მისაწვდომი, შემდეგ ჩაიტვირთება ოპერაციული სისტემა და შესაძლებელი გახდება ქსელური რესურსების გაზიარება. ქსელთან დაკავშირების პრობლემის შემთხვევაში, Raspberry

Pi მაინც ჩაიტვირთება, თუმცა ამას შედარებით დიდი დრო დასჭირდება. ჩვენს შემთხვევაში, ეს პარამეტრი დავტოვეთ გამორთულ მდგომარეობაში (სურ.13).

Raspberry Pi-ს კონფიგურირების ფანჯრის System ჩანართის ბოლო პარამეტრია Splash Screen. ამ პარამეტრის ჩართვით სისტემის ჩატვირთვისას გრაფიკული ინტერფეისი გამოჩნდება, წინააღმდეგ შემთხვევაში - ტექსტური.

Raspberry Pi-ს კონფიგურირების ფანჯრის Display ჩანართში შესაძლებელია სამი სხვადასხვა პარამეტრის კონფიგურირება:

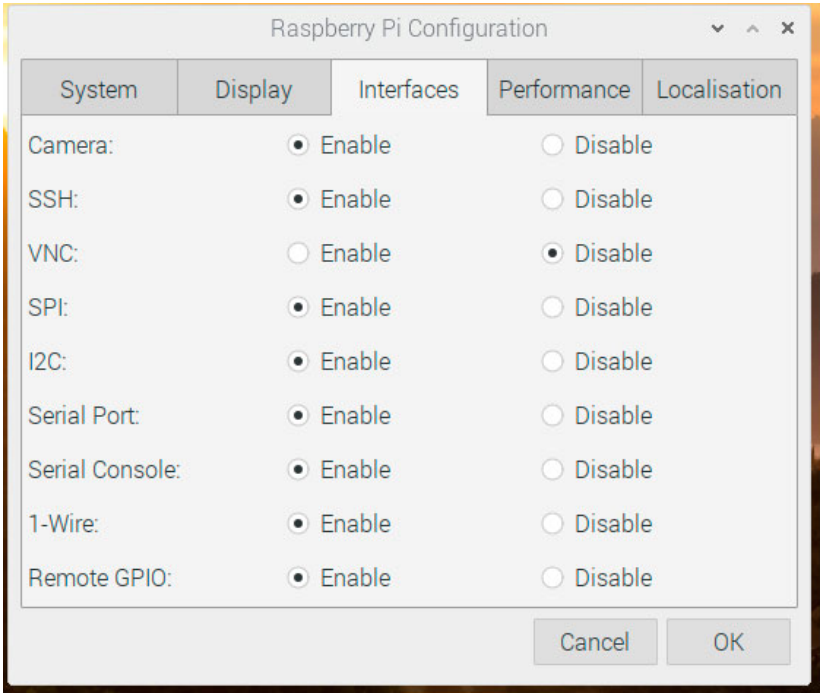
- Overscan
- Screen Blanking
- Headless Resolution



სურ. 14. Raspberry Pi-ს კონფიგურირების ფანჯარა

თუ Rasperry Pi-ს ჩატვირთვის შემდეგ ეკრანზე შავი ჩარჩო ჩანს, Overscan პარამეტრის გამორთვა ამ ჩარჩოს მოშორებას უზრუნველყოფს. Screen Blanking პარამეტრი განსაზღვრავს ეკრანი ჩაქრეს თუ არა, გარკვეული პერიოდით უმოქმედობის შემდეგ. Headless Resolution-ის საშუალებით ვირჩევთ ეკრანის გარჩევადობის პარამეტრებს.

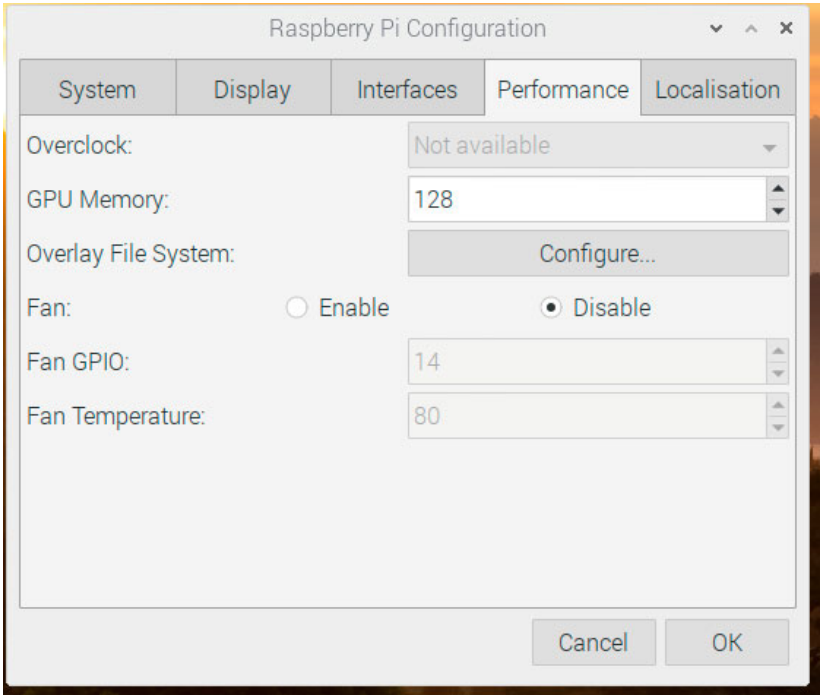
Raspberry Pi-ს კონფიგურირების ფანჯრის Interfaces ჩანართში, წინა ჩანართებისგან განსხვავებით, ბევრი პარამეტრის დაყენებაა შესაძლებელი. საჭიროების შემთხვევაში, შეგვიძლია ისინი ჩავრთოთ ან დავტოვოთ ისე, როგორც არის.



სურ. 15. Raspberry Pi-ს კონფიგურირების ფანჯარა

მაგალითად, თუ Raspberry Pi-თან დაკავშირება ხდება SSH პროტოკოლით, მაშინ ვრთავთ შესაბამის პარამეტრს. პარამეტრების უმეტესობა დაკავშირებულია გარე პერიფერიულ მოწყობილობებთან.

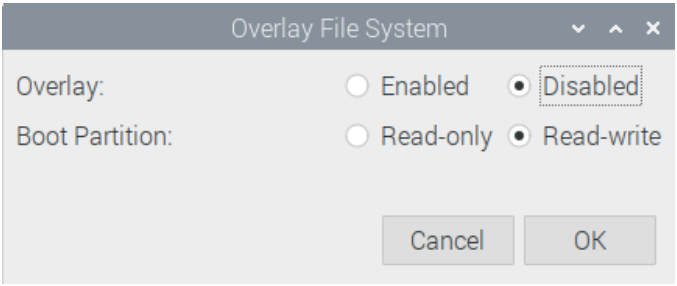
Raspberry Pi-ს კონფიგურირების ფანჯრის Performance ჩანართში მონაცემების შეცვლა განსაკუთრებულ სიფრთხილეს მოითხოვს. გამოუცდელი მომხმარებლის შემთხვევაში, უმჯობესია პარამეტრები უცვლელი დავტოვოთ.



სურ. 16. Raspberry Pi-ს კონფიგურირების ფანჯარა

როგორც სურ. 16-დან ჩანს, GPU Memory პარამეტრი საშუალებას იძლევა დავაყენოთ Raspberry Pi-ს ოპერატიული მეხსიერების ის მოცულობა, რომელიც გრაფიკული დამუშავებისათვისაა საჭირო. ჩვენ შეგვიძლია ამ მნიშვნელობის გაზრდა, თუ Raspberry Pi-ს გამოყენება მაღალი დონის გრაფიკას უკავშირდება, და პირიქით - შემცირება, თუ ძირითადად ტერმინალთან ვაპირებთ მუშაობას. ამ ეტაპზე ჩვენი რჩევაა ეს პარამეტრი დატოვოთ ისე, როგორც არის და არაფერი შეცვალოთ.

ძალიან საინტერესოა Overlay File System პარამეტრი. ამ პარამეტრის გასწვრივ, შესაბამის ველში „Configure”-ზე დაჭერის შედეგად გამოჩნდება ქვემოთ მოცემული ფანჯარა.

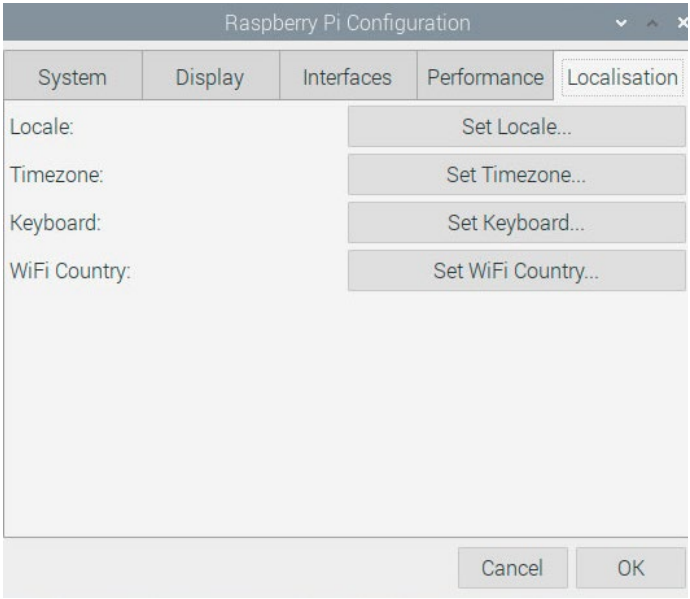


სურ. 17. Raspberry Pi-ს კონფიგურირების ფანჯარა

თუ Overlay პარამეტრი ჩართულია, ფაილური სისტემა ჩატვირთვისას მისაწვდომი გახდება მხოლოდ წასაკითხად. ეს ნიშნავს, რომ ყველაფერი რასაც გავაკეთებთ სისტემაში, მხოლოდ მეხსიერებაში მოხდება და გადატვირთვისას ჩვენს მიერ განხორციელებული ყველა ცვლილება დაიკარგება. ეს

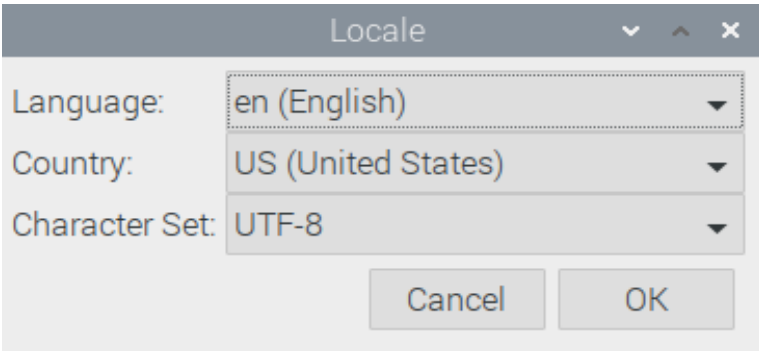
ძალიან მოსახერხებელია სასწავლო პროცესში: როდესაც სტუდენტების ერთი ჯგუფი დაასრულებს მუშაობას, პედაგოგს არ უწევს ძალისხმევის დახარჯვა იმაზე, რომ ახალი ჯგუფის სტუდენტებმაც ზუსტად იქიდან დაიწყონ მუშაობა, საიდანაც წინა ჯგუფმა. Boot Partition პარამეტრის ჩართვა გულისხმობს Read-only მდგომარეობას. მომხმარებელს არ შეუძლია Boot Partition-ში ცვლილების შეტანა.

Raspberry Pi-ს კონფიგურირების ფანჯრის ბოლო ჩანართი localisation საშუალებას გვაძლევს სისტემაში მივუთითოთ ჩვენი ადგილმდებარეობა, საათობრივი სარტყელი, კლავიატურის განლაგება და Wi-Fi მოდულის კონფიგურაცია.



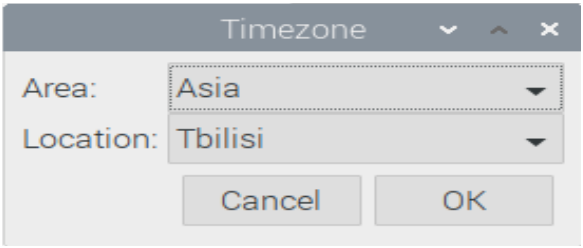
სურ. 18. Raspberry Pi-ს კონფიგურირების ფანჯარა

Set Locale პარამეტრზე დაჭერით მივიღებთ სურ. 19-ზე მოცემულ ფანჯარას, რომელშიც შესაძლებელია ენის (Language), ქვეყნის (Country) და სიმბოლოების ნაკრების (Character Set) არჩევა.



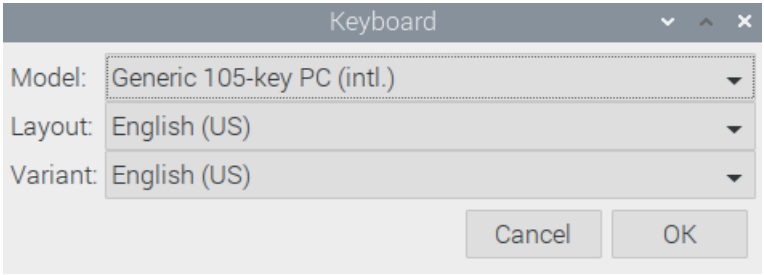
სურ. 19. Raspberry Pi-ს კონფიგურირება

Timezone პარამეტრები გამოიყენება Raspberry Pi-ზე შესაბამისი საათობრივი სარტყელის დასაყენებლად. ამისათვის უნდა ავირჩიოთ ჩვენი კონტინენტი და მდებარეობა.



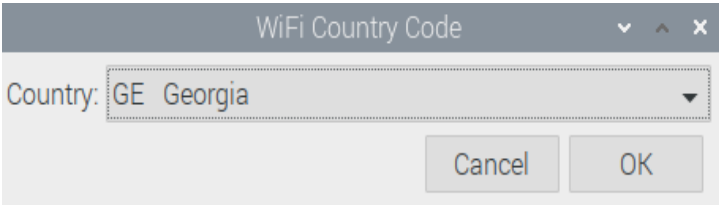
სურ. 20. Raspberry Pi-ს კონფიგურირება

Keyboard პარამეტრი საშუალებას გვაძლევს შევარჩიოთ ჩვენს მიერ გამოყენებული კლავიატურის ტიპი და განლაგება.



სურ. 21. Raspberry Pi-ს კონფიგურირება

Raspberry Pi-ს კონფიგურირების ფანჯრის Localisation ჩანართში Set Wi-Fi Country ველში დაჭერის შედეგად მივიღებთ სურ. 22-ზე მოცემულ ფანჯარას, სადაც ვირჩევთ ქვეყანას, რომელშიც ვიმყოფებით. ჩვენს ქვეყანაში მიღებული სტანდარტების მიხედვით, ავტომატურად ხდება Wi-Fi მოდულის სიხშირისა და გამოსხივების სიმძლავრის შერჩევა.

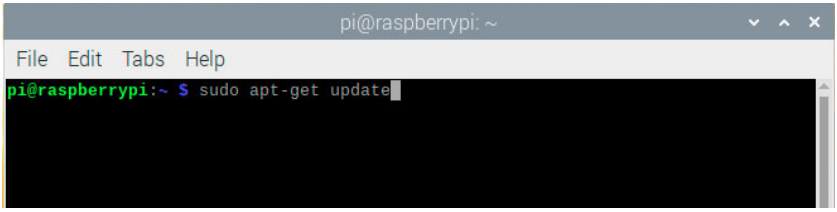


სურ. 22. Raspberry Pi-ს კონფიგურირება

შევნიშნავთ, რომ უმჯობესია რამდენიმე პარამეტრის ერთდროულად შეცვლა, OK ღილაკის დაჭერამდე. ეს მნიშვნელოვანია, რადგანაც სისტემაში ცვლილების შეტანა ხშირად მოითხოვს გადატვირთვას.

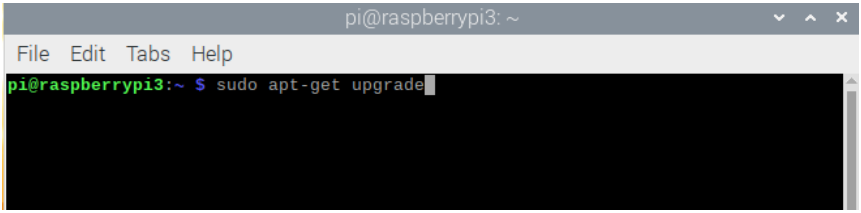
Raspberry Pi OS ოპერაციული სისტემის განახლება

იმისათვის, რომ განვაახლოთ Raspberry Pi ოპერაციული სისტემა და პროგრამული უზრუნველყოფა, ტერმინალის ფანჯარაში ავკრიფოთ ბრძანება *sudo apt update* (იგივეა, რაც *sudo apt-get update*)



სურ.23. Raspberry Pi ოპერაციული სისტემის და პროგრამული უზრუნველყოფის განახლება

ამის შემდეგ ტერმინალის ფანჯარაში ავკრიფოთ *sudo apt upgrade* ბრძანება. ამით დასრულდება სისტემის სრული პროგრამული განახლება.

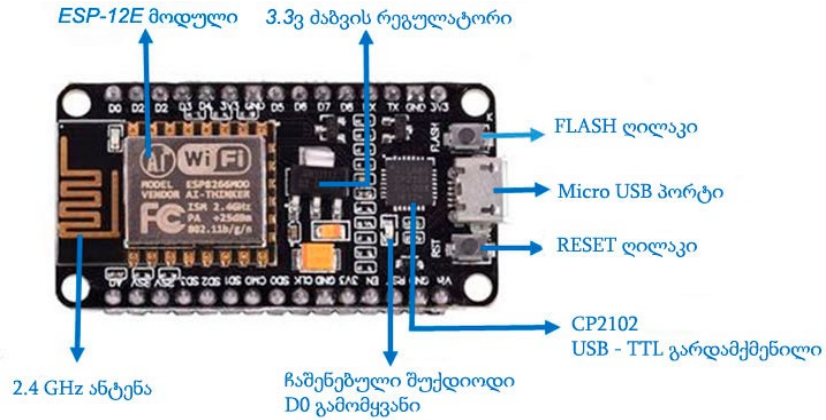


სურ. 24. Raspberry Pi ოპერაციული სისტემის და პროგრამული უზრუნველყოფის განახლების დასრულება

ამ ორი ბრძანების რეგულარული შესრულებით ჩვენი ოპერაციული სისტემა მუდამ იქნება განახლებულ მდგომარეობაში.

რა არის NodeMCU ESP8266?

ESP8266 წარმოადგენს იაფ IoT პლატფორმას WiFi კავშირით, ის შექმნილია ჩინური კომპანია Espressif-ის მიერ. დღეისათვის ESP8266 მოდული წარმოადგენს ერთ-ერთ ყველაზე პოპულარულ გადაწყვეტას IoT პროექტებისთვის Wi-Fi-ს ფუნქციის დასამატებლად. ჩვენ გამოვიყენებთ NodeMCU V3 დაფას (შემდგომში NodeMCU), რომელიც NodeMCU ESP8266-ის ერთ-ერთი ნაირსახეობაა.



სურ. 25. NodeMCU

NodeMCU დაფა აღჭურვილია ESP-12E მოდულით. მწარმოებლის ვებგვერდის მიხედვით, NodeMCU დაფის ტექნიკური მახასიათებლებია:

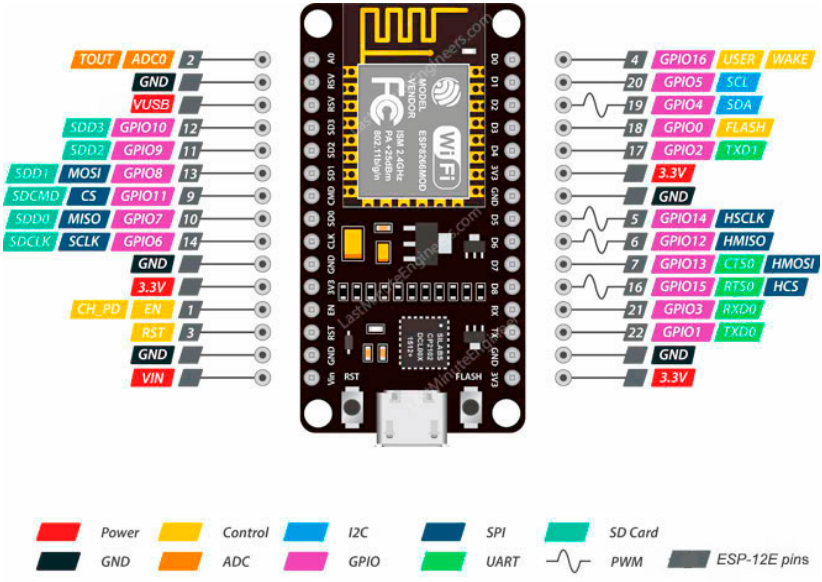
- GPIO ინტერფეისის ძაბვა: 3.3V;
- ანტენის ტიპი: ნაბეჭდ ფირფიტაში ჩაშენებული ანტენა;

- უსადენო ქსელის სტანდარტი: 802.11 b/g/n;
- Wi-Fi 2.4GHz სიხშირეზე, WPA/WPA2 უსაფრთხოების რეჟიმის მხარდაჭერა;
- STA/AP/STA + AP მუშაობის რეჟიმების მხარდაჭერა;
- TCP/IP პროტოკოლების ჩაშენებული სტეკი რამდენიმე TCP კლიენტთან დაკავშირების მხარდასაჭერად;
- D0 - D8, SD1 - SD3: გამოიყენება როგორც GPIO, PWM, I2C და.ა.შ. მაქსიმალური დატვირთვის დენი 15mA;
- AD0: 1 არხიანი ანალოგურ-ციფრული გარდამქმნელი (აცგ);
- შესასვლელი ძაბვა: 4.5V - 9V, ასევე შესაძლებელია USB-დან კვება;
- მოხმარებული დენი: მუშაობის რეჟიმი: ≈70mA (200mA მაქსიმალური მნიშვნელობა), მოლოდინის რეჟიმი: <200μA;
- გადაცემის სიჩქარე: 110-460800 bps;
- UART მონაცემთა გადაცემის ინტერფეისის მხარდაჭერა;
- ჩაშენებული პროგრამული უზრუნველყოფის დისტანციურად განახლების შესაძლებლობა;
- ფლეშ-მეხსიერების მოცულობა: 4MB

ESP8266-ში ინტეგრირებულია 802.11b/g/n HT40 Wi-Fi ტრანსივერი, ამიტომ მას არა მხოლოდ შეუძლია WiFi ქსელთან შეერთება და ურთიერთქმედება ინტერნეტთან,

არამედ საკუთარი ქსელის შექმნაც, რომელსაც შეუძლია პირდაპირ შეუერთდნენ სხვა მოწყობილობები. ეს NodeMCU-ს კიდევ უფრო უნივერსალურს ხდის.

NodeMCU-ს გააჩნია 30 საკონტაქტო გამომყვანი. საკონტაქტო გამომყვანების განლაგება მოცემულია სურ. 26-ზე.



სურ. 26. NodeMCU-ს საკონტაქტო გამომყვანების განლაგება

სიმარტივისთვის ქვემოთ დაჯგუფებულია მსგავსი ფუნქციების მქონე საკონტაქტო გამომყვანები და მოცემულია მათი მოკლე აღწერა:

კვების საკონტაქტო გამომყვანები. NodeMCU-ს დაფაზე გვაქვს კვების 5 საკონტაქტო გამომყვანი: VIN, VUSB და სამი

3.3V. VIN საკონტაქტო გამომყვანი შეიძლება გამოვიყენოთ NodeMCU-ს და მისი პერიფერიული მოწყობილობების კვებისთვის, იმ შემთხვევაში, თუ გვაქვს 5V ძაბვის სტაბილიზებული წყარო. 3.3V საკონტაქტო გამომყვანები წარმოადგენენ გამოსასვლელებს დაფაზე ჩაშენებული ძაბვის რეგულატორისთვის. ეს გამომყვანები შეიძლება გამოვიყენოთ გარე კომპონენტების კვებისთვის. VUSB (VU) საკონტაქტო გამომყვანი დაკავშირებულია +5V-თან, NodeMCU-ს USB გასართზე. VUSB გამომყვანი შეიძლება გამოვიყენოთ გარე კომპონენტების კვებისთვის, როდესაც ის მიერთებულია USB-სთან, USB კვების წყაროს სიმძლავრის ფარგლებში.

დამიწება. GND არის დამიწების საკონტაქტო გამომყვანი NodeMCU დაფისთვის.

I2C საკონტაქტო გამომყვანები გამოიყენება I2C სენსორებისა და პერიფერიული მოწყობილობების მისაერთებლად.

GPIO საკონტაქტო გამომყვანები. NodeMCU-ს აქვს 17 GPIO გამომყვანი, რომელთაც პროგრამულად შეიძლება სხვადასხვა ფუნქცია მივანიჭოთ.

ADC (აცგ) არხი. NodeMCU-ში ჩაშენებულია 10-ბიტი სიზუსტის ADC.

UART საკონტაქტო გამომყვანები. NodeMCU-ს აქვს 2 UART ინტერფეისი: UART0 და UART1, ისინი უზრუნველყოფენ ასინქრონულ კომუნიკაციას (RS232 და RS485), შეუძლიათ მონაცემთა გაცვლა 4.5Mbps სიჩქარეზე.

SPI (Serial Peripheral Interface) საკონტაქტო გამომყვანები. ამ გამომყვანებს აქვთ შემდეგი ფუნქციების მხარდაჭერა:

- SPI ფორმატის გადაცემის სინქრონიზაციის 4 რეჟიმი;
- სიხშირე 80 MHz-მდე და განაწილებული ტაქტური სიხშირე 80 MHz;
- 64 ბაიტამდე FIFO.

SDIO (Secure Digital Input/Output Interface) საკონტაქტო გამომყვანები გამოიყენება SD ბარათთან უშუალო კომუნიკაციისთვის.

PWM საკონტაქტო გამომყვანები. NodeMCU-ს დაფაზე 4 PWM არხია. PWM გამოსასვლელის რეალიზება შესაძლებელია პროგრამულად. PWM-ის სიხშირე რეგულირდება 100Hz-დან 1 kHz დიაპაზონში.

მართვის საკონტაქტო გამომყვანები გამოიყენება ESP8266-ის სამართავად.

- EN გამომყვანი – ESP8266 აქტიურდება, როდესაც EN კონტაქტი არის HIGH მდგომარეობაში; როდესაც EN იმყოფება LOW მდგომარეობაში, მაშინ მიკროსქემა მუშაობს მინიმალურ სიმძლავრეზე.
- RST გამომყვანი გამოიყენება ESP8266 მიკროკონტროლერის გადასატვირთად.
- WAKE გამომყვანი გამოიყენება მიკროკონტროლერის ძილის რეჟიმიდან გამოსაყვანად.

NodeMCU დაფის Arduino IDE-ში დაპროგრამებისას, ჩვენ შეგვიძლია გამოვიყენოთ A0 საკონტაქტო გამომყვანი, როგორც ADC და D0-D10 გამომყვანები, როგორც ციფრული გამომყვანები (D9 არის RX და D10 არის TX).

ქვემოთ მოცემულ ცხრილში აღწერილია NodeMCU დაფის ის გამომყვანები, რომლებსაც ვიყენებთ Arduino IDE-ში დაპროგრამებისთვის.

დაფა	GPIO	გამოყენება	კომენტარი
A0	ADC0	მხოლოდ შესასვლელი	ანალოგური შესასვლელი
D0	GPIO16	Wake up	მოჭიმულია მიწასთან (pulled LOW), ჩატვირთვისას იმყოფება მაღალ ლოგიკურ მდგომარეობაში
D1	GPIO5	SCL I2C	
D2	GPIO4	SDA I2C	
D3	GPIO0	FLASH	მოჭიმულია პლუსისკენ (pulled HIGH), არ შეიძლება მოჭიმვა მიწისკენ
D4	GPIO2	ჩაშენებული შუქდიოდი	მოჭიმულია პლუსისკენ (pulled HIGH), არ შეიძლება მოჭიმვა მიწისკენ

D5	GPIO14	SCLK SPI	
D6	GPIO12	MISO SPI	
D7	GPIO13	MOSI SPI	
D8	GPIO15	CS SPI	მოჭიმულია მიწისკენ (pulled low), არ შეიძლება მოჭიმვა პლუსისკენ
RX	GPIO3	RX	შეიძლება გამოყენებულ იქნეს, როგორც შესასვლელი, ჩატვირთვისას იმყოფება მაღალ ლოგიკურ მდგომარეობაში
TX	GPIO1	TX	შეიძლება გამოყენებულ იქნეს, როგორც გამოსასვლელი, ჩატვირთვისას იმყოფება მაღალ ლოგიკურ მდგომარეობაში

ცხრილი 1. NodeMCU დაფის გამომყვანების აღწერა

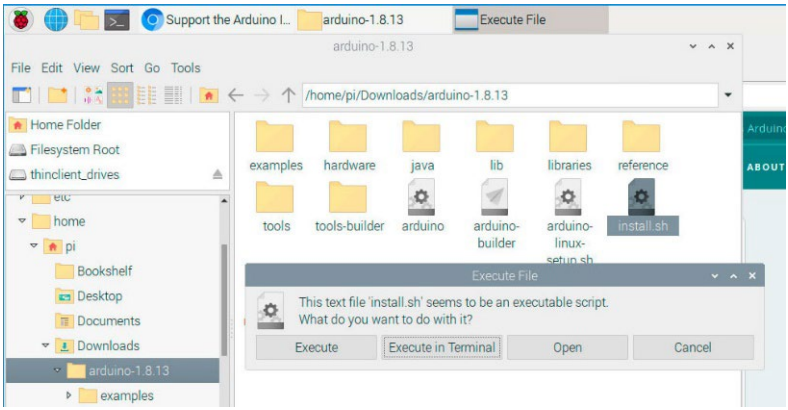
არდუინო IDE –ს ინსტალაცია

NodeMCU-ს დასაპროგრამებლად, დაპროგრამების გარემოდ გამოვიყენებთ Arduino IDE-ს, ამისათვის საჭიროა არდუინოს ვებსაიტიდან (arduino.cc/en/software) ჩამოვტვირთოთ Arduino IDE და დავაინსტალიროთ Raspberry Pi-ში. ვებსაიტზე დავინახავთ Arduino IDE-ს უახლეს ვერსიას, ოპერაციული სისტემების ნუსხიდან ავირჩიოთ ის ოპერაციული სისტემა, რომელსაც ჩვენ ვიყენებთ (Linux ARM 32 bit).

სურ. 27. Arduino IDE-ს ინსტალაცია

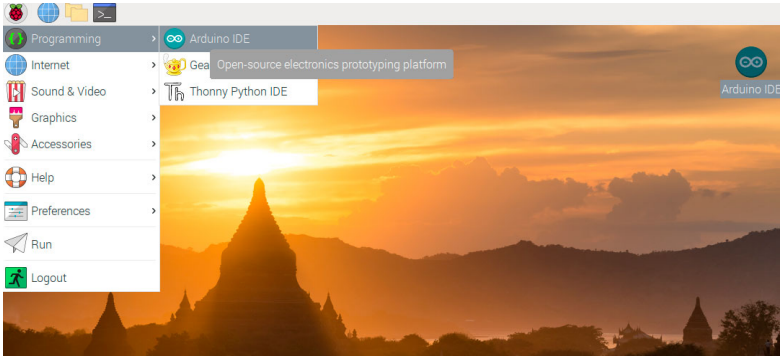
Arduino IDE-ს ჩამოტვირთვის დასრულების შემდეგ, განვაარქივოთ საქაღალდე.

Arduino IDE-ს დასაინსტალირებლად ორჯერ დავაწკაპუნოთ install.sh ფაილზე და შემდეგ - Execute ან Execute in Terminal ღილაკზე (სურ. 28).



სურ. 28. Arduino IDE-ს ინსტალაცია

Arduino IDE დაინსტალირების შემდეგ გამოჩნდება Programming მენიუში ან Raspberry Pi-ს სამუშაო მაგიდაზე (Desktop).



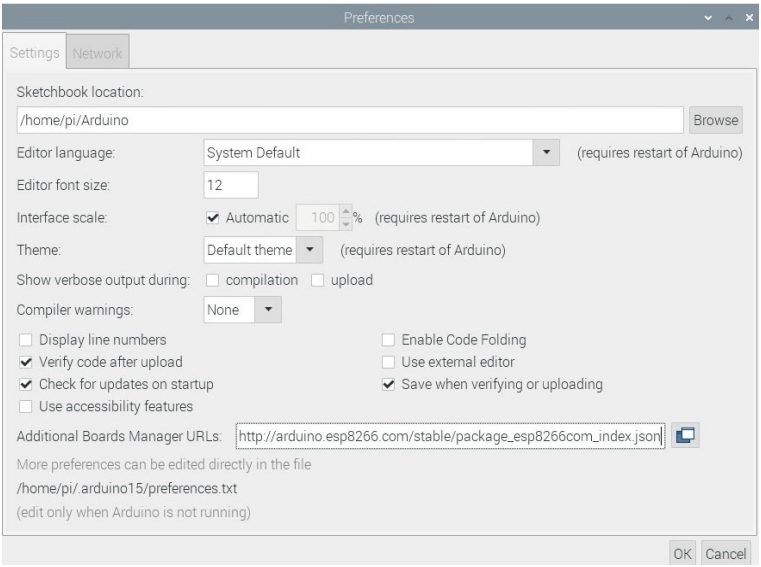
სურ. 29. Arduino IDE-ს გაშვება

Arduino IDE-ში მუშაობის დაწყებამდე, პირველ რიგში უნდა მივუთითოთ, რომელ დაფასთან ვიმუშავებთ. ამისათვის Arduino IDE-ში გავხსნათ File->Preferences

ფანჯარა და Additional Board Manager URLs ველში ჩავსვათ მისამართი:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

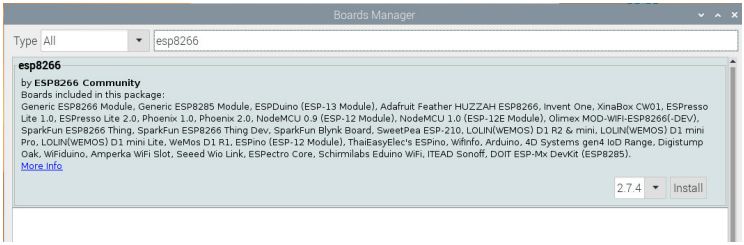
იმ შემთხვევაში, თუ საჭიროა სხვა მისამართების დამატება, მათი ერთმანეთისგან გამოსაყოფად გამოვიყენოთ მძიმე (მაგალითად, ESP32-თვის ჩავწეროთ შემდეგი მისამართი: https://dl.espressif.com/dl/package_esp32_index.json)



სურ. 30. ESP8266 პლატფორმის შემცველი დაფების ნუსხის მისამართის მითითება

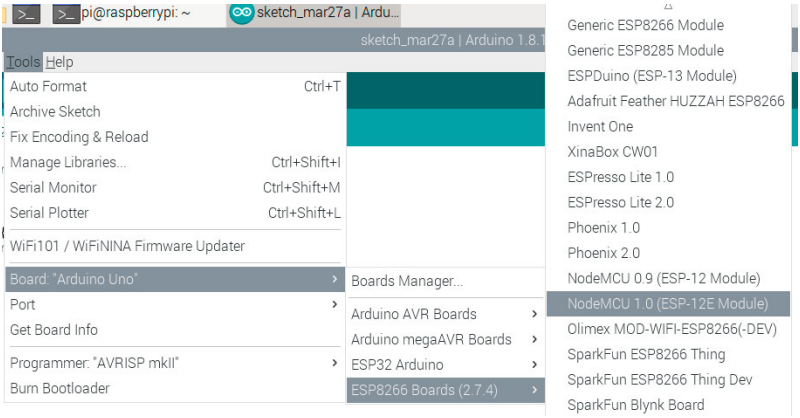
ზემოთ აღნიშნული პროცედურის შესრულების შემდეგ Tools-> Board მენიუდან გავხსნათ Boards Manager ფანჯარა და დავაინსტალიროთ ESP8266 პლატფორმა. დაფის მენეჯერი ჩამოტვირთავს ESP8266 პლატფორმის შემცველი დაფების

შესაბამის ფაილებს იმ მისამართიდან, რომელიც ჩვენ უკვე მივუთითეთ File->Preferences ფანჯარაში და დააინსტალირებს მათ.



სურ. 31. ESP8266 პლატფორმის დაინსტალირება

ESP8266 პლატფორმის დაინსტალირების შემდეგ, ESP8266-ის შემცველი დაფების დასახელებები გამოჩნდება ისე, როგორც ეს ქვემოთ მოცემულ სურათზეა ნაჩვენები. ახლა ჩვენ უკვე შეგვიძლია ავირჩიოთ NodeMCU 1.0 (ESP-12E Module) Tools->Board მენიუდან.



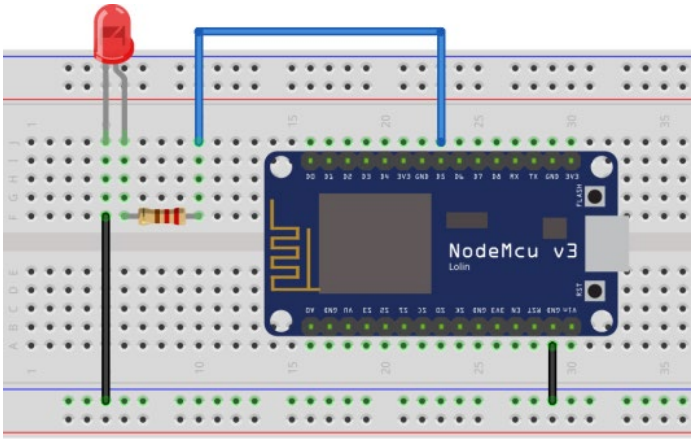
სურ. 32. NodeMCU 1.0 (ESP-12E Module)-ის არჩევა

პირველი სკეტჩის ატვირთვა NodeMCU-ზე

იმისათვის, რომ ჩვენი პირველი სკეტჩი ავტვირთოთ NodeMCU დაფაზე, პირველ რიგში, NodeMCU უნდა მივუერთოთ Raspberry Pi-ს და დავრწმუნდეთ, რომ Arduino IDE-ში დაინსტალირებულია ESP8266 პლატფორმა და Tools-> Board მენიუში მითითებულია NodeMCU დაფა.

ჩვენი პირველი სკეტჩის განსახორციელებლად, შემდეგი კომპონენტები დაგვჭირდება:

- შუქდიოდი
- რეზისტორი (220 Ω)
- სამაკეტო დაფა
- სამონტაჟო შემაერთებლები (Jumper wires)
- Micro USB კაბელი (გამოიყენება NodeMCU დაფის კომპიუტერთან (Raspberry Pi) შესაერთებლად).



სურ. 33. NodeMCU-ს გამოსასვლელზე მაღალი და დაბალი დონის სიგნალების მიხედვით შუქდიოდის მდგომარეობის ცვლილება

ამოცანის მიზანია შუქდიოდის ანთება და ჩაქრობა 1 წამიანი ინტერვალით. სქემის აწყობა დავიწყოთ სამაკეტო დაფაზე შუქდიოდის მოთავსებით. 220Ω წინაღობის მქონე რეზისტორის ერთი გამომყვანი მივაერთოთ შუქდიოდის ანოდთან, მეორე გამომყვანი კი სამონტაჟო შემაერთებლის მეშვეობით - NodeMCU დაფის D5 გამომყვანთან. ავიღოთ კიდევ ერთი სამონტაჟო შემაერთებელი, მისი ერთი ბოლო დავუკავშიროთ შუქდიოდის კათოდს, მეორე ბოლო - NodeMCU დაფის GND („მიწა“) გამომყვანს.

ჩვენ მოვახდენთ NodeMCU დაფის D5 გამოსასვლელის კონფიგურირებას ისე, რომ მასზე 1 წამიანი ინტერვალით, 1 წამის ხანგრძლივობით მივიღოთ დაბალი და მაღალი დონის სიგნალები. ეს პროცესი უსასრულოდ გრძელდება.

დავწეროთ შესაბამისი კოდი:

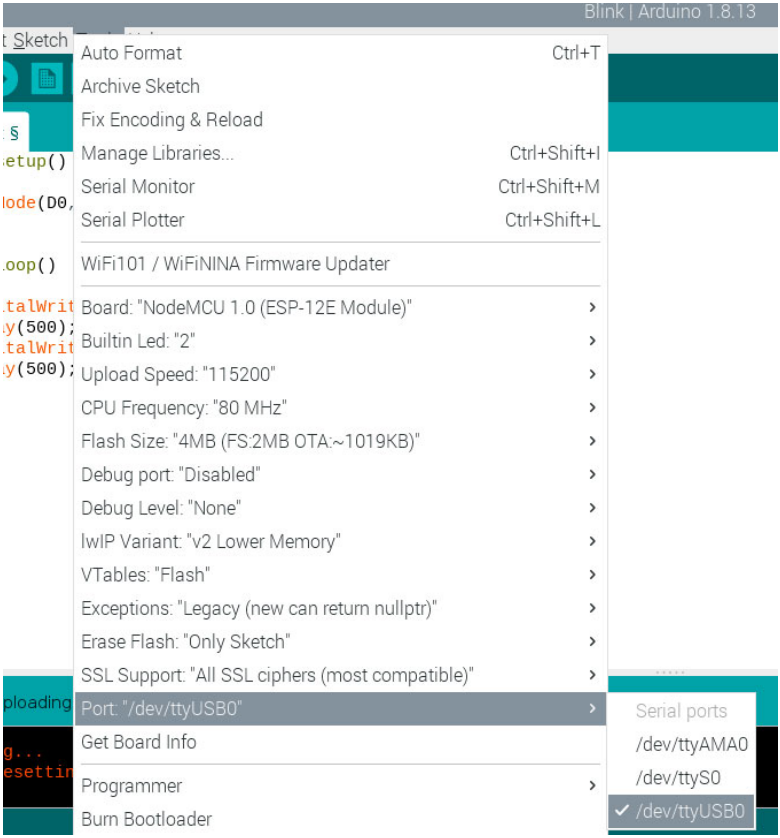
```
#define led_pin D5
void setup() {
  //შუქდიოდის საკონტაქტო გამომყვანი, როგორც
გამოსასვლელი
  pinMode(led_pin, OUTPUT);
}
void loop() {
  digitalWrite(led_pin, HIGH);
  //დაყოვნების დრო 1 წმ
  delay(1000);
  digitalWrite(led_pin, LOW);
  //დაყოვნების დრო 1 წმ
  delay(1000);
}
```

ჩვენი კოდი Arduino IDE-ში გამოიყურება ისე, როგორც ეს ქვემოთ მოცემულ სურათზეა ნაჩვენები:

სურ. 34. ჩვენი პირველი სკეტჩი არდუინო IDE-ში

სანამ NodeMCU-ზე კოდს ავტვირთავდეთ, ვნახოთ Arduino IDE-ში მითითებულია თუ არა შესაბამისი COM პორტი. Windows სისტემისთვის, Tools->Port მენიუში პორტი გამოჩნდება COM#, Mac/Linux სისტემების შემთხვევაში -

/dev/ttyUSBXX სახით. Tools->upload speed მენიუში მოვნიშნოთ ატვირთვის სიჩქარე: 115200.



სურ. 35. მიმდევრობითი პორტის მითითება Arduino IDE-ში

შესაბამისი პორტის მითითების შემდეგ დავაჭიროთ ატვირთვის ღილაკს Arduino IDE-ში. კოდის NodeMCU-ზე

ატვირთვის შემდეგ, შუქდიოდი ციმციმს დაიწყებს. შესაძლოა დაგვჭირდეს RST ღილაკი კოდის ხელახლა გასაშვებად.

პროგრამაში გამოყენებული გვაქვს digitalWrite() ფუნქცია, რომლის საშუალებითაც შეგვიძლია ვცვალოთ გამოსასვლელზე სიგნალის ლოგიკური დონეები (მაღალი ან დაბალი), ეს კი თავის მხრივ ცვლის შუქდიოდის მდგომარეობას.

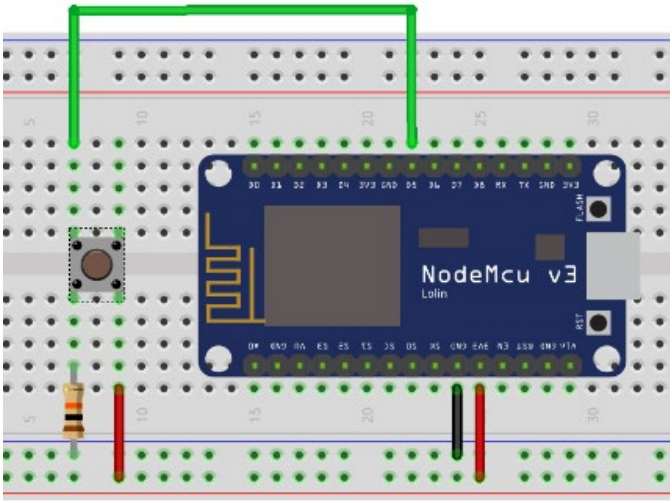
ციფრული სიგნალის წაკითხვა

NodeMCU-ს GPIO გამომყვანების გამოყენება შესაძლებელია ციფრული სიგნალის წასაკითხად. ეს საშუალებას მოგვცემს ვმართოთ ჩვენი პროგრამა შესასვლელი სიგნალების გამოყენებით, ასევე გავუწიოთ მონიტორინგი სენსორიდან მიღებულ მონაცემებს.

ქვემოთ განხილულ მაგალითში ნაჩვენებია, როგორ წავიკითხოთ ციფრული სიგნალები GPIO გამომყვანების გამოყენებით.

ავაწყოთ სქემა (სურ. 36). როგორც სურათიდან ჩანს, NodeMCU-ს D5 გამომყვანი, მომჭიმავი (Pull-down, 4.7-10 კილოომი) რეზისტორის გავლით, მიერთებულია მიწასთან (GND). ჩვენ დავაკონფიგურირებთ D5-ს, როგორც შესასვლელს, შემდეგ წავიკითხავთ ამ საკონტაქტო გამომყვანზე სიგნალის მნიშვნელობას digitalRead() ფუნქციის საშუალებით. მიღებულ მნიშვნელობას გამოვიტანთ

მიმდევრობითი მონიტორის ფანჯარაში. ეს პროცედურა უნდა განმეორდეს ყოველ 1 წამში.

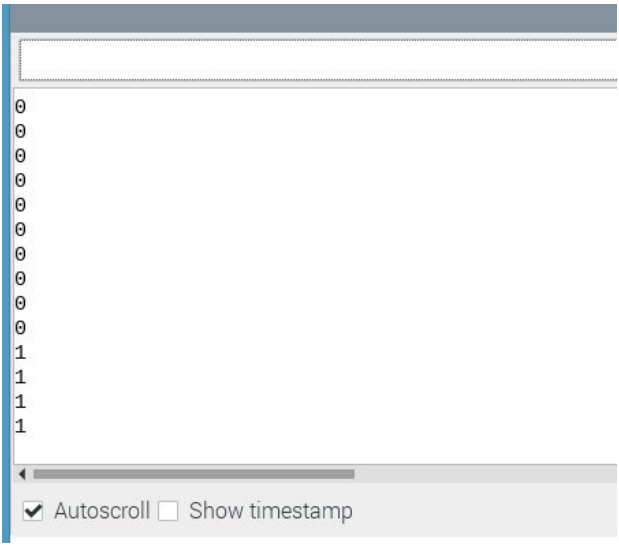


სურ. 36. ციფრული სიგნალების წაკითხვა GPIO გამომყვანის გამოყენებით

```
#define inputPin D5
int val = 0;
void setup() {
  Serial.begin(115200);
  pinMode(inputPin, INPUT);
}
void loop() {
  // D5 შესასვლელზე სიგნალის მნიშვნელობის წაკითხვა
  val = digitalRead(inputPin);
  //D5 შესასვლელზე სიგნალის მნიშვნელობების გამოტანა მიმდევრობით მონიტორში
  Serial.println(val);
  delay(1000);
}
```

მიმდევრობითი მონიტორის ფანჯრის გახსნისას, დავინახავთ, რომ ეკრანზე გამოჩნდება ნული, შემდეგ ისევ ნული და ა.შ. ეს ნიშნავს, რომ D5 შესასვლელზე წაკითხული სიგნალების მნიშვნელობები ნულის ტოლია. როგორც პროგრამის კოდიდან ჩანს, პროგრამის დასაწყისში ცხადდება GPIO საკონტაქტო გამომყვანის სახელი, შემდეგ ვაცხადებთ val ცვლადს, სადაც ამ გამომყვანის მდგომარეობა ინახება. void setup() ფუნქციაში მითითებულია მიმდევრობითი კომუნიკაციის პორტის სიჩქარე 115200 ბაუდი და GPIO 14 (D5) გამომყვანი კონფიგურირებულია, როგორც შესასვლელი. სკეტჩის loop ნაწილიდან ჩანს, რომ პროგრამა კითხულობს D5 შესასვლელზე ციფრული სიგნალის მნიშვნელობას და ამ მნიშვნელობას ინახავს val ცვლადში. Serial.println(val) ბრძანება უზრუნველყოფს val ცვლადში შენახული მნიშვნელობის გამოტანას მიმდევრობითი მონიტორის ფანჯარაში. დაყოვნების დრო არის 1 წმ.

როგორც ზემოთ უკვე ავღნიშნეთ, მიმდევრობითი მონიტორის ფანჯარაში მხოლოდ ნულები ჩანს. ეს ასე იქნება სულ, რადგანაც D5 გამომყვანი GND-სთან არის მიერთებული. იმისათვის, რომ D5 შესასვლელზე სიგნალის მნიშვნელობა 1-ის ტოლი გახდეს, ის უნდა მივაერთოთ 3.3V გამომყვანთან. ჩვენს სქემაში, ამ შეერთებას უზრუნველყოფს ღილაკზე დაჭერა. ღილაკზე დაჭერისას მიმდევრობითი მონიტორის ფანჯარაში დავინახავთ ერთიანს ნულის ნაცვლად, რადგანაც D5 შესასვლელზე უკვე მაღალი დონის სიგნალი გვაქვს.

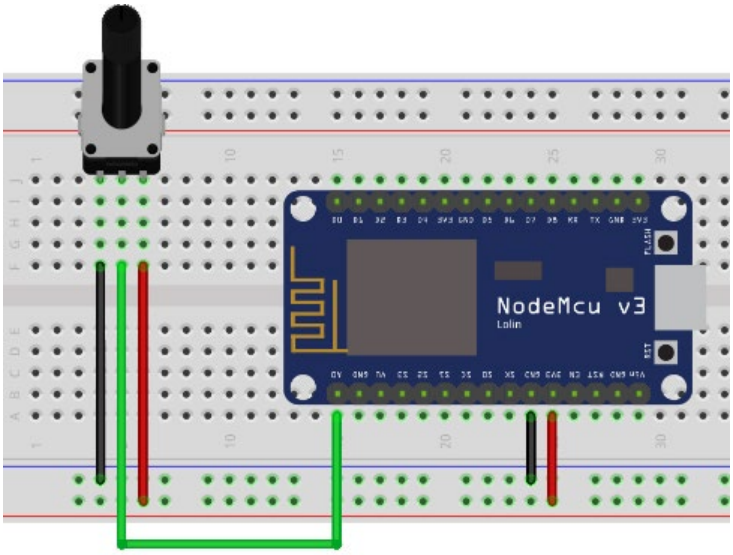


სურ. 37. NodeMCU-ს D5 გამომყვანზე წაკითხული ციფრული სიგნალის მნიშვნელობები

ანალოგური სიგნალის წაკითხვა

NodeMCU-ს აქვს ერთი ანალოგური საკონტაქტო გამომყვანი (A0), მისი გამოყენება შეიძლება ანალოგური სიგნალების წასაკითხად. განვიხილოთ, როგორ შეიძლება დავწეროთ სკეტჩი, რომელიც ანალოგური სიგნალების წაკითხვის საშუალებას მოგვცემს.

ავაწყოთ სქემა ქვემოთ მოცემული სურათის მიხედვით. 10 kΩ პოტენციომეტრის გამომყვანები შევაერთოთ NodeMCU დაფის A0, GND და 3.3V გამომყვანებს (სურ. 38).



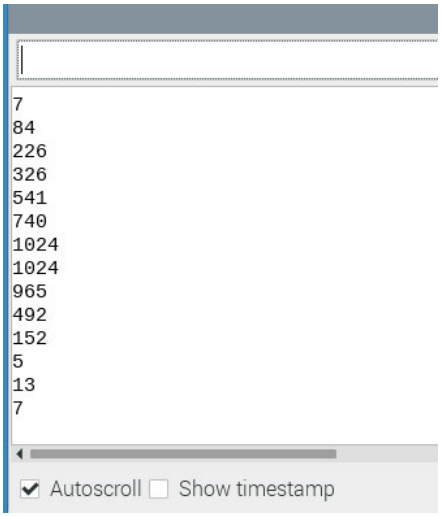
სურ. 38. ანალოგური სიგნალის წაკითხვა

analogRead() ფუნქციას გამოვიყენებთ ანალოგურ A0 (ანალოგურ-ციფრული გარდამქმნელის შესასვლელი) შესასვლელზე ანალოგური სიგნალის წასაკითხად, ანალოგური სიგნალის მნიშვნელობებს კი გამოვიტანთ მიმდევრობითი მონიტორის ფანჯარაში. ეს პროცედურა უნდა განმეორდეს ყოველ 1 წამში.

```
int val = 0;
void setup() {
  Serial.begin(115200);
}
void loop() {
  // ანალოგური სიგნალის მნიშვნელობების წაკითხვა
  val = analogRead(A0);
```

```
  // სიგნალის მნიშვნელობების გამოტანა მიმდევრობითი მონიტორის ფანჯარაში
  Serial.println(val);
  delay(1000);
}
```

როგორც პროგრამის კოდიდან ჩანს, val ცვლადში ინახება ანალოგური სიგნალის A0 შესასვლელზე ჩვენს მიერ წაკითხული მნიშვნელობები. სკეტჩის setup() ნაწილში მიმდევრობითი კომუნიკაციის პორტის სიჩქარედ მითითებული გვაქვს 115200 ბაუდი. Serial.println(val) ბრძანება უზრუნველყოფს val ცვლადში შენახული ანალოგური სიგნალის მნიშვნელობების გამოტანას 1 წამიანი ინტერვალით მიმდევრობითი მონიტორის ფანჯარაში.

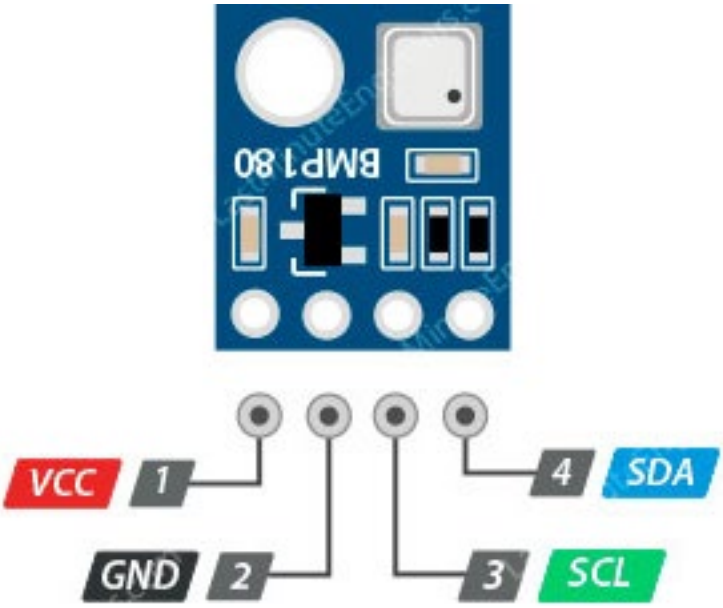


სურ. 39. ანალოგური სიგნალის მნიშვნელობები მიმდევრობითი მონიტორის ფანჯარაში

მონაცემების წაკითხვა ციფრული სენსორიდან

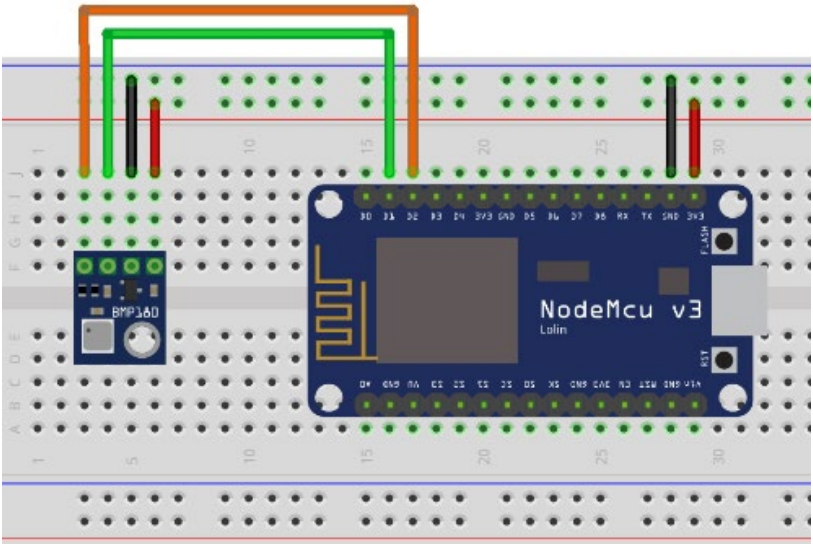
NodeMCU დაფა შეიძლება გამოვიყენოთ როგორც ციფრული, ასევე ანალოგური სენსორებიდან მონაცემების წასაკითხად და მათი შემდგომი მონიტორინგისთვის. ამის სადემონსტრაციოდ, ქვემოთ განვიხილავთ მაგალითს, სადაც მივიღებთ მონაცემებს ტემპერატურისა და ატმოსფერული წნევის ციფრული BMP180 სენსორიდან, შემდეგ კი მონაცემებს გამოვიტანთ მიმდევრობით მონიტორში.

BMP180 სენსორის საკონტაქტო გამომყვანების კონფიგურაცია მოცემულია სურ. 40-ზე:



სურ. 40. BMP180 სენსორის საკონტაქტო გამომყვანების კონფიგურაცია

ავაგოთ სქემა (სურ. 41) მოცემული შეერთებების მიხედვით: GND <-> GND, 3V3 <-> VCC, D1 <-> SCL, D2 <-> SDA



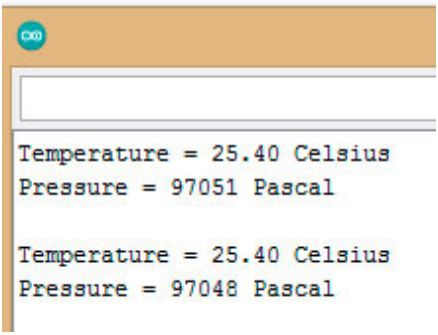
სურ. 41. მონაცემების წაკითხვა BMP180 სენსორიდან

ჩვენ დაგვჭირდება Adafruit BMP085-ის ბიბლიოთეკა. მისი გადმოწერა შესაძლებელია მისამართიდან https://github.com/adafruit/Adafruit-BMP085-Library, თუმცა ამის გაკეთება უფრო მარტივია Arduino IDE-ში, Sketch->Include Library> Manage Libraries მენიუს გამოყენებით. ამისათვის ბიბლიოთეკების ნუსხაში უნდა მოვძებნოთ შესაბამისი ბიბლიოთეკა და შემდეგ დავაინსტალიროთ.

```
//Adafruit BMP085/BMP180-ის ბიბლიოთეკა
#include <Adafruit_BMP085.h>
//BMP180 სენსორის ინიციალიზაცია
Adafruit_BMP085 bmp_sensor;
void setup() {
  Serial.begin(115200);
  /*bmp.begin ფუნქცია აბრუნებს ლოგიკურ
მნიშვნელობას, სისტემის მიერ  BMP-180 სენსორის აღქმის
მიხედვით. თუ სენსორის დეტექცია მოხდა, მაშინ
ფუნქცია აბრუნებს true-ს, წინააღმდეგ შემთხვევაში
false-ს  */
  if (!bmp_sensor.begin()) {
    /*თუ bmp.begin ფუნქციის მნიშვნელობა არის false,
ეს ნიშნავს რომ სისტემამ ვერ დაინახა სენსორი. ეკრანზე
გამოდის შესაბამისი შეტყობინება*/
   Serial.println("Could not find BMP180 or BMP085
sensor at address 0x77");
//უსასრულო ციკლი
    while (1) {}
  }
}
void loop() {
  Serial.print("Temperature = ");
  /*ტემპერატურის მნიშვნელობის წაკითხვა სენსორიდან
და მისი გამოტანა მიმდევრობით მონიტორში  */
  Serial.print(bmp_sensor.readTemperature());
  Serial.println(" Celsius");
  /*წნევის მნიშვნელობის წაკითხვა სენსორიდან და მისი
გამოტანა მიმდევრობით მონიტორში*/
  Serial.print("Pressure = ");
  Serial.print(bmp_sensor.readPressure());
  Serial.println(" Pascal");
  Serial.println();
```

```
  //5 წამიანი დაყოვნება სენსორიდან მონაცემების
წაკითხვებს შორის
  delay(5000);
}
```

სკეტჩის გაშვების შემდეგ მიმდევრობითი მონიტორის ფანჯარაში დავინახავთ სენსორიდან მიღებულ ტემპერატურისა და წნევის მნიშვნელობებს:



სურ. 42. BMP180 სენსორიდან მიღებული გარემოს ტემპერატურისა და ატმოსფერული წნევის მნიშვნელობები

NodeMCU დაფის Wi-Fi ქსელთან დაკავშირება

იმისათვის, რომ NodeMCU დაფის გამოყენებით შევძლოთ მონაცემების ინტერნეტის ქსელით გადაცემა ან მიღება, ადვილია იმის მიხვედრა რომ NodeMCU დაფა ჩვენს Wi-Fi ქსელთან უნდა დავაკავშიროთ. დაკავშირება ხდება სულ თავიდან, ერთხელ, პროგრამის setup() ნაწილში.

NodeMCU-ს Wi-Fi ქსელთან დასაკავშირებლად დაგვჭირდება ESP8266-ის ბიბლიოთეკის პროგრამაში ჩართვა, Wi-Fi ქსელის სახელის (ssid) და პაროლის მითითება.

ქვემოთ მოცემულ სკეტჩში, მიმდევრობითი მონიტორის ფანჯარაში გამოვიტანთ NodeMCU-ს Wi-Fi ქსელთან დაკავშირების დამადასტურებელ შეტყობინებას და NodeMCU დაფის IP მისამართს.

```
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან
დაკავშირებისთვის
#include <ESP8266WiFi.h>
//Wi-Fi ქსელის სახელის მითითება
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
void setup() {
  Serial.begin(115200);
  // Wi-Fi ქსელთან დაკავშირება
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //WiFi ბიბლიოთეკის ქსელური პარამეტრების
ინიციალიზაცია
  WiFi.begin(ssid, wifiPassword);
  /*  Wi_Fi ქსელთან დაკავშირება. თუ status-ის
მნიშვნელობა არ არის WL_CONNECTED-ის ტოლი, მაშინ
დაელოდე 500 მილიწამი და შეამოწმე სტატუსი ხელახლა.
WL_CONNECTED სტატუსი ენიჭება მაშინ, როდესაც Wi-Fi
ქსელთან კავშირი გვექნება*/
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
```

```
  Serial.println("IP address: ");
  //NodeMCU-ს IP მისამართი ლოკალურ ქსელში
  Serial.println(WiFi.localIP());
}
void loop() {
}
```

პროგრამის გაშვების შედეგად მიმდევრობითი მონიტორის ფანჯარაში მივიღებთ:

სურ. 43. NodeMCU დაფის Wi-Fi ქსელთან დაკავშირება და NodeMCU-ს IP მისამართის ბეჭდვა

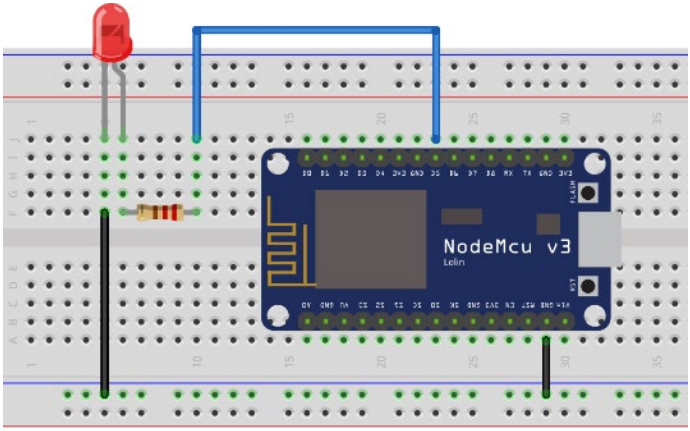
ESP8266WiFi ბიბლიოთეკის WiFi.begin() ფუნქციის საშუალებით ხდება ქსელის პარამეტრების (სახელისა და პაროლი) ინიციალიზაცია, შემდეგ სრულდება WiFi.status () ფუნქცია. თუ WiFi.status() ფუნქციამ დააბრუნა WL_Connected მნიშვნელობა, ეს ნიშნავს რომ NodeMCU დაფა წარმატებით დაუკავშირდა WiFi ქსელს.

თუ NodeMCU დაფის Wi-Fi ქსელთან კავშირი ვერ ხერხდება, პროგრამა ხელახლა ცდილობს WiFi.status()

ფუნქციის დახმარებით ქსელთან დაკავშირებას. ეს პროცესი მეორდება მანამ, სანამ WiFi.status () ფუნქცია არ დააბრუნებს WL_Connected მნიშვნელობას. წარმატებული კავშირის შემდეგ, პროგრამას მიმდევრობითი მონიტორის ფანჯარაში გამოაქვს შესაბამისი შეტყობინება და NodeMCU დაფის IP მისამართი ლოკალურ ქსელში.

NodeMCU დაფის გამოყენებით შუქდიოდის მდგომარეობის მართვა ლოკალურ უსადენო ქსელში

ის, რასაც ჩვენ ქვემოთ განვიხილავთ, სხვა არაფერია თუ არა მარტივი IoT პროექტი. ამოცანა შეეხება ვებ გვერდიდან Wi-Fi ქსელთან დაკავშირებული NodeMCU დაფის მეშვეობით შუქდიოდის მართვას. ავაგოთ სქემა ქვემოთ მოცემული სურათის მიხედვით (სურ. 44).



სურ. 44. Wi-Fi ქსელში ჩართული შუქდიოდის მართვა NodeMCU დაფის გამოყენებით

```
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან
დაკავშირებისთვის
#include <ESP8266WiFi.h>
#define ssid "your_wifi_ssid"
#define wifiPassword "your_wifi_password"
//D5 გამომყვანი NodeMCU-ზე
int ledPin = D5;
/*სერვერის შექმნა, რომელიც მითითებულ პორტზე
არსებულ კავშირებს უწევს მონიტორინგს */
WiFiServer server(80);
/*კლიენტის შექმნა, რომელსაც შეუძლია დაუკავშირდეს
მითითებულ IP მისამართსა და პორტს*/
WiFiClient client;
void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  //Wi-Fi ქსელთან დაკავშირება
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //WiFi ბიბლიოთეკის ქსელური პარამეტრების
ინიციალიზაცია
  WiFi.begin(ssid, wifiPassword);
   /*Wi_Fi ქსელთან დაკავშირება. თუ status-ის მნიშვ-
ნელობა არ არის WL_CONNECTED-ის ტოლი, მაშინ დაელოდე
500 მილიწამი და შეამოწმე სტატუსი ხელახლა.
WL_CONNECTED სტატუსი ენიჭება მაშინ, როდესაც Wi-Fi
ქსელთან კავშირი გვექნება*/
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }
  Serial.println("");
  Serial.println("WiFi connected");
   /*server.begin() ფუნქცია უზრუნველყოფს შემავალი კავშირების მონიტორინგს სერვერის მხრიდან, მითითებულ პორტზე*/
  server.begin();
  Serial.println("Server started");
  // IP მისამართის გამოტანა მიმდევრობით მონიტორში
  Serial.print("Use this URL to connect: ");
  Serial.print("http://");
  //NodeMCU-ს ლოკალური IP მისამართის დაბეჭდვა
  Serial.print(WiFi.localIP());
  Serial.println("/");
}
void loop() {
  /*კლიენტის მხრიდან კავშირის შემოწმება. server.available() ფუნქცია უზრუნველყოფს კლიენტთან სერვერის კავშირს, მასში მოთავსებულია კლიენტისგან გამოგზავნილი მონაცემები*/
  WiFiClient client = server.available();
  //თუ არ არის კავშირი კლიენტთან, მაშინ მოხდება ციკლიდან გასვლა
  if (!client) {
    return;
  }
  //ველოდებით, სანამ კლიენტი გამოაგზავნის მონაცემებს
  Serial.println("new client");
  /*client.available ფუნქცია აბრუნებს ინფორმაციას იმის შესახებ, გააგზავნა თუ არა კლიენტმა სერვერზე რაიმე სახის მონაცემი*/
  while (!client.available()) {
```

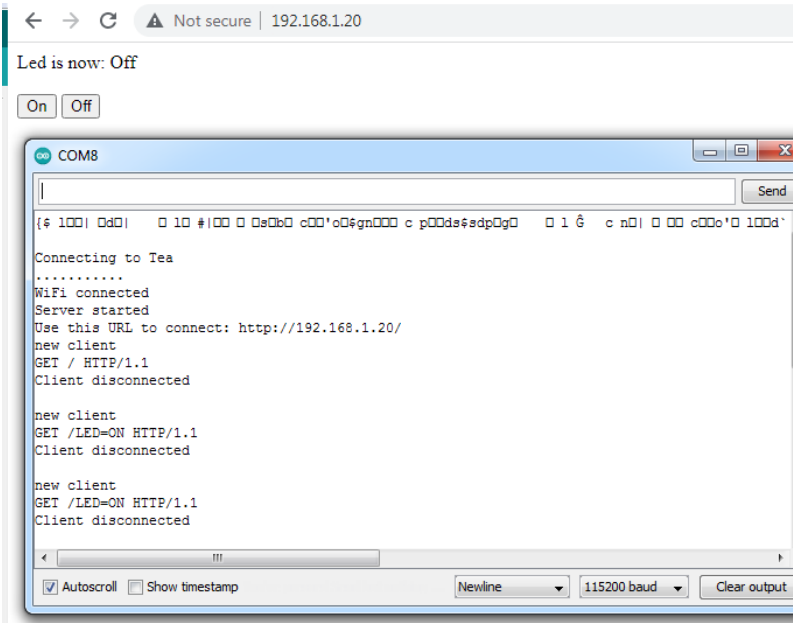
```
    /*თუ არაფერი არ არის კლიენტისგან, მაშინ დავე-
ლოდოთ 1 მილიწამი და შევამოწმოთ ისევ*/
    delay(1);
  }
    //მოთხოვნის პირველი სტრიქონის წაკითხვა.
readStringUntil() კითხულობს სიმბოლოებს სტრიქო-
ნიდან, მანამ სანამ არ გამოჩნდება '\r' სიმბოლო
სტრიქონში
  String request = client.readStringUntil('\r');
  //წაკითხული მნიშვნელობის დაბეჭდვა
  Serial.println(request);
   /*კლიენტის მიერ დაწერილი, მაგრამ ჯერ კიდევ არ
წაკითხული მონაცემების უგულებელყოფა*/
  client.flush();
   /*ცვლადი სადაც შეინახება შუქდიოდის მდგომარეობა.
თავიდანვე ვანიჭებთ LOW მნიშვნელობას, ანუ შუქდიოდი
გამორთულია*/
  int value = LOW;
  /* request.indexOf("/LED=ON") ეძებს 'request' სტრი-
ქონში, ამ სტრიქონის პირველი სიმბოლოდან დაწყებული
'/LED=ON' სტრიქონს და დამთხვევის შემთხვევაში,
ასრულებს შესაბამის მოქმედებას */
  if (request.indexOf("/LED=ON") != -1) {
    // შუქდიოდის ჩართვა
    digitalWrite(ledPin, HIGH);
    //შუქდიოდის მნიშვნელობის შენახვა
    value = HIGH;
  }
  /* request.indexOf("/LED=OFF") ეძებს 'request' სტრი-
ქონში, ამ სტრიქონის პირველი სიმბოლოდან დაწყებული
'/LED=OFF' სტრიქონს და დამთხვევის შემთხვევაში
ასრულებს შესაბამის მოქმედებას */
  if (request.indexOf("/LED=OFF") != -1) {
```

```
    // შუქდიოდის გამორთვა
    digitalWrite(ledPin, LOW);
    //შუქდიოდის მნიშვნელობის შენახვა
    value = LOW;
  }
  /*Html გვერდის შექმნა
HTTP სათაურები ყოველთვის იწყება პასუხის კოდით (მაგ.HTTP/1.1 200 OK) და კონტენტის ტიპის მითითებით, შემდეგ კი აუცილებლად გვაქვს ცარიელი სტრიქონი. HTTP/1.1 200 OK ტექსტით ბრაუზერს ვატყობინებთ, რომ მოთხოვნა წარმატებული იყო*/
  client.println("HTTP/1.1 200 OK");
  //კონტენტის ტიპის მითითება
  client.println("Content-Type: text/html");
  // ცარიელი სტრიქონი
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");
  client.print("Led is now: ");
  if (value == HIGH) {
    client.print("On");
  } else {
    client.print("Off");
  }
  client.println("<br><br>");
  client.println("<a href =\"/LED=ON\"\"> <button>On
</button></a>");
  client.println("<a href=\"/LED=OFF\"\"> <button>Off
</button></a><br />");
  client.println("</html>");
  delay(1);
     client.stop();
  Serial.println("Client disconnected");
```

```
  Serial.println("");
}
```

ქვემოთ, სურ.45-ზე მოცემულია პროგრამის გაშვების შედეგები მიმდევრობით მონიტორსა და ბრაუზერში. გამოვიყენოთ Reset ღილაკი იმ შემთხვევაში, თუ მიმდევრობითი მონიტორის ფანჯარაში მონაცემები არ გამოჩნდება.

ბრაუზერში უნდა ავკრიფოთ NodeMCU-ს IP მისამართი. ON ღილაკზე დაჭერის შედეგად აინთება შუქდიოდი, OFF ღილაკზე დაჭერისას კი შუქდიოდი გამოირთვება.



სურ. 45. პროგრამის გაშვების შედეგები მიმდევრობით მონიტორსა და ბრაუზერში

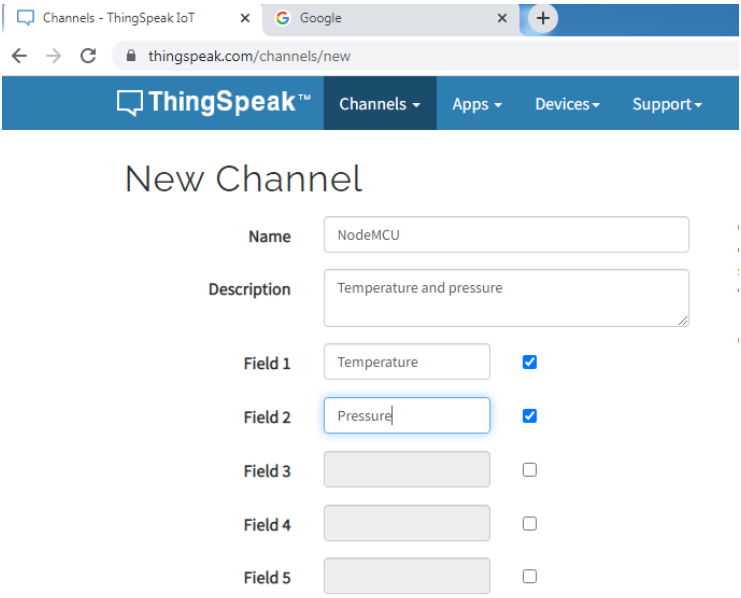
სენსორიდან მიღებული მონაცემების ატვირთვაThingSpeak ღრუბლოვან პლატფორმაზე

ჩვენ უკვე განვიხილეთ, როგორ შეიძლება მივიღოთ მონაცემები ტემპერატურისა და ატმოსფერული წნევის ციფრული BMP180 სენსორიდან, შემდეგ კი ეს მონაცემები გამოვიტანეთ მიმდევრობით მონიტორში, თუმცა არაფერი გვითქვამს ამ მონაცემების შენახვისა და ანალიზის შესაძლებლობაზე (იხილეთ პარაგრაფი „მონაცემების წაკითხვა ციფრული სენსორიდან“).

ThingSpeak არის ღრუბლოვანი პლატფორმა, სადაც ჩვენ შეგვიძლია გავაგზავნოთ და შევინახოთ სენსორებიდან მიღებული მონაცემები.

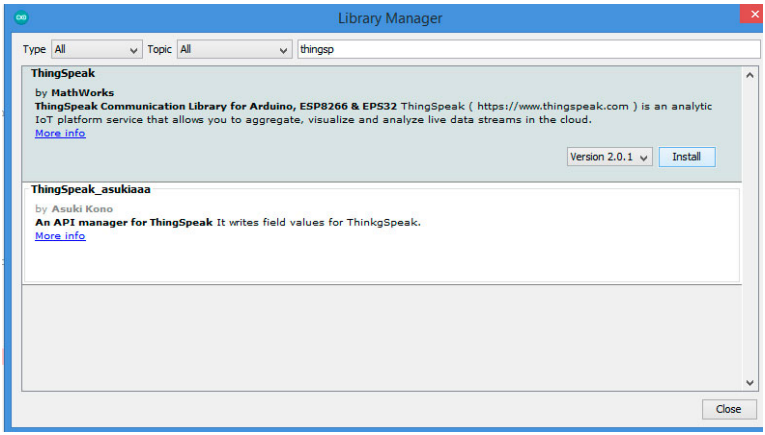
შევქმნათ ანგარიში https://thingspeak.com-ზე. ანგარიშის შექმნის შემდეგ დაგვჭირდება ჩვენი არხის (channel) შექმნა. სწორედ ThingSpeak-ის არხები არის ის ადგილი, სადაც მონაცემები ინახება.

ThingSpeak არხი იქმნება New Channel ბრძანებით. დავარქვათ ჩვენს არხს „NodeMCU”, Field1-ში ჩავწეროთ „Temperature”, Field2-ში - „Pressure”. არხის შექმნის პროცესის დასასრულებლად დავაჭიროთ Save Channel ღილაკს. ThingSpeak ღრუბლოვან პლატფორმაზე არხის შექმნის პროცესი ნაჩვენებია სურ. 46-ზე.



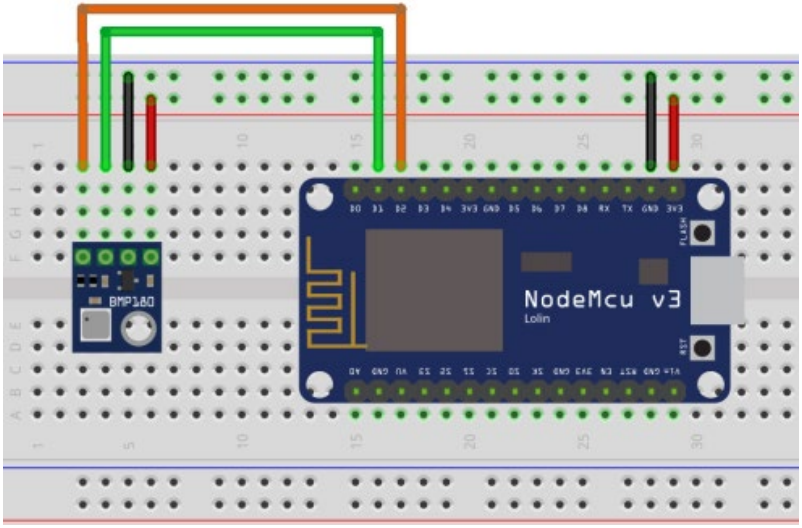
სურ. 46. არხის შექმნა

Arduino IDE-ში Library Manager-დან დავამატოთ და დავაინსტალიროთ ThingSpeak ბიბლიოთეკა.



სურ. 47. ThingSpeak ბიბლიოთეკის დაინსტალირება

ავაგოთ ქვემოთ მოცემული სურათის შესაბამისი სქემა (სურ. 48) და ავტვირთოთ კოდი.



სურ. 48. ტემპერატურისა და წნევის მნიშვნელობების წაკითხვა BMP180 სენსორიდან

ქვემოთ მოცემულია პროგრამის კოდი, რომელიც უზრუნველყოფს ტემპერატურისა და წნევის მნიშვნელობების მიღებას შესაბამისი სენსორიდან, ატვირთავს მონაცემებს ThingSpeak-ზე, კოდში მითითებული დაყოვნების დროის შემდეგ ისევ ატვირთავს ახალ მონაცემებს, ეს პროცესი უწყვეტად მეორდება.

კოდში არ უნდა დაგვავიწყდეს ჩვენი ქსელის სახელის (“your_wifi_ssid”), პაროლის (“your_wifi_password”) და ThingSpeak-ის პარამეტრების (Write API Key, ChannelID) მითითება.

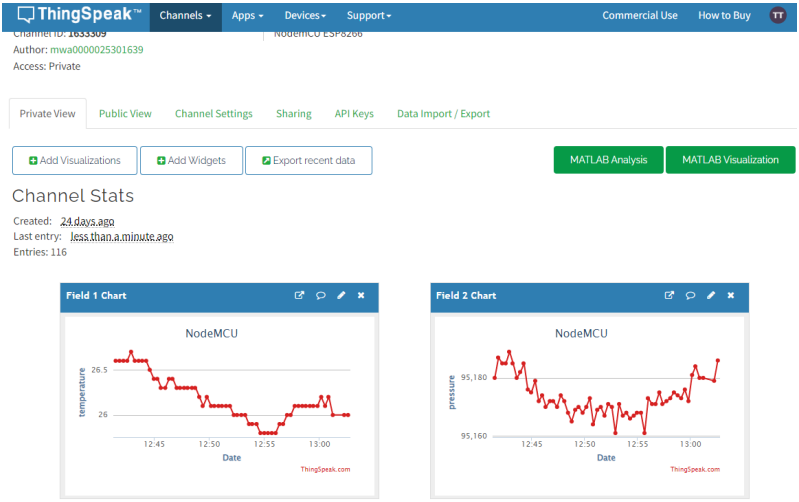
```
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან
დაკავშირებისთვის
#include <ESP8266WiFi.h>
//ThingSpeak-ის ბიბლიოთეკა Arduino, ESP8266 და
ESP32-თვის.
#include <ThingSpeak.h>
//Adafruit BMP085/BMP180-ის ბიბლიოთეკა
#include <Adafruit_BMP085.h>
//BMP180 სენსორის ინიციალიზაცია
Adafruit_BMP085 bmp;
//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
//თქვენი ChannelID
unsigned long myChannelID = 34568348;
// თქვენი ThingsPeak API Key
#define myWriteAPIKey "4354DNT74O098630"
//კლიენტის შექმნა, რომელსაც შეუძლია დაუკავშირდეს
მითითებულ IP მისამართს
WiFiClient client;

void setup() {
  Serial.begin(115200);
  //ThingSpeak ბიბლიოთეკის ინიციალიზაცია
  ThingSpeak.begin(client);
}

void loop()
{
  /*  Wi_Fi ქსელთან დაკავშირება. თუ status-ის მნიშვ-
ნელობა არ არის WL_CONNECTED-ის ტოლი, მაშინ დაელოდე
500 მილიწამი და შეამოწმე სტატუსი ხელახლა.
```

```
WL_CONNECTED სტატუსი ენიჭება მაშინ, როდესაც Wi-Fi
ქსელთან კავშირი გვექნება*/
  if (WiFi.status() != WL_CONNECTED) {
    //WiFi ბიბლიოთეკის ქსელური პარამეტრების
ინიციალიზაცია
    WiFi.begin(ssid, wifiPassword);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }
    Serial.println("\nConnected.");
  }
  /*თუ bmp.begin ფუნქციის მნიშვნელობა არის false,
ეს ნიშნავს რომ პროგრამამ ვერ დაინახა სენსორი. ეკრანზე
გამოდის შესაბამისი შეტყობინება*/
  if (!bmp.begin()) {
    Serial.println("Could not find BMP180 or BMP085
sensor at 0x77");
    while (1) {}
  }
  //ტემპერატურის მნიშვნელობის წაკითხვა სენსორიდან
  float temperature = bmp.readTemperature();
  //წნევის მნიშვნელობის წაკითხვა სენსორიდან
  float pressure = bmp.readPressure();
  // არხის ველებისთვის  შესაბამისი მნიშვნელობების
მინიჭება
  ThingSpeak.setField(1, temperature);
  // ტემპერატურის მნიშვნელობის მინიჭებაThingSpeak-ის
პირველი არხისთვის
  ThingSpeak.setField(2, pressure);
  // წნევის მნიშვნელობის მინიჭება ThingSpeak-ის მეორე
არხისთვის
```

```
  //ThingSpeak არხის ველებისთვის მინიჭებული
მნიშვნელობების ჩაწერა
  int httpCode = ThingSpeak.writeFields(myChannelID,
myWriteAPIKey);

  //შევამოწმოთ ThingSpeak-დან დაბრუნებული კოდის
მნიშვნელობა, თუ ეს კოდი 200 - ის ტოლია, ეს ნიშნავს,
რომ ჩაწერის პროცესი წარმატებით დასრულდა
  if (httpCode == 200) {
    /* დაბრუნებული კოდების შესაძლო მნიშვნელობების
ნუსხა
      Value  Meaning
      200 OK / Success
      404 Incorrect API key (or invalid ThingSpeak
server address)
      -101  Value is out of range or string is too
long (> 255 characters)
      -201  Invalid field number specified
      -210  setField() was not called before
writeFields()
      -301  Failed to connect to ThingSpeak
      -302  Unexpected failure during write to
ThingSpeak
      -303  Unable to parse response
      -304  Timeout waiting for server to respond
      -401  Point was not inserted (most probable
cause is the rate limit of once every 15 seconds)
      0 Other error  */

    Serial.println("Channel write successful.");
  }
  else {
```

```
    //პრობლემის შემთხვევაში, გამოიტანს მიღებული
კოდის შესაბამის მნიშვნელობას
    Serial.println("Problem writing to channel.HTTP
error code " + String(httpCode));
  }
  // არხის მონაცემების განახლება ყოველ 20 წამში
  delay(20000);
}
```

შევიდეთ ThingSpeak.com-ზე და ვნახოთ ჩვენს მიერ შექმნილი არხი.

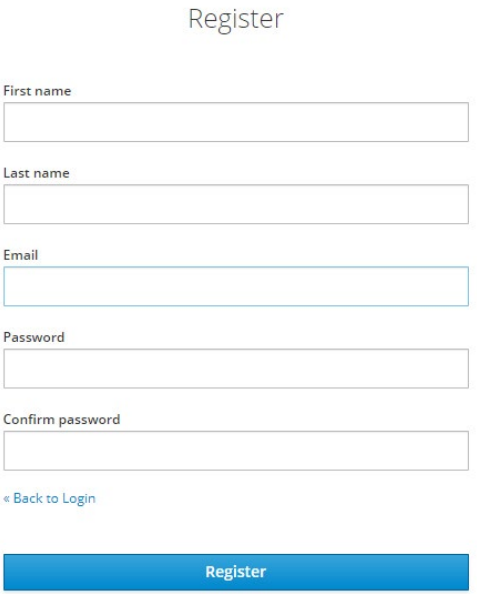


სურ. 49. BMP180-დან მიღებული ტემპერატურისა და წნევის მნიშვნელობები ThingSpeak-ზე

შუქდიოდის მდგომარეობის მართვა Cayenne ღრუბლოვანი პლატფორმის გამოყენებით

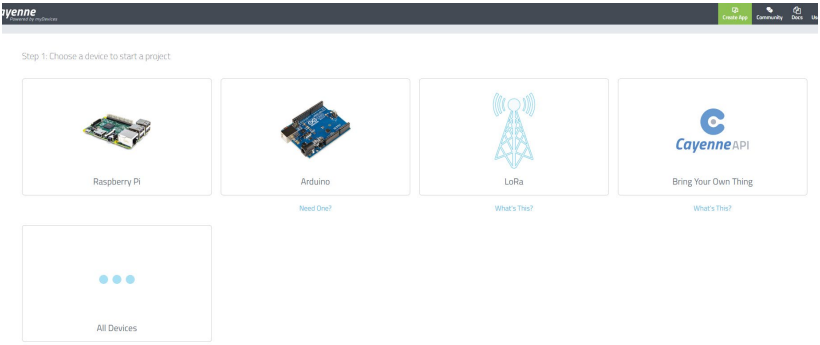
მოცემულ პარაგრაფში NodeMCU დაფის გამოყენებით ვიმუშავებთ Cayenne ღრუბლოვან პლატფორმასთან. Cayenne-ის საშუალებით, შეგვიძლია დისტანციურად ვმართოთ ჩვენი IoT პროექტები, დავამუშაოთ და შევინახოთ სხვადასხვა მოწყობილობიდან მიღებული მონაცემები.

შევქმნათ ჩვენი ანგარიში Cayenne პლატფორმაზე. https://developers.mydevices.com/cayenne/features/ ვებ გვერდის მენიუდან ავირჩიოთ Sign up Free.



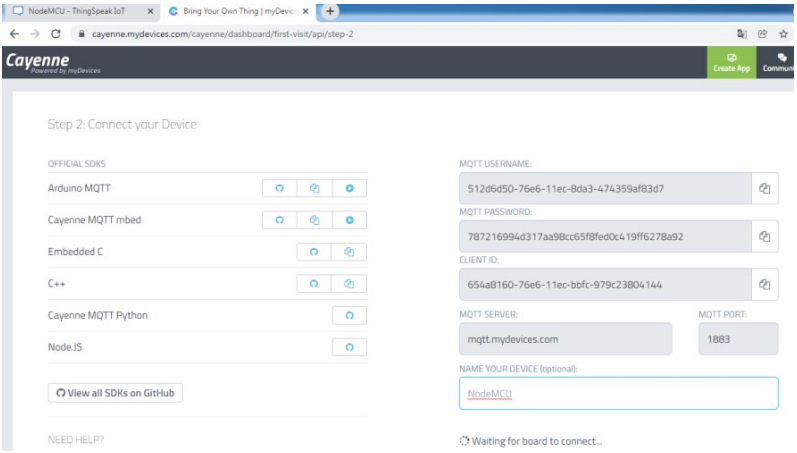
სურ. 50. ანგარიშის შექმნა Cayenne-ზე

ანგარიშის შექმნის შემდეგ გამოჩნდება ფანჯარა:



სურ. 51. მოწყობილობის არჩევა

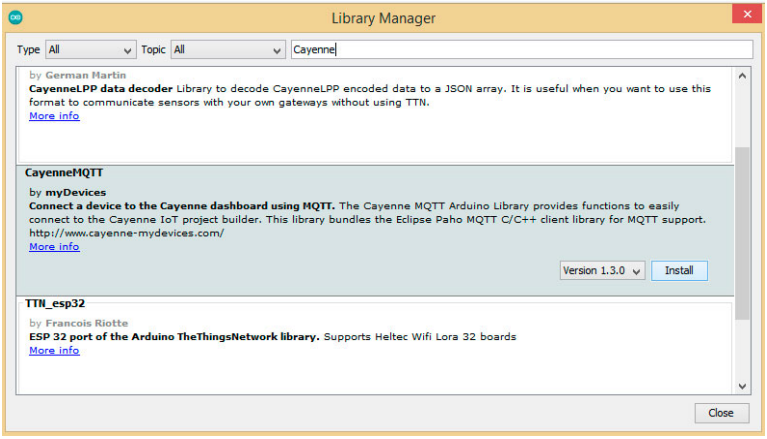
Bring Your own thing-ზე დაჭერით მივიღებთ MQTT მომხმარებლის სახელს, პაროლს და CLIENT ID-ს. NAME YOUR DEVICE ველში ვწერთ ჩვენს მიერ გამოყენებული მოწყობილობის სახელს (NodeMCU).



სურ. 52. Cayenne პლატფორმასთან დაკავშირებისთვის საჭირო პარამეტრები

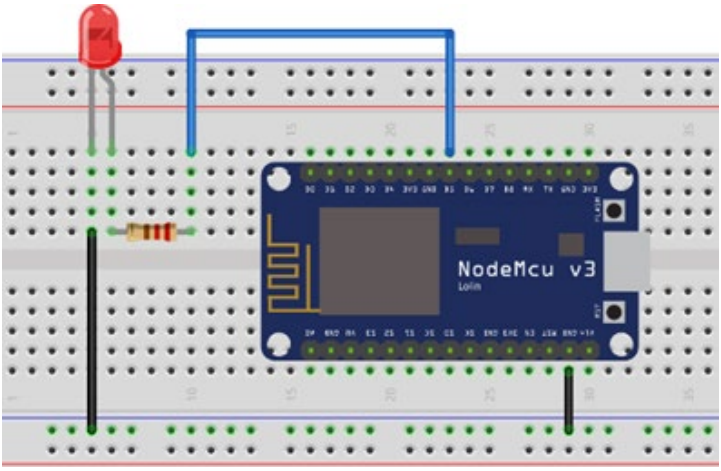
ამ ფანჯარას (სურ. 52) არ ვხურავთ. ის ელოდება ჩვენს მოწყობილობასთან (NodeMCU) დაკავშირებას. ფანჯარაში ჩანს ტექსტი „Waiting for board to connect".

გავხსნათ Arduino IDE და დავაინსტალიროთ Cayenne MQTT ბიბლიოთეკა. MQTT წარმოადგენს სპეციალიზებულ პროტოკოლს, რომელიც IoT სისტემაში მონაცემების გადასაცემად გამოიყენება. MQTT პროტოკოლსა და მასთან დაკავშირებულ საკითხებს დეტალურად ქვემოთ, შემდეგ პარაგრაფებში განვიხილავთ.



სურ. 53. Cayenne MQTT ბიბლიოთეკის დაინსტალირება

ავაგოთ შესაბამისი სქემა (სურ. 54) და ავტვირთოთ კოდი NodeMCU-ზე. ამ კოდში ვწერთ Cayenne-ის ფანჯარაში (სურ.52) მოცემულ მნიშვნელობებს (MQTT USERNAME, MQTT PASSWORD, CLIENT ID).



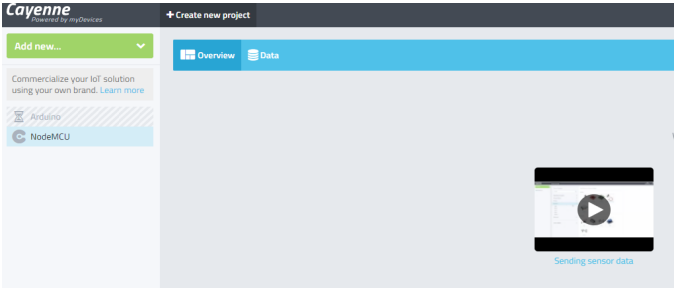
სურ. 54. შუქდიოდის მდგომარეობის მართვა Cayenne-ის გამოყენებით

```
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან დაკავშირებისთვის
#include <ESP8266WiFi.h>
//Cayenne MQTT ბიბლიოთეკა
#include <CayenneMQTTESP8266.h>
//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
//Cayenne პლატფორმასთან დაკავშირებისთვის საჭირო პარამეტრები
char mqtt_username[] = "512d6d50-76e69af83d7";
char mqtt_password[] = "787216994f8fed0c419f";
char client_id[] = "161ee080-7c46-11ec-8c443";
//GPIO14, D5 გამომყვანი NodeMCU დაფაზე
int ledPin = D5;
void setup() {
```

```
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  // WiFi ქსელთან დაკავშირება
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //WiFi ბიბლიოთეკის ქსელური პარამეტრების
ინიციალიზაცია
  WiFi.begin(ssid, wifiPassword);
  /* Wi_Fi ქსელთან დაკავშირება. თუ status-ის
მნიშვნელობა არ არის WL_CONNECTED-ის ტოლი, მაშინ
დაელოდე 500 მილიწამი და შეამოწმე სტატუსი ხელახლა.
WL_CONNECTED სტატუსი ენიჭება მაშინ, როდესაც Wi-Fi
ქსელთან კავშირი გვექნება*/
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  //NodeMCU-სა და Cayenne-ს შორის კავშირის დაწყება
  Cayenne.begin(mqtt_username, mqtt_password,
                client_id, ssid, wifiPassword);
}
void loop() {
  Cayenne.loop();
}
//მონაცემების წაკითხვა Cayenne-ის არხიდან, რომლის
ნომერია 0
CAYENNE_IN(0) {
```

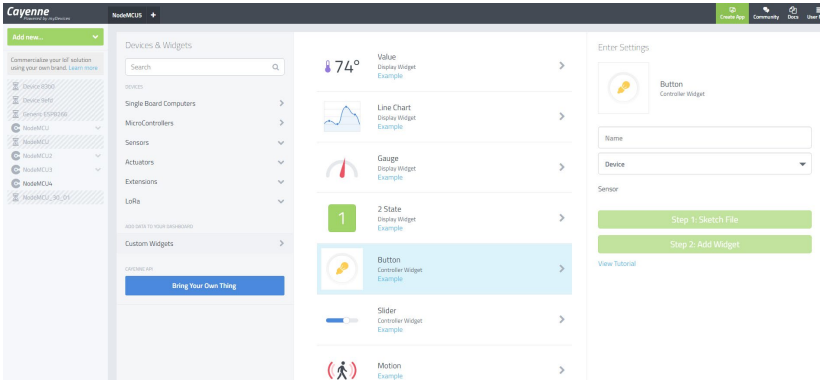
```
  //გამომყვანის სტატუსის შეცვლა წაკითხული მონაცემის მიხედვით. AsInt() ნიშნავს, რომ მონაცემს ვკითხულობთ, როგორც მთელი ტიპისას
  digitalWrite(ledPin, getValue.asInt());
}
```

კოდის ატვირთვის შემდეგ გამოჩნდება ფანჯარა:



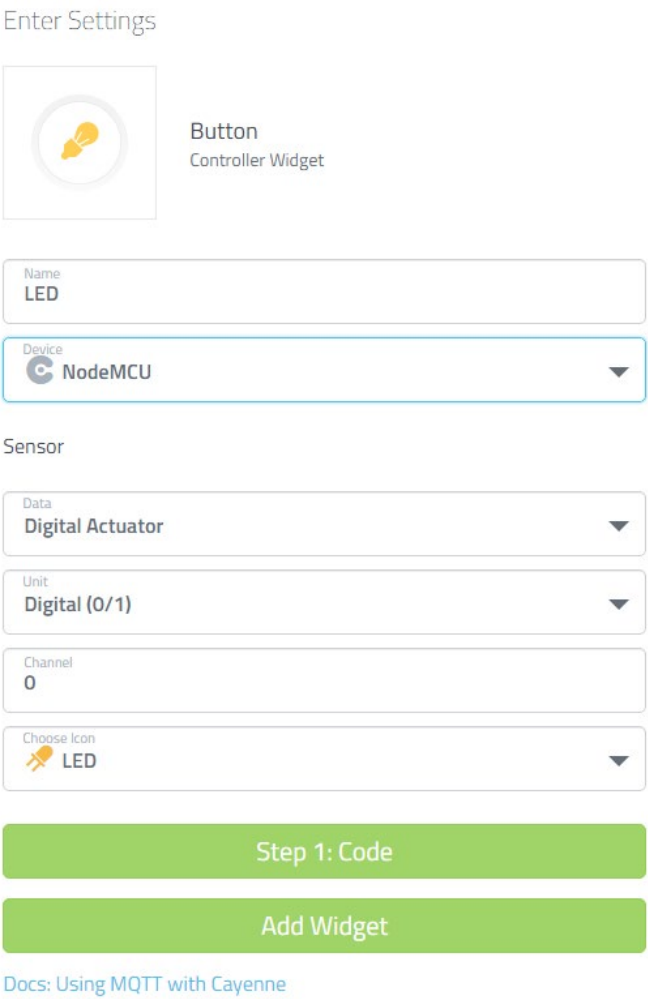
სურ. 55. კოდის ატვირთვის შემდეგ მიღებული ფანჯარა Cayenne პლატფორმაზე

დავაჭიროთ Add new->Device/Widget ღილაკს, შემდეგ Custom Widgets განყოფილებიდან შევარჩიოთ Button, როგორც ეს სურ. 56-ზეა მოცემული.



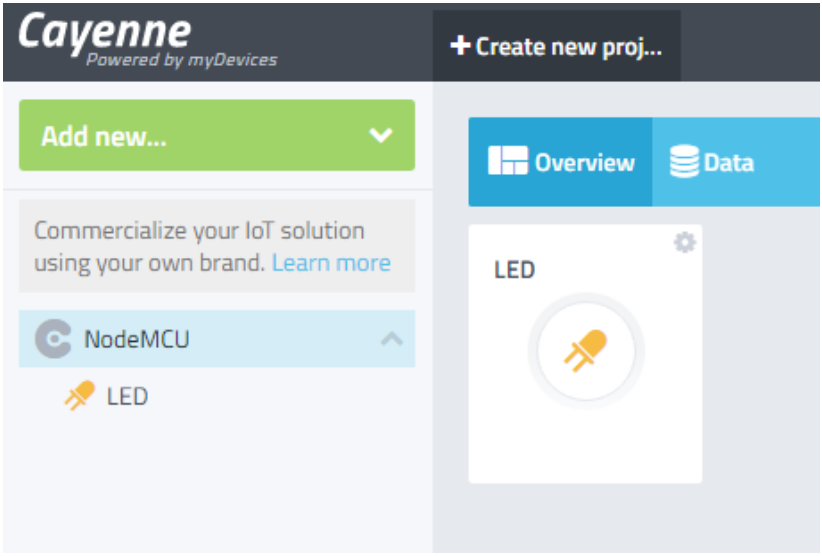
სურ. 56. შესაბამისი ვიჯეტის არჩევა

Button-ის შერჩევის შემდეგ, მოვახდინოთ მისი კონფიგურირება:



სურ. 57. Button-ის კონფიგურირება

Button-ის კონფიგურირების შემდეგ ვაჭერთ “Add Widget” ღილაკს, შეიქმნება შუქდიოდის მდგომარეობის სამართავი ღილაკი (სურ. 58).



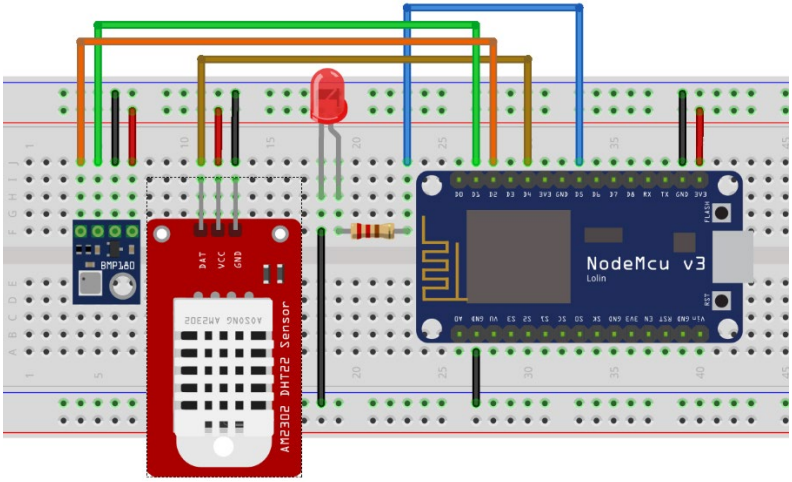
სურ. 58. Cayenne პლატფორმაზე შუქდიოდის მდგომარეობის სამართავი ღილაკი

LED ღილაკზე დაჭერით ვცვლით სქემაზე (სურ. 54) დაერთებული შუქდიოდის მდგომარეობას.

შუქდიოდის მართვა და სენსორებიდან მიღებული მონაცემების ატვირთვა Cayenne ღრუბლოვან პლატფორმაზე

მას შემდეგ, რაც ვისწავლეთ Cayenne ღრუბლოვან პლატფორმასთან მუშაობა, შეგვიძლია ჩვენი ამოცანა კიდევ უფრო კომპლექსური გავხადოთ: შუქდიოდის მდგომარეობის მართვას დავამატოთ სენსორებიდან (BMP180 და DHT22) მიღებული მონაცემების ატვირთვა Cayenne-ზე; BMP180-დან ვიღებთ წნევის, DHT22-დან - ტემპერატურისა და ტენიანობის მნიშვნელობებს.

ავაგოთ შესაბამისი სქემა (სურ. 59) და ავტვირთოთ კოდი.



სურ. 59. შუქდიოდის მდგომარეობის მართვა და ტემპერატურისა და წნევის მნიშვნელობების მიღება სენსორებიდან

```
//Cayenne MQTT ბიბლიოთეკა
#include <CayenneMQTTESP8266.h>
//Adafruit BMP085/BMP180-ის ბიბლიოთეკა
#include <Adafruit_BMP085.h>
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან
დაკავშირებისთვის
#include <ESP8266WiFi.h>
//DHT სენსორის ბიბლიოთეკა. არდუინოს ბიბლიო-
თეკა  DHT11, DHT22 და სხვა ტემპერატურისა და ტენიანობის
სენსორებისთვის
#include <DHT.h>
//გამოიყენება DHT22 (AM2302) და AM2321 სენსორებისთვის
#define DHTTYPE DHT22
//DHT სენსორთან დაკავშირებული ციფრული გამომყვანი,
D4 გამომყვანი NodeMCU დაფაზე
#define DHTPIN D4
//DHT სენსორის ინიციალიზაცია
DHT dht(DHTPIN, DHTTYPE);
//BMP180 სენსორის ინიციალიზაცია
Adafruit_BMP085 bmp;
//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
//Cayenne პლატფორმასთან დაკავშირებისთვის საჭირო
პარამეტრები
char mqtt_username[] = "512d6d50-76e69af83d7";
char mqtt_password[] = "787216994f8fed0c419f";
char client_id[] = "161ee080-7c46-11ec-8c443";
//float temperature_bmp;
float temperature_dht;
float hum_dht;
float pressure_bmp;
```

```
//GPIO14, D5 გამომყვანი NodeMCU დაფაზე
int ledPin = D5;
void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  // WiFi ქსელთან დაკავშირება
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  //WiFi ბიბლიოთეკის ქსელური პარამეტრების
ინიციალიზაცია
  WiFi.begin(ssid, wifiPassword);
  /* Wi_Fi ქსელთან დაკავშირება. თუ status-ის
მნიშვნელობა არ არის WL_CONNECTED-ის ტოლი, მაშინ
დაელოდე 500 მილიწამი და შეამოწმე სტატუსი ხელახლა.
WL_CONNECTED სტატუსი ენიჭება მაშინ, როდესაც Wi-Fi
ქსელთან კავშირი გვექნება*/
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  //NodeMCU-სა და Cayenne-ს შორის კავშირის დაწყება
  Cayenne.begin(mqtt_username, mqtt_password,
client_id);
}
void loop() {
  /*თუ bmp.begin ფუნქციის მნიშვნელობა არის false,
ეს ნიშნავს რომ პროგრამამ ვერ დაინახა სენსორი. ეკრანზე
გამოდის შესაბამისი შეტყობინება*/
```

```
  if (!bmp.begin()) {
    Serial.println("Could not find BMP180 or BMP085
sensor at 0x77");
    while (1) {}
  }
  //temperature_bmp = bmp.readTemperature();
  //წნევის მნიშვნელობის წაკითხვა და შენახვა
  pressure_bmp = bmp.readPressure();
  // DHT სენსორის ინიციალიზაცია
  dht.begin();
  //ტენიანობის და ტემპერატურის მნიშვნელობების
წაკითხვა და შენახვა
  hum_dht = dht.readHumidity();
  temperature_dht = dht.readTemperature();
  Serial.println(temperature_dht);
  //Serial.println (temperature_bmp);
  Serial.println(pressure_bmp);
  Serial.println(hum_dht);
  //დაყოვნება სენსორიდან მიღებული მნიშვნელობების
წაკითხვისას, ამის მითითება აუცილებელია BMP180
სენსორისთვის, რადგანაც წაკითხული მნიშვნელობების
სიზუსტე დამოკიდებულია სენსორიდან მონაცემების
წაკითხვის სიხშირეზე
  delay(500);
  Cayenne.loop();
}
//მონაცემების წაკითხვა Cayenne-ის არხიდან, რომლის
ნომერია 0
CAYENNE_IN(0) {
  //გამომყვანის სტატუსის შეცვლა წაკითხული მონაცემის
მიხედვით. AsInt() ნიშნავს, რომ მონაცემს ვკითხულობთ,
როგორც მთელი ტიპისას
  digitalWrite(ledPin, getValue.asInt());
```

```
}
//მონაცემების გაგზავნა Cayenne-ზე (არხის ნომერი,
მონაცემი, მონაცემის ტიპი, ერთეული)
CAYENNE_OUT(1) {
  Cayenne.virtualWrite(1, temperature_dht,
TYPE_TEMPERATURE, UNIT_CELSIUS);
}
CAYENNE_OUT(2) {
  Cayenne.virtualWrite(2, pressure_bmp,
TYPE_BAROMETRIC_PRESSURE, UNIT_PASCAL);
}
CAYENNE_OUT(3) {
  Cayenne.virtualWrite(3, hum_dht,
TYPE_RELATIVE_HUMIDITY, UNIT_PERCENT);
}
```

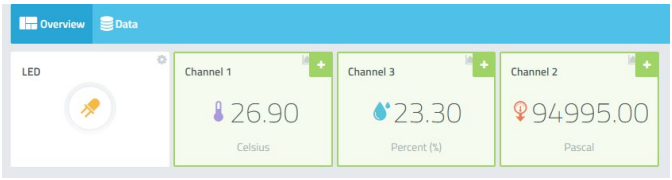
DHT22 სენსორისთვის ჩვენ დაგვჭირდება DHT.h-ის ბიბლიოთეკა. Arduino IDE-ში, Sketch->Include Library> Manage Libraries მენიუს გამოყენებით, ბიბლიოთეკების ნუსხაში უნდა მოვძებნოთ DHT sensor library ბიბლიოთეკა და შემდეგ დავაინსტალიროთ.

მონაცემების და მათი შესაბამისი ერთეულებისთვის ტიპების აღწერა მოცემულია Cayenne MQTT Client ბიბლიოთეკაში (https://github.com/myDevicesIoT/Cayenne-MQTT-ESP/blob/master/src/CayenneUtils/CayenneTypes.h#L21):

```
#define TYPE_BAROMETRIC_PRESSURE "bp" // Barometric
pressure
#define TYPE_BATTERY "batt" // Battery
```

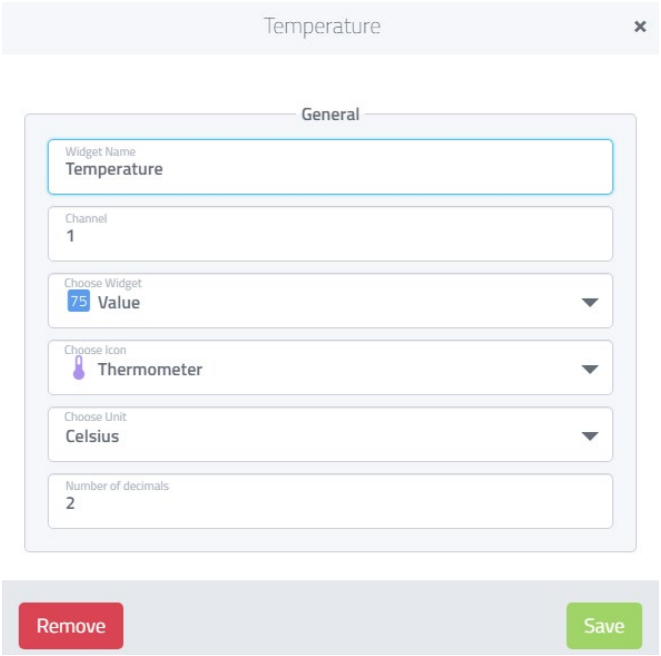
```
#define TYPE_LUMINOSITY "lum" // Luminosity
#define TYPE_PROXIMITY "prox" // Proximity
#define TYPE_RELATIVE_HUMIDITY "rel_hum" // Relative
Humidity
#define TYPE_TEMPERATURE "temp" // Temperature
#define TYPE_VOLTAGE "voltage" // Voltage
#define TYPE_DIGITAL_SENSOR "digital_sensor" // Voltage
// Unit types
#define UNIT_UNDEFINED "null"
#define UNIT_PASCAL "pa" // Pascal
#define UNIT_HECTOPASCAL "hpa" // Hectopascal
#define UNIT_PERCENT "p" // % (0 to 100)
#define UNIT_RATIO "r" // Ratio
#define UNIT_VOLTS "v" // Volts
#define UNIT_LUX "lux" // Lux
#define UNIT_MILLIMETER "mm" // Millimeter
#define UNIT_CENTIMETER "cm" // Centimeter
#define UNIT_METER "m" // Meter
#define UNIT_DIGITAL "d" // Digital (0/1)
#define UNIT_FAHRENHEIT "f" // Fahrenheit
#define UNIT_CELSIUS "c" // Celsius
#define UNIT_KELVIN "k" // Kelvin
#define UNIT_MILLIVOLTS "mv" // Millivolts
```

კოდის ატვირთვის შემდეგ Cayenne პლატფორმაზე მივიღებთ:



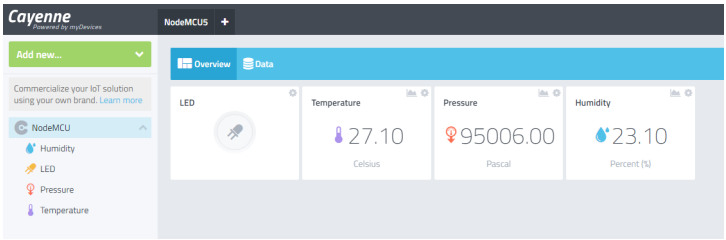
სურ. 60. შუქდიოდის მდგომარეობა, ტემპერატურის, ტენიანობის და წნევის მნიშვნელობები Cayenne-ზე

თითოეული არხის ზედა მარჯვენა კუთხეში „+“ ნიშანზე დაჭერით სრულდება „Add to Dashboard” ბრძანება. თითოეული არხისთვის „Settings” დიალოგური ფანჯრიდან შეგვიძლია ჩვენთვის სასურველი პარამეტრების მითითება.



სურ.61. არხის პარამეტრების რედაქტირება (ტემპერატურის არხის მაგალითზე)

საბოლოოდ, Cayenne-ზე მივიღებთ შემდეგ სურათს:



სურ. 62. შუქდიოდის მდგომარეობა, ტემპერატურის, ტენიანობის და წნევის მნიშვნელობები Cayenne-ზე

Node-RED პლატფორმა

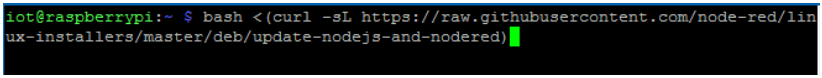
იმისათვის, რომ მარტივად ვმართოთ ჩვენი IoT პროექტები ლოკალური ქსელიდან, საჭიროა გვქონდეს მართვის გრაფიკული ინტერფეისი. ასეთი ინტერფეისის მქონე ერთერთ პლატფორმას წარმოადგენს Node-RED. Node-RED კვანძების შეერთებაზეა დაფუძნებული, შეიცავს სხვადასხვა სახის მარტივსა და რთულ IoT ფუნქციას.

სანამ Node-RED პლატფორმის ინსტალაციას დავიწყებთ, საჭიროა განვაახლოთ Raspberry Pi OS-ის პროგრამული უზრუნველყოფა, ამისათვის ტერმინალის ფანჯარაში ავკრიფოთ ბრძანება *sudo apt update;* ამ ბრძანების შესრულების შემდეგ ავკრიფოთ *sudo apt upgrade*. ამით დასრულდება სისტემის სრული პროგრამული განახლება.

როგორც წესი, Node-RED სისტემაში უკვე დაყენებულია, მაგრამ ჩვენ გირჩევთ, დააინსტალიროთ Node-Red-ის ბოლო

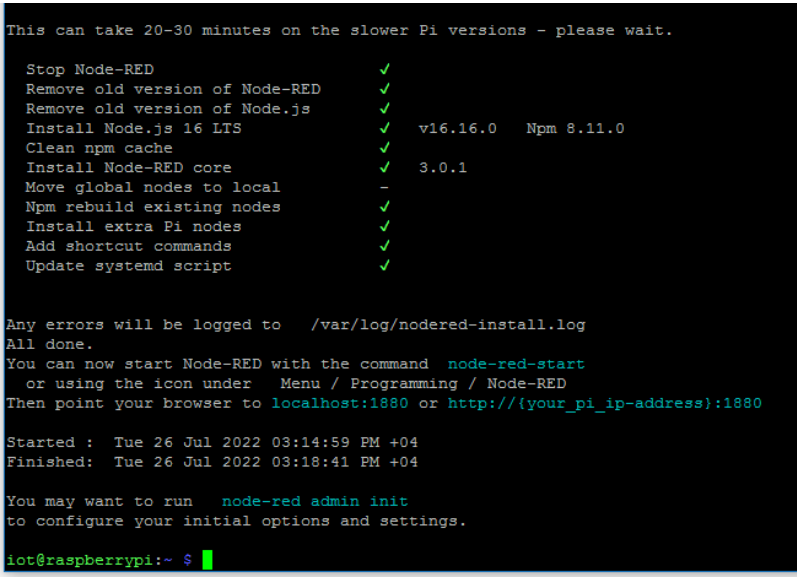
ვერსია Raspberry Pi-ს ტერმინალზე შემდეგი ბრძანების გაშვებით:

bash <(curl -sL *https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered*)



სურ. 63. Node-RED-ის ინსტალაცია

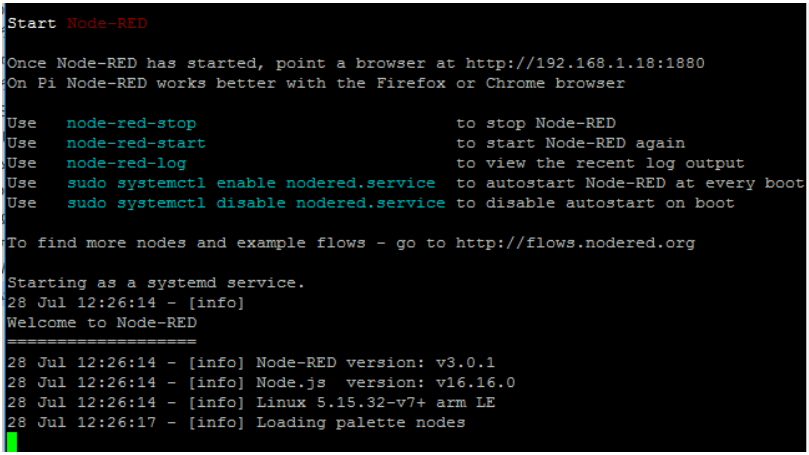
ინსტალაციის დასრულების შემდეგ მივიღებთ:



სურ. 64. Node-RED-ის ინსტალაციის დასრულება

Node-RED-თან მუშაობის დასაწყებად Raspberry Pi-ს სამუშაო მაგიდის ინტერფეისის მენიუდან უნდა მოვნიშნოთ

Programming->Node-RED ან გავხსნათ ტერმინალის ფანჯარა და ავკრიფოთ *node-red start* ბრძანება.



სურ 65. Node-RED-ის გაშვება

დაველოდოთ რამდენიმე წამი, მანამ სანამ Node-RED მუშაობას დაიწყებს. Node-RED-ის გაშვების შემდეგ, პირველივე სტრიქონში, ეკრანზე გამოჩნდება Raspberry pi-ს IP მისამართი და პორტი. ჩვენს შემთხვევაში, Raspberry pi-ს მისამართია 192.168.1.18 (სურ. 65).

Node-RED-ის გაშვების შემდეგ გავხსნათ ვებ ბრაუზერი და სამისამართო ველში ჩავწეროთ Raspberry pi-ს მისამართი: http://192.168.1.18:1880.

ეკრანზე გამოჩნდება Node-RED-ის სამუშაო გარემო (სურ. 66).

სურ. 66. Node-RED-ის სამუშაო გარემო

Node-RED-ის სამუშაო გარემოს დამატებითი კომპონენტების ინსტალაციისთვის, ტერმინალის ფანჯარაში ავკრიფოთ ქვემოთ მოცემული ბრძანებები:

node-red-stop

cd ~/.node-red

npm install node-red-dashboard

node-red-start

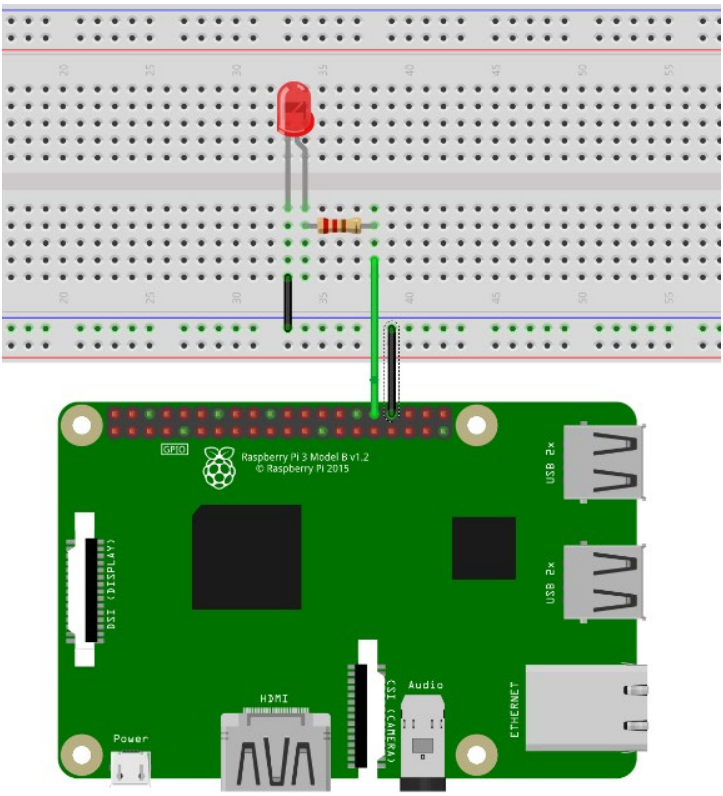
```
iot@raspberrypi:~ $ node-red-stop
Stop Node-RED

Use   node-red-start   to start Node-RED again

iot@raspberrypi:~ $ cd ~/.node-red
iot@raspberrypi:~/.node-red $ npm install node-red-dashboard

up to date, audited 79 packages in 2s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
iot@raspberrypi:~/.node-red $
```

სურ. 67. Node-RED-ის სამუშაო გარემოს დამატებითი კომპონენტების ინსტალაცია

ქვემოთ მოცემულია რამდენიმე ბრძანება, რომელთა ცოდნა სასარგებლო იქნება Node-RED-თან სამუშაოდ: სააღრიცხვო ჩანაწერების (log) სანახავად, გამოიყენება node-red-log ბრძანება; თუ გვინდა, რომ Node-RED ჩაიტვირთოს Raspberry Pi OS-ის ჩატვირთვისას, ტერმინალის ფანჯარაში უნდა ავკრიფოთ *sudo systemctl enable nodered.service* ბრძანება; Node-RED-ის მუშაობის შეწყვეტისთვის გამოიყენება *node-red-stop* ბრძანება. ტერმინალის ფანჯრის დახურვა (ან ctrl+c კლავიშების კომბინაციაზე დაჭერა) არ იწვევს Node-RED-ის გაჩერებას, ასეთ შემთხვევაში, პროგრამა ფონურ რეჟიმში აგრძელებს მუშაობას.

Raspberry Pi-ს GPIO გამომყვანების დისტანციურად მართვა Node-RED-ის გამოყენებით

როგორც ზემოთ აღვნიშნეთ, Raspberry Pi-ს აქვს GPIO გამომყვანები, რომლებთანაც, როგორც წესი, შესაძლებელია შემსრულებელი მექანიზმებისა და სენსორების მიერთება და მათი პროგრამული მართვა. Node-RED საშუალებას იძლევა მარტივად დავუკავშირდეთ Raspberry pi-ს GPIO გამომყვანებს. იმისათვის, რომ ვაჩვენოთ როგორ შეიძლება Node-RED-ის გამოყენებით Raspberry pi-ს GPIO გამომყვანის დისტანციურად მართვა, განვიხილოთ შუქდიოდის ანთება/ჩაქრობის მარტივი ამოცანა. ავაგოთ სურ. 68-ზე მოცემული სქემა.

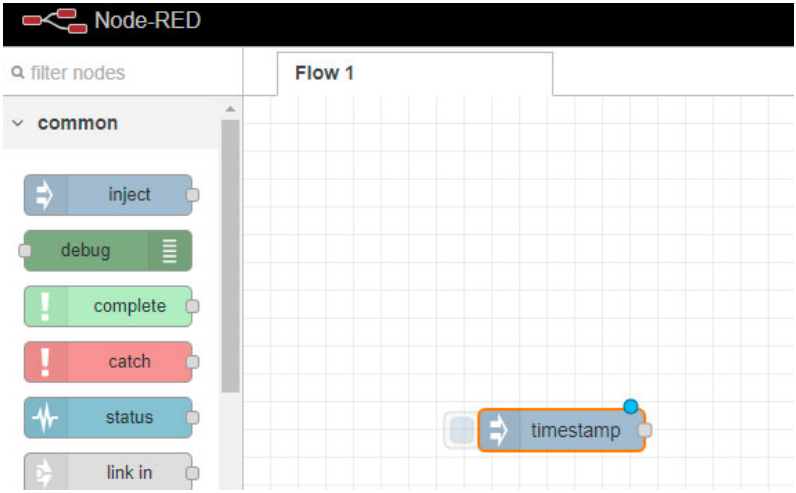


სურ. 68. შუქდიოდის მართვა Node-RED-ის საშუალებით

გავხსნათ ბრაუზერის ფანჯარა და სამისამართო ველში ჩავწეროთ Raspberry pi-ს მისამართი http://192.168.1.18:1880). ეკრანზე გამოჩნდება Node-RED-ის სამუშაო გარემო, რომლის მარცხენა მხარეს მოთავსებულია ჯგუფებად დაყოფილი კვანძები. ეკრანის შუა ნაწილში შეიძლება მოვათავსოთ სხვადასხვა ნაკადი (flow). ნაკადი წარმოადგენს კვანძების

შეერთებას, რომლებიც ფუნქციონირებენ მათთვის მინიჭებული ფუნქციების შესაბამისად. ეკრანის მარჯვენა ნაწილში არსებული ინფორმაციული ჩანართები აჩვენებს დეტალურ ინფორმაციას პროგრამის მუშაობის შესახებ. პროგრამის მუშაობის შედეგების ასახვის მიზნით, წითელი ფერის Deploy ღილაკი ეკრანის ზედა მარჯვენა კუთხეში ახდენს ნაკადის შენახვას.

Node RED-ის სამუშაო გარემოში, Flow1 არეში გადავიტანოთ ორი inject კვანძი (სურ. 69). Inject კვანძზე მაუსის ღილაკის ორჯერ დაჭერით ეკრანზე მივიღებთ inject

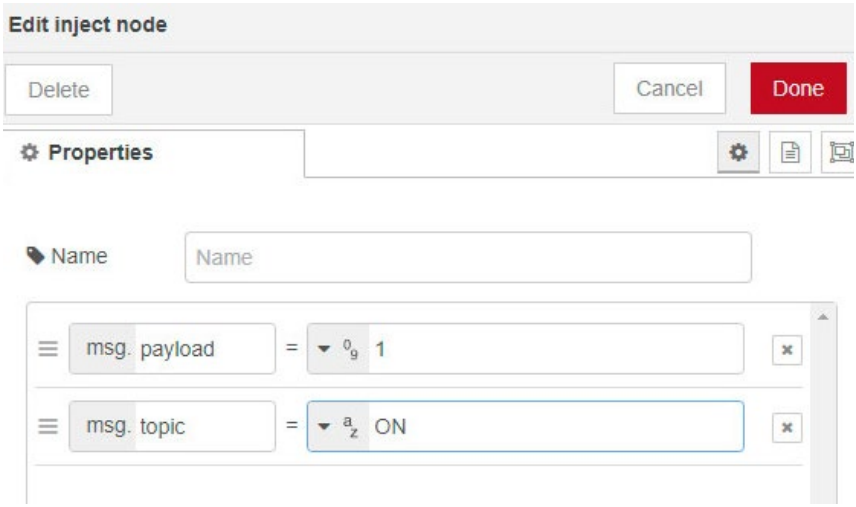


სურ. 69. Node-RED-ის გარემოში მუშაობა

კვანძის რედაქტირების ფანჯარას, რომელშიც მოცემულია inject კვანძის თვისებები (სურ. 70). რედაქტირების ფანჯრის ველები შევსებულია შემდეგი წესით:

msg.payload → number = 1

msg.topic -> string = ON



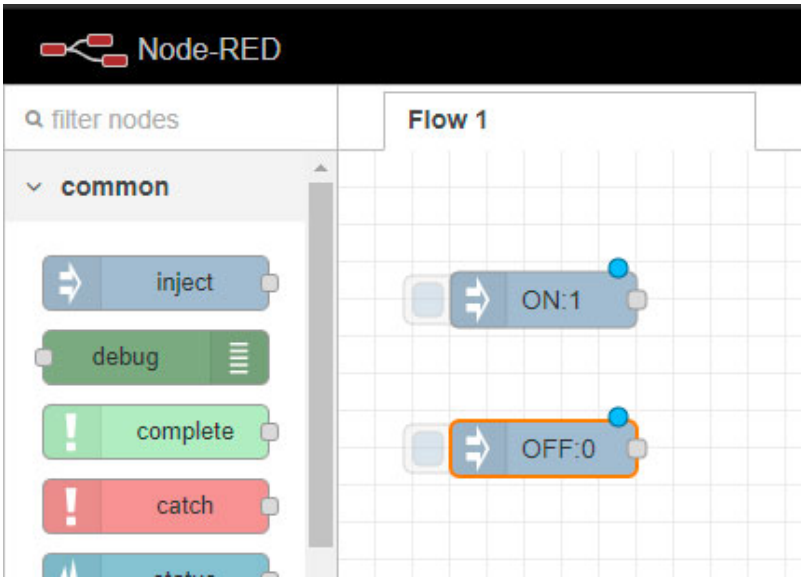
სურ.70. inject კვანძის თვისებები

იგივე პროცედურა გავიმეოროთ მეორე inject ტიპის კვანძისთვის, რომლისთვისაც რედაქტირების ფანჯრის ველები შეივსება ქვემოთ მოცემული გზით:

msg.payload → number = 0

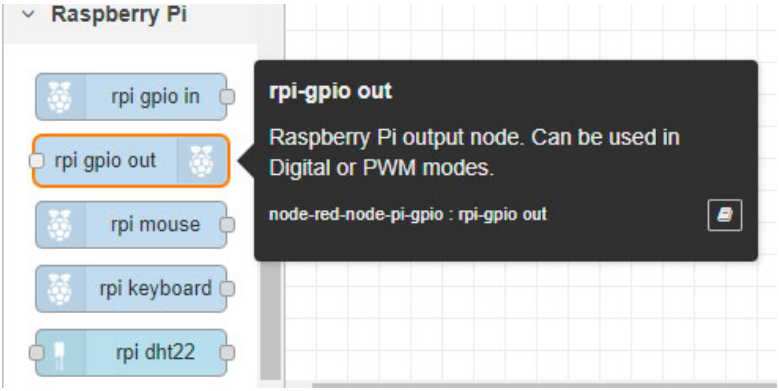
msg.topic -> string = OFF

ორივე inject კვანძის თვისებების რედაქტირების შედეგად მივიღებთ:



სურ. 71. Node-RED-ის გარემოში მუშაობა

Node-RED-ის სამუშაო გარემოში მოვათავსოთ rpi-gpio out კვანძი.



სურ. 72. Node-RED-ის გარემოში მუშაობა

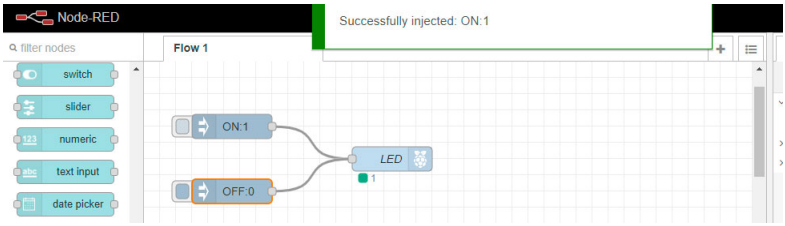
rpi-gpio out კვანძზე მაუსის ორჯერ დაწკაპუნების შედეგად გამოჩნდება rpi-gpio out კვანძის რედაქტირების ფანჯარა. ამ ფანჯრის Pin განყოფილებაში მოვნიშნავთ 32-GPIO12, Type ველში ვირჩევთ Digital output-ს, Name ველში ვწერთ LED-ს.

სურ. 73. rpi-gpio out კვანძის თვისებები

Node-RED-ის სამუშაო გარემოში მივიღებთ სამ კვანძს, მათ ვაკავშირებთ ისე, როგორც ეს სურ. 74-ზეა მოცემული.

სურ. 74. კვანძების დაკავშირება

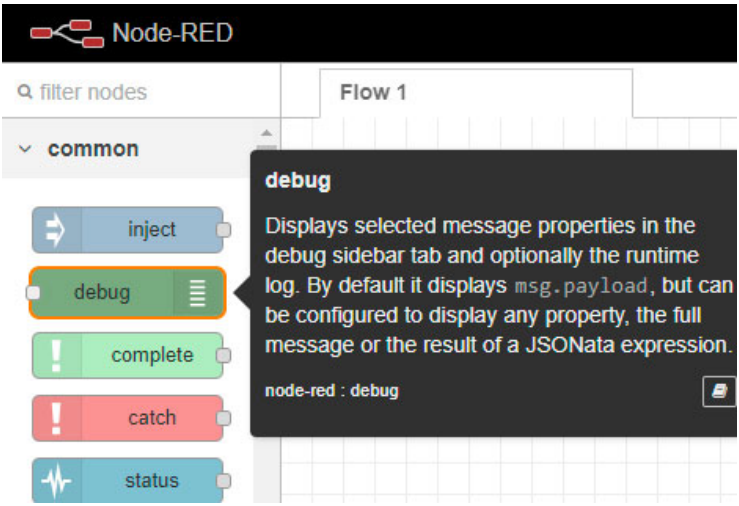
ნაკადის (Flow 1) შენახვის მიზნით, ეკრანის ზედა მარჯვენა კუთხეში მაუსის საშუალებით ვაჭერთ Deploy ღილაკს. ON კვანძის მარცხნივ მდებარე ღილაკზე დაჭერის შედეგად, შუქდიოდი აინთება; OFF კვანძის მარცხნივ მდებარე ღილაკზე დაჭერის შედეგად, შუქდიოდი ჩაქრება.



სურ. 75. შუქდიოდის მართვა Node-RED-ის საშუალებით

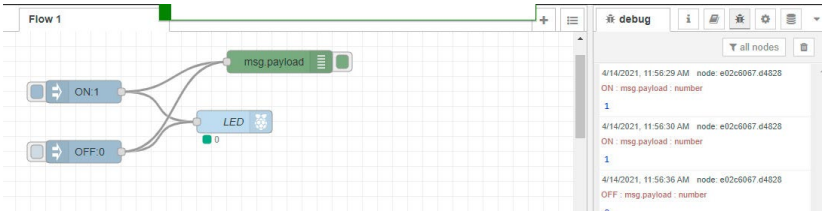
Node-RED-ის სამუშაო გარემოში ასევე შეგვიძლია გამოვიტანოთ ინფორმაცია პროგრამის გამართვის თაობაზე.

ამისათვის ჩვენს მიერ შექმნილი კვანძები შევაერთოთ debug კვანძთან.



სურ.76. debug კვანძი

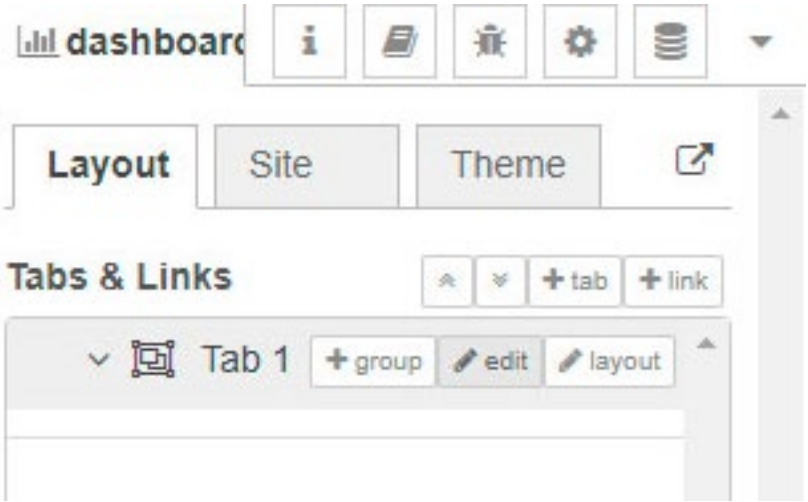
debug კვანძი დავაკავშიროთ ჩვენს მიერ შექმნილ inject ტიპის კვანძებთან, შემდეგ დავაჭიროთ Deploy ღილაკს. ON და OFF ღილაკებზე დაჭერისას, Node-RED-ის სამუშაო გარემოს მარჯვენა ნაწილში, debug ჩანართში, გამოჩნდება შესაბამისი შეტყობინებები.



სურ.77. debug კვანძთან დაკავშირება

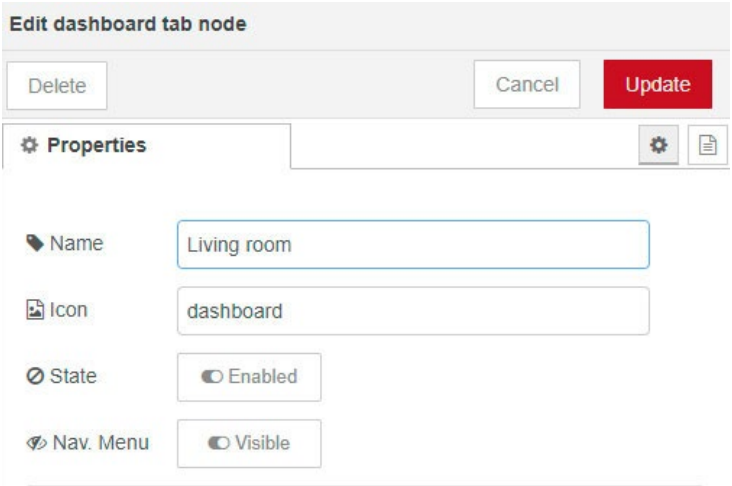
იმისათვის რომ შევქმნათ მომხმარებლის ინტერფეისი, წავშალოთ inject და debug ტიპის კვანძები, LED კვანძს დავარქვათ LAMP.

Node-RED-ის სამუშაო გარემოს მარჯვენა ნაწილში, დავაჭიროთ dashboard → Layout → Tabs & Links→ + tab → Tab 1 → edit ელემენტებს.



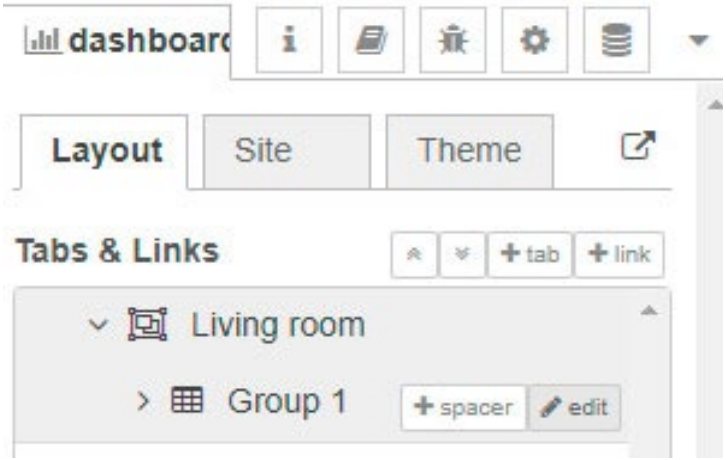
სურ. 78. მომხმარებლის ინტერფეისის შექმნა

აღნიშნული პროცედურების განხორციელების შედეგად მიღებულ ფანჯარაში Name ველში ჩავწეროთ Living room და დავაჭიროთ update ღილაკს. (სურ. 79).



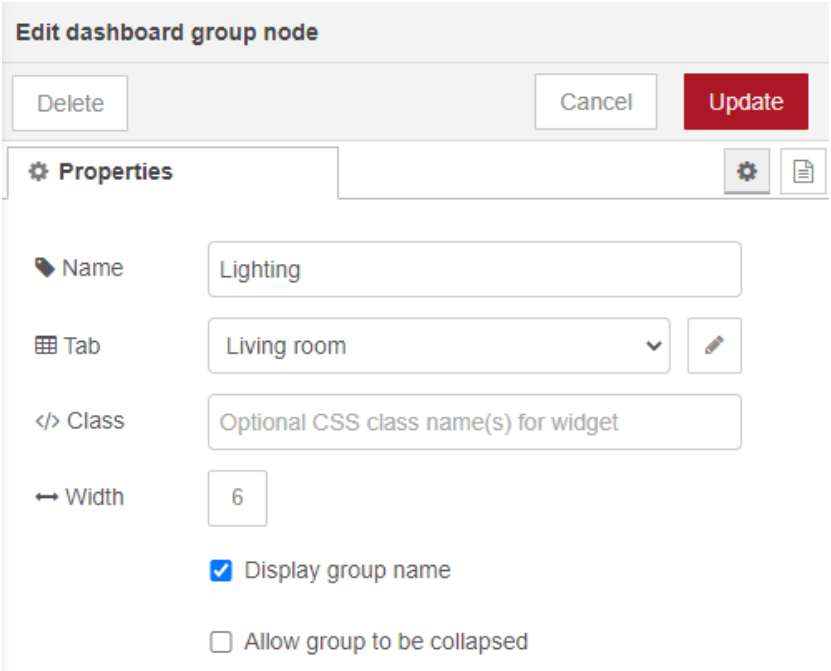
სურ. 79. მომხმარებლის ინტერფეისის შექმნა

შემდეგ ეტაპზე შევასრულოთ მოქმედებები: Living room → + group → edit.



სურ. 80. მომხმარებლის ინტერფეისის შექმნა

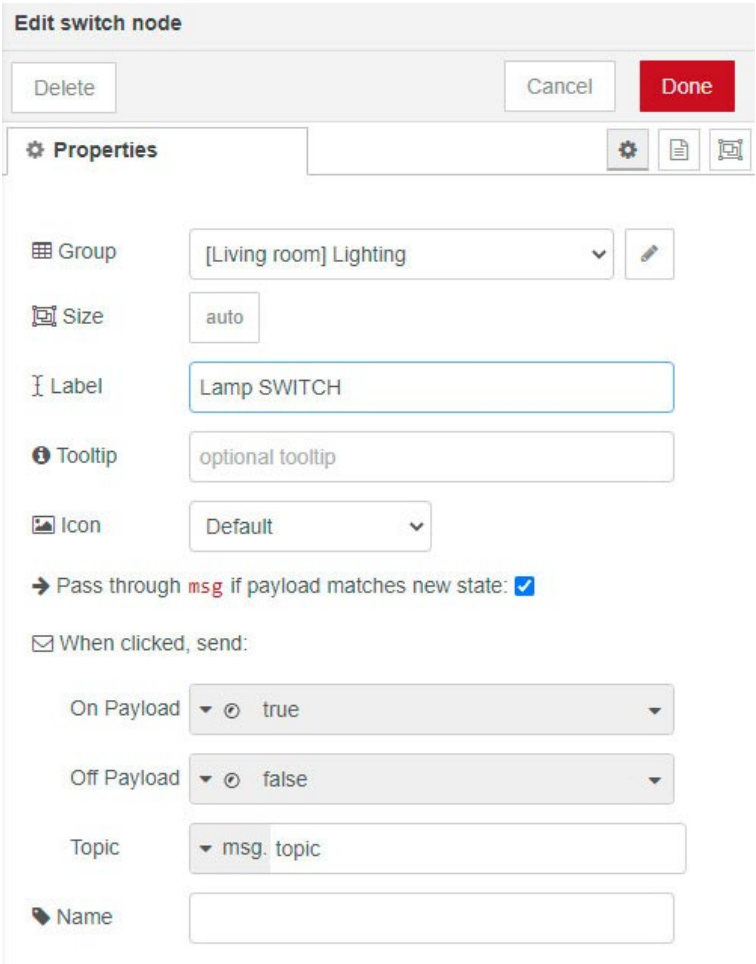
Edit ღილაკზე დაჭერის შედეგად გამოჩნდება ფანჯარა, რომლის Name ველში ვწერთ Lighting-ს. დავაჭიროთ Update ღილაკს.



სურ. 81. მომხმარებლის ინტერფეისის შექმნა

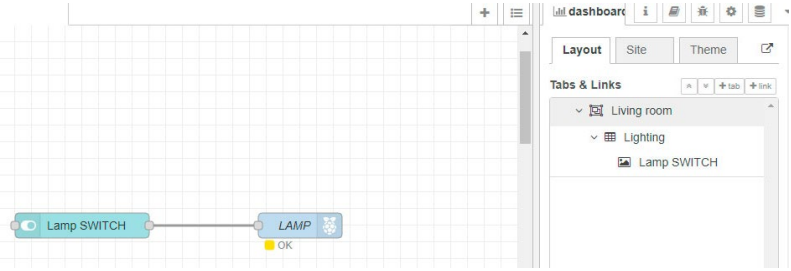
Node-RED-ის სამუშაო გარემოში დავამატოთ switch კვანძი. მასზე მაუსის ღილაკის ორჯერ დაწკაპუნებით ეკრანზე გამოჩნდება კვანძის რედაქტირების ფანჯარა (სურ.85). Label ველში ჩავწეროთ Lamp SWITCH. მივაქციოთ

ყურადღება, რომ Group ველში გვქონდეს [Living room] Lighting.



სურ. 82. switch კვანძის თვისებები

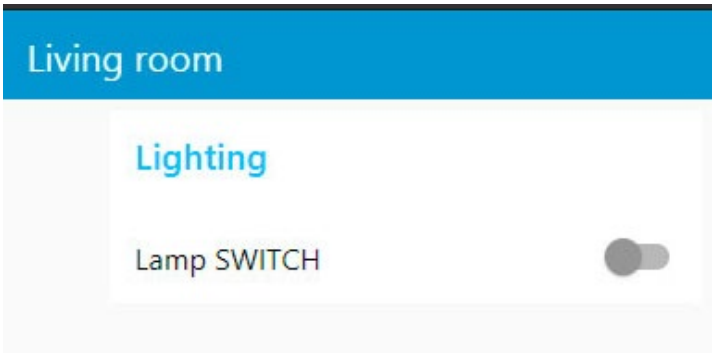
შევაერთოთ კვანძები ერთმანეთთან, დავაჭიროთ Deploy ღილაკს (სურ.83).



სურ. 83. კვანძების შეერთება

ბრაუზერის ახალ ჩანართში ავკრიფოთ მისამართი: http://raspberry-pi-ip-address:1880/ui

ეკრანზე მივიღებთ მომხმარებლის ინტერფეისს, რომელშიც მოთავსებულია Lamp SWITCH ჩამრთველი. მასზე დაწკაპუნებით ხდება შუქდიოდის ჩართვა ან გამორთვა.



სურ. 84. მომხმარებლის ინტერფეისის საბოლოო სახე

MQTT პროტოკოლის აღწერა და MQTT-ის ინსტალაცია Raspberry pi-ზე

IoT მოწყობილობების სამართავად არსებობს ბევრი სხვადასხვა პროტოკოლი. მათ შორის ერთ-ერთია MQTT პროტოლი, რომელსაც ჩვენ გამოვიყენებთ წიგნში მოცემული ამოცანების განსახორციელებლად.

MQTT წარმოადგენს სპეციალიზებულ პროტოკოლს, რომლის გამოყენებაც შესაძლებელია დაბალსიჩქარულ, არასაიმედო ქსელებში. MQTT-ს უპირატესობაა დიდი მოცულობის მონაცემების გადაცემის შესაძლებლობა ანალიტიკური პლატფორმებისა და ღრუბლოვანი სერვერებისთვის.

MQTT პროტოკოლის საშუალებით შეტყობინებები გადაიცემა გამომქვეყნებელს (publisher), ბროკერს (MQTT – broker) და გამომწერს (subscriber) შორის. უფრო კონკრეტულად, გამომქვეყნებლები ბროკერის საშუალებით გადასცემენ მონაცემებს გამომწერებს. გამომქვეყნებლებსა და გამომწერებს ხშირად MQTT კლიენტებს (MQTT Clients) უწოდებენ.

გამომქვეყნებლები (publishers), როგორც წესი, შეტყობინებებს აგზავნიან; მაგალითად, გამომქვეყნებლები შეიძლება იყვნენ IoT მოწყობილობებში ჩამონტაჟებული სენსორები.

ბროკერი (MQTT-broker) MQTT სისტემის ცენტრალური კვანძია, რომელიც გამომქვეყნებლებსა და გამომწერებს

შორის ინფორმაციის გადაცემას უზრუნველყოფს. ის ერთგვარი შუამავლის როლს თამაშობს MQTT კლიენტებს (გამომქვეყნებლები და გამომწერები) შორის. ბროკერი იღებს ინფორმაციას გამომქვეყნებლებისგან, ამუშავებს მას, გადასცემს გამომწერებს და ასევე, აკონტროლებს ინფორმაციის მიწოდების პროცესს. ჩვეულებრივ, ბროკერს წარმოადგენს სერვერული პროგრამული უზრუნველყოფა (MQTT Server).

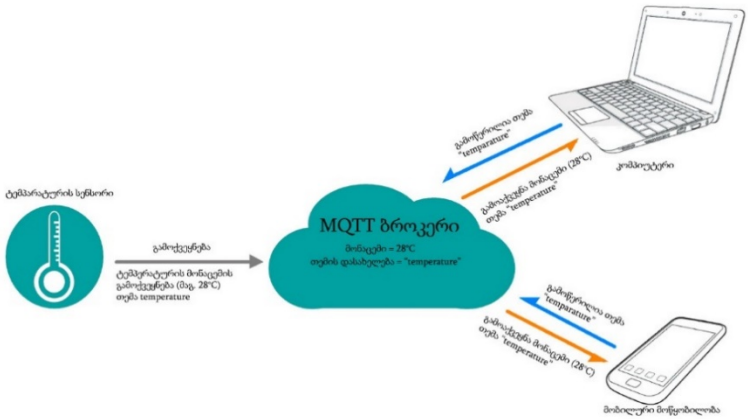
გამომწერები (Subscribers) სენსორებიდან გაგზავნილი მონაცემების საბოლოო მიმღებები არიან; გამომწერი შეიძლება იყოს მაგალითად, ღრუბელში მოთავსებული ანალიტიკური სისტემა.

MQTT პროტოკოლის საშუალებით გადაცემულ მონაცემებს გააჩნიათ შესაბამისი თემა (Topic), ასევე მომსახურების ხარისხი (Quality of Service), რომელიც შეტყობინების გადაცემის პრიორიტეტს გულისხმობს. MQTT-ში გათვალისწინებულია ეწ. QoS (Quality of Service) ალმები:

- QoS 0 დონე - შეტყობინების გადაცემა ხდება ერთჯერ, თუმცა არ არსებობს მიღების გარანტია. შეტყობინების მიმღები არ ადასტურებს მიღებას, მისი ხელახლა გადაგზავნა კი არ ხდება. თუ რაიმე მიზეზის გამო პროცესი შეწყდა, შეტყობინება შეიძლება დაიკარგოს.
- QoS 1 დონე - ამ შემთხვევაში აუცილებლად არსებობს იმის გარანტია, რომ შეტყობინებას მიმღები თუნდაც ერთხელ მაინც მიიღებს. გამგზავნი შეტყობინებას

ინახავს მანამ, სანამ მიმღები მიღებას არ დაადასტურებს. შესაძლებელია, შეტყობინება გაიგზავნოს და მიღებულ იქნას რამდენჯერმე.

- QoS 2 დონე - ეს დონე იძლევა გარანტიას იმისა, რომ თითოეული შეტყობინება შესაბამის მიმღებებს მხოლოდ ერთხელ მიეწოდება. QoS2 ყველაზე ნელი, მაგრამ ამასთანავე ყველაზე უსაფრთხოა. შეფერხების შემთხვევაში, შეტყობინების გადაცემის პროცესი ნელდება, თუმცა ინფორმაცია ადრესატამდე აუცილებლად მიდის კავშირის განახლების შემდეგ.



სურ. 85. MQTT პროტოლი IoT სისტემაში

MQTT სისტემის ასამუშავებლად საჭიროა MQTT ბროკერი. ჩვენ გამოვიყენებთ Mosquitto ბროკერს. სანამ Mosquitto-ს დავაყენებდეთ, განვაახლოთ Raspberry Pi.

ტერმინალის ფანჯარაში ჩავწეროთ ბრძანებები: *sudo apt update* და *sudo apt upgrade*. ამ ბრძანებების შესრულების შემდეგ, დავაინსტალიროთ Mosquitto ბროკერი და mosquitto-clients პაკეტები:

sudo apt install mosquitto -y

sudo apt install mosquitto-clients -y

```
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
iot@raspberrypi:~ $ sudo apt install mosquitto -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  brltty espeak-ng-data gir1.2-atspi-2.0 gir1.2-gstreamer-1.0 gir1.2-wnck-3.0
  libao-common libao4 libatk-adaptor libaudio2 libbluetooth3 libbrlapi0.8
  libdotconf0 libespeak-ng1 libpcaudio0 libpcre2-32-0 libsonic0 libspeechd2
  perl-tk python3-brlapi python3-louis python3-pyatspi python3-speechd
  python3-xdg sound-icons speech-dispatcher speech-dispatcher-audio-plugins
  speech-dispatcher-espeak-ng xbrlapi xkbset
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libcjson1 libdlt2 libev4 libmosquitto1 libwebsockets16
Suggested packages:
  apparmor
The following NEW packages will be installed:
  libcjson1 libdlt2 libev4 libmosquitto1 libwebsockets16 mosquitto
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 591 kB of archives.
After this operation, 1,464 kB of additional disk space will be used.
Get:1 http://raspbian.mirror.net.in/raspbian/raspbian bullseye/main armhf libcjs
on1 armhf 1.7.14-1 [20.8 kB]
Get:2 http://raspbian.mirror.net.in/raspbian/raspbian bullseye/main armhf libdlt
2 armhf 2.18.6-1 [45.3 kB]
Get:3 http://raspbian.mirror.net.in/raspbian/raspbian bullseye/main armhf libev4
```

სურ. 86. Mosquitto MQTT ბროკერის ინსტალაცია

Mosquitto ბროკერისა და mosquitto-clients პაკეტების ინსტალაციის შემდეგ, საჭიროა ბროკერის კონფიგურირება. ტექსტური რედაქტორის (nano) მეშვეობით გავხსნათ mosquitto ბროკერის კონფიგურაციის ფაილი:

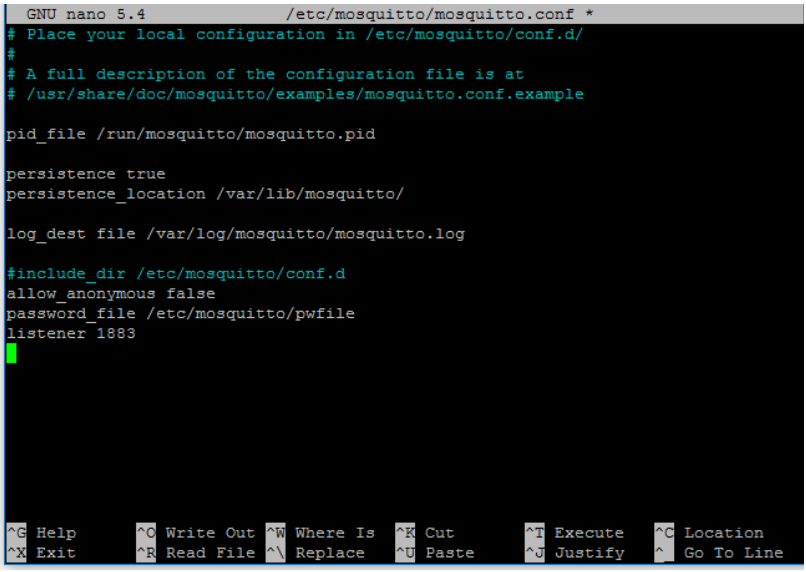
sudo nano /etc/mosquitto/mosquitto.conf

კონფიგურაციის ფაილის ბოლო სტრიქონი (include_dir /etc/mosquitto/conf.d) წავშალოთ ან დავაკომენტაროთ. ფაილის ბოლოს დავამატოთ შემდეგი სტრიქონები (სურ.87):

allow_anonymous false

password_file /etc/mosquitto/pwfile

listener 1883



სურ.87. MQTT ბროკერის კონფიგურირება

ამ სტრიქონების დამატება ნიშნავს, რომ mosquitto ბროკერთან დაკავშირება შესაძლებელი იქნება მხოლოდ მომხმარებლის სახელისა და პაროლის მეშვეობით; mosquito ბროკერი მიაყურადებს შეტყობინებებს პორტზე ნომრით 1883.

იმ შემთხვევაში, თუ ჩვენ არ გვსურს, რომ ბროკერმა მოითხოვოს მომხმარებლის სახელი და პაროლი, კონფიგურაციის ფაილის ბოლოში არ უნდა დავამატოთ სტრიქონები allow_anonymous false და password_file /etc/mosquitto/pwfile. mosquitto ბროკერის კონფიგურაციის ფაილში ცვლილებების განხორციელების შემდეგ დავხუროთ ეს ფაილი CTRL+x ღილაკზე დაჭერით, შემდეგ აუცილებლად უნდა დავაჭიროთ Y კლავიშს.

რადგანაც ჩვენ გადავწყვიტეთ, რომ mosquitto ბროკერთან დასაკავშირებლად გამოვიყენებთ მომხმარებლის სახელსა და პაროლს, ცხადია, უნდა განვსაზღვროთ ეს პარამეტრები. ტერმინალის ფანჯარაში ავკრიფოთ შემდეგი ბრძანება: *sudo mosquitto_passwd -c /etc/mosquitto/pwfile username* username-ის ნაცვლად ავკრიფოთ მომხმარებლის სახელი (ჩვენს შემთხვევაში, pi), შემდეგ კი - პაროლი, როდესაც სისტემა მის შეტანას მოგვთხოვს.

```
iot@raspberrypi:~ $ sudo nano /etc/mosquitto/mosquitto.conf
iot@raspberrypi:~ $ sudo mosquitto_passwd -c /etc/mosquitto/pwfile pi
Password:
Reenter password:
```

სურ. 88. Mosquitto MQTT ბროკერის მომხმარებლის და პაროლის შექმნა

რადგანაც ჩვენ ახლახან შევცვალეთ mosquitto ბროკერის კონფიგურაციის ფაილი, *sudo reboot* ბრძანებით გადავტვირთოთ Raspberry Pi. მას შემდეგ, რაც Raspberry Pi

გადაიტვირთება, ჩვენ უკვე ხელთ გვექნება სრულად ფუნქციონირებადი MQTT ბროკერი.

Raspberry Pi-ში mosquitto-ს დაინსტალირების შემდეგ ჩვენ შეგვიძლია ჩავატაროთ პატარა ტესტი, რათა დავრწმუნდეთ, რომ ყველაფერი სწორად მუშაობს. ამ მიზნით, შეგვიძლია გამოვიყენოთ ორი ბრძანება - *mosquitto_pub* და *mosquitto_sub*.

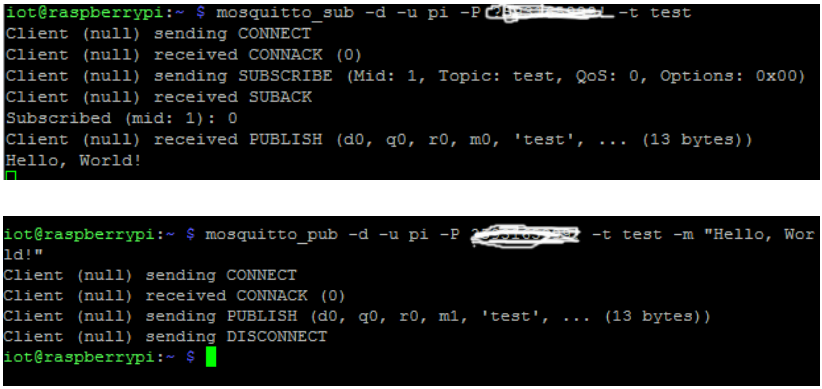
ბროკერის დასატესტად დაგვჭირდება ტერმინალის ორი ფანჯრის გახსნა. ტერმინალის ერთ-ერთ ფანჯარაში ავკრიფოთ ქვემოთ მოცემული ბრძანება. ცხადია, username-სა და password-ის ნაცვლად უნდა ჩავწეროთ ჩვენს მიერ შერჩეული მომხმარებლის სახელი და პაროლი:

mosquitto_sub -d -u username -P password -t test

იმ შემთხვევაში, თუ ბროკერთან დაკავშირება ხდება მომხმარებლის სახელისა და პაროლის გარეშე, *mosquitto_sub* ბრძანებას ექნება შემდეგი სახე: *mosquitto_sub -d -t test*

mosquitto_sub ბრძანება იწერს თემას (topic) და ეკრანზე გამოიტანს ნებისმიერ შეტყობინებას, რომელიც გაიგზავნება ამ თემით ტერმინალის ფანჯარაში. ბრძანებაში *-d* ნიშნავს გამართვის რეჟიმს, ამიტომაც ყველა შეტყობინებისა და მოქმედების გამოტანა ხდება ეკრანზე. *-u* აღნიშნავს მომხმარებლის სახელს, *-P* - პაროლს, *-t* - იმ თემის სახელს, რომლის გამოწერაც გვინდა. ჩვენს შემთხვევაში, თემის სახელია “test”.

ტერმინალის მეორე ფანჯარაში, ჩვენ უნდა გამოვაქვეყნოთ შეტყობინება თემაში „test”. ავკრიფოთ ბრძანება: *mosquitto_pub -d -u username -P password -t test -m "Hello, World!".* enter-ზე დაჭერის შემდეგ, ტერმინალის პირველ ფანჯარაში დავინახავთ შეტყობინებას "Hello, World!".



სურ. 89. Mosquitto-ს მუშაობის შემოწმება

სენსორიდან მიღებული მონაცემების გამოქვეყნება MQTT -ის მეშვეობით

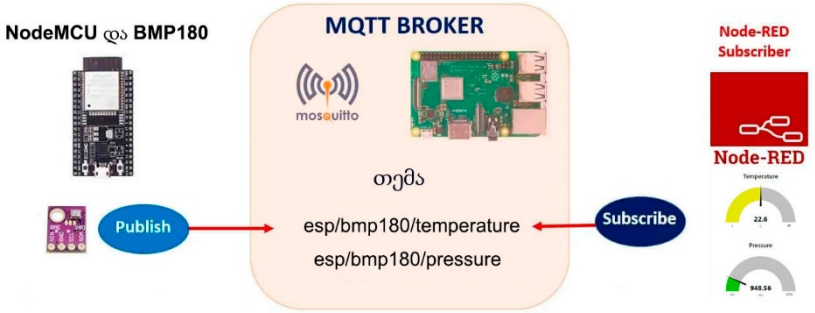
განვიხილოთ ამოცანა, სადაც სენსორიდან მიღებულ მონაცემებს გადავცემთ MQTT პროტოკოლის მეშვეობით და გამოვაქვეყნებთ Node-RED-ის მიერ შექმნილ გრაფიკულ გარემოში.

NodeMCU მოდულის node-RED პლატფორმასთან ინტეგრირებისთვის MQTT პროტოკოლი გამოიყენება.

NodeMCU-თვის, როგორც MQTT კლიენტისთვის, არსებობს სპეციალიზებული ბიბლიოთეკები.

განვიხილოთ, როგორ გამოვაქვეყნოთ NodeMCU-ს მიერ BMP180 სენსორიდან მიღებული ტემპერატურისა და წნევის მნიშვნელობები MQTT ბროკერისა და Node-RED პლატფორმის მეშვეობით.

ქვემოთ მოცემული გამოსახულება (სურ. 90) სრულად გამოხატავს განსახილველი ამოცანის არსს.

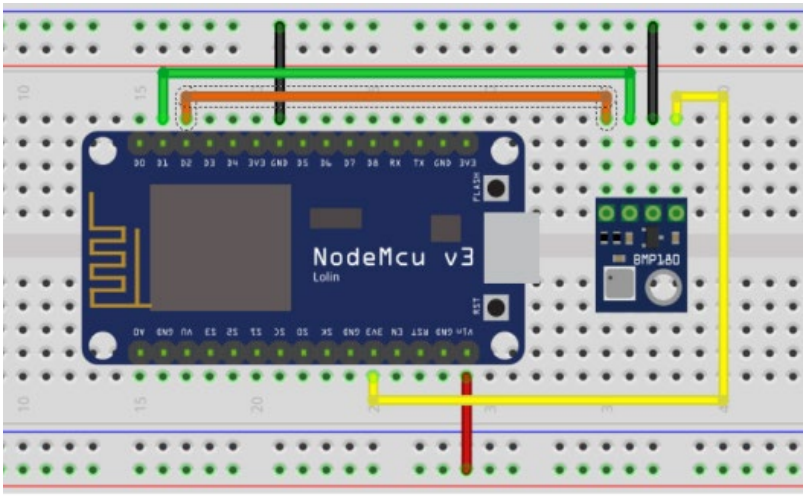


სურ. 90. სენსორიდან მიღებული მონაცემების გამოქვეყნება MQTT ბროკერის მეშვეობით

როგორც სურ.90-დან ჩანს, NodeMCU იღებს ტემპერატურისა და წნევის მნიშვნელობებს BMP180 სენსორიდან. NodeMCU ტემპერატურის და წნევის მნიშვნელობებს აქვეყნებს შეტყობინებების სახით, რომელთა თემების სახელწოდებაა შესაბამისად, „esp/bmp180/temperature“ და „esp/bmp180/pressure”. ამ ინფორმაციას იღებს MQTT ბროკერი. როგორც ზემოთ აღვნიშნეთ, ბროკერი ერთგვარი შუამავლის როლს თამაშობს MQTT კლიენტებს (გამომქვეყნებლები და

გამომწერები) შორის. ჩვენს შემთხვევაში, NodeMCU არის გამომქვეყნებელი, Node-RED კი - გამომწერი. Node-RED იწერს თემებს: „esp/bmp180/temperature“ და „esp/bmp180/pressure”. Node-RED-ს მიღებული ინფორმაცია გამოჰყავს გამზომ მოწყობილობებზე, რომლებიც თვალსაჩინოდ გვიჩვენებს სიდიდეების მნიშვნელობას.

ავაგოთ სურ. 91-ზე მოცემული სქემა. BMP180 სენსორის SDA საკონტაქტო გამომყვანი შევაერთოთ NodeMCU-ს D1 გამომყვანთან, ხოლო SCL საკონტაქტო გამომყვანი შევაერთოთ D2-თან.



სურ. 91. BMP180 სენსორის მიერთება NodeMCU-სთან

ჩავწეროთ ქვემოთ მოცემული კოდი Arduino IDE-ში. Arduino IDE-ში, Sketch->Include Library> Manage Libraries

მენიუში მოვძებნოთ PubSubClient ბიბლიოთეკა და დავა-ინსტალიროთ. წინა ამოცანების მსგავსად, ამ მაგალითშიც უნდა მივუთითოთ ჩვენი ქსელის და MQTT ბროკერის პარამეტრები. ავტვირთოთ კოდი Arduino IDE-ში.

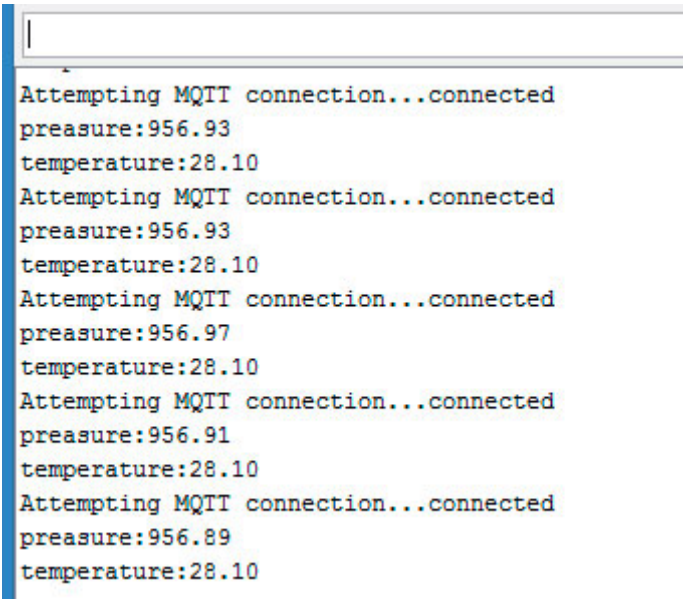
```
//MQTT კლიენტის პროტოკოლის ბიბლიოთეკა. ეს ბიბლიო-თეკა საშუალებას იძლევა გამოიწეროს და გამოაქვეყნოს შეტყობინებები
#include <PubSubClient.h>
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან დაკავშირებისთვის
#include <ESP8266WiFi.h>
//Adafruit BMP085/BMP180-ის ბიბლიოთეკა
#include <Adafruit_BMP085.h>
//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
// Mosquitto MQTT Broker-ის ip მისამართი, იგივეა რაც Raspberry pi-ს ip მისამართი
#define MQTT_HOST "your  mosquitto ip address"
// Mosquitto MQTT Broker-ის  პორტის ნომერი
#define MQTT_PORT 1883
// Mosquitto MQTT Broker-ის  მომხმარებლის სახელი
#define MQTT_USERNAME "your mosquitto username"
// Mosquitto MQTT Broker-ის  მომხმარებლის პაროლი
#define MQTT_PASSWORD "your mosquitto password"
//MQTT კლიენტის ID, რომელიც გამოიყენება სერვერთან დასაკავშირებლად
#define MQTT_CLIENT_ID "NodeMCUClient"
// MQTT თემები
#define MQTT_PUB_TEMP "esp/bmp180/temperature"
#define MQTT_PUB_PRES "esp/bmp180/pressure"
```

```
//კლასის ობიექტების შექმნა
WiFiClient NodeMCUHome;
PubSubClient client(NodeMCUHome);
// BMP180 სენსორის ინიციალიზაცია
Adafruit_BMP085 myBMPSensor;
//ამ ცვლადში ინახება ტემპერატურის მნიშვნელობა
float temp;
//ამ ცვლადში ინახება წნევის მნიშვნელობა
float pres;
//ტაიმერის დამხმარე ცვლადები
long now = millis();
long lastMeasure = 0;
//მასივი,სადაც ინახება MQTT შეტყობინებები
char msg[10];
void check_sensors() {
  // BMP180 სენსორის მნიშვნელობების წაკითხვა
  pres = myBMPSensor.readPressure() / 100.0;
  temp = myBMPSensor.readTemperature();
}
// NodeMCU-ს ხელახალი დაკავშირება MQTT ბროკერთან
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(MQTT_CLIENT_ID, MQTT_USERNAME,
MQTT_PASSWORD)) {
      Serial.println("connected");
      //თემების გამოწერა
      client.subscribe(MQTT_PUB_TEMP);
      client.subscribe(MQTT_PUB_PRES);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
```

```
      delay(5000);
    }
  }
}
void setup() {
  Serial.begin(115200);
  //უსადენო ქსელთან დაკავშირება
  WiFi.begin(ssid, wifiPassword);
  Serial.println("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP
Address: ");
  Serial.println(WiFi.localIP());
  //BMP სენსორის შემოწმება
  if (!myBMPSensor.begin()) {
    Serial.println("Could not find a valid BMP180
sensor, check wiring!");
    while (1) {}
  }
  //ბროკერთან დასაკავშირებლად უნდა გამოვიძახოთ
მეთოდი setServer, რომელსაც გადავცემთ ბროკერის
მისამართს და პორტს.
  client.setServer(MQTT_HOST, MQTT_PORT);
}
void loop() {
  //MQTT ბროკერთან კავშირის შემოწმება
  if (!client.connected()) {
    reconnect();
  }
```

```
  now = millis();
  // აქვეყნებს ტენიანობის მნიშვნელობას ყოველ 30 წამში ერთხელ
  if (now - lastMeasure > 30000) {
    lastMeasure = now;
    check_sensors();
    //მცოცავმძიმიანი რიცხვის მნიშვნელობას ამრგვალებს ისე რომ მძიმის შემდეგ მხოლოდ ერთი ციფრი იყოს და ინახავს msg[] მასივში
    sprintf(msg, "%.1f", pres);
    //MQTT-ში გამოქვეყნება
    client.publish(MQTT_PUB_PRES, msg);
    Serial.print("pressure:");
    Serial.println(pres);
    //მცოცავმძიმიანი რიცხვის მნიშვნელობას ამრგვალებს ისე რომ მძიმის შემდეგ მხოლოდ ერთი ციფრი იყოს და ინახავს msg[] მასივში
    sprintf(msg, "%.1f", temp);
    //MQTT-ში გამოქვეყნება
    client.publish(MQTT_PUB_TEMP, msg);
    Serial.print("temperature:");
    Serial.println(temp);
  }
}
```

კოდის გაშვების შემდეგ გავხსნათ მიმდევრობითი მონიტორის ფანჯარა, სადაც დავინახავთ NodeMCU-ს გამოქვეყნებულ შეტყობინებებს და თემებს.

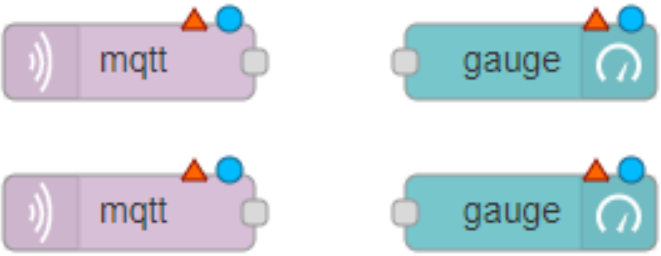


სურ.92. NodeMCU-ს მიერ გამოქვეყნებული შეტყობინებები

NodeMCU ყოველ 30 წამში ერთხელ კითხულობს ტემპერატურისა და წნევის მნიშვნელობებს სენსორიდან და აქვეყნებს თემებში: esp/bmp180/temperature და esp/bmp180/pressure.

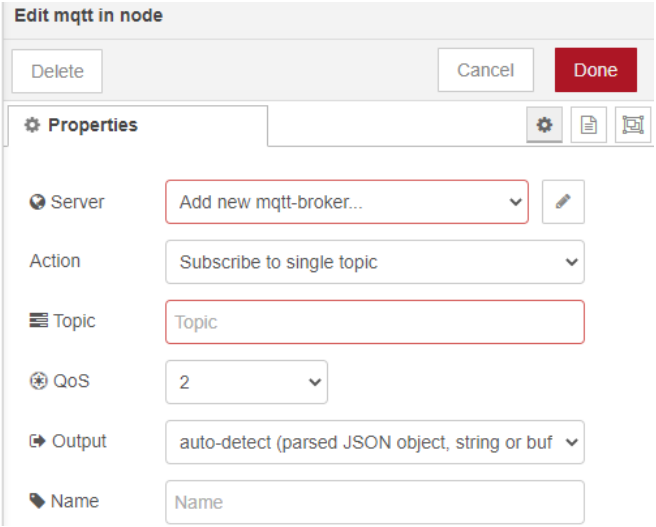
შევქმნათ პროგრამის ინტერფეისი Node-RED გარემოში. Node-RED-ში შესასვლელად გადავიდეთ მისამართზე: http://raspberry-pi-ip-address:1880

Node-RED გარემოში გადმოვიტანოთ ოთხი კვანძი, აქიდან ორი MQTT in და ორი გამზომი მოწყობილობის კვანძი (gauge).



სურ.93. mqtt in და gauge კვანძები

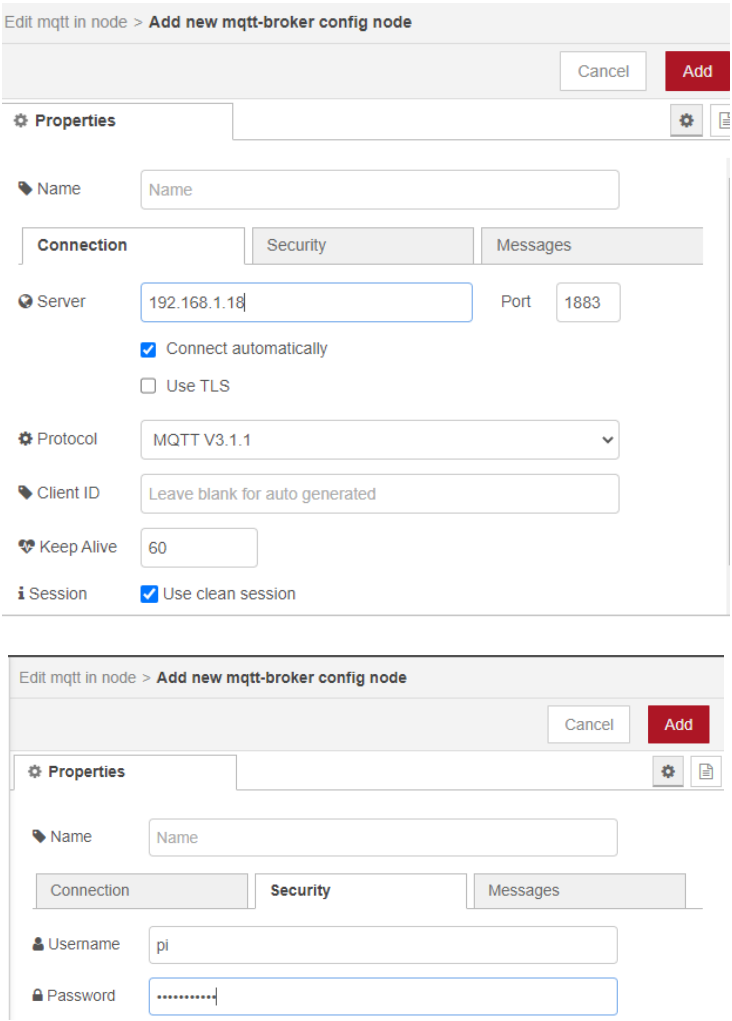
mqtt in კვანძის რედაქტირებისთვის, დავაჭიროთ ამ კვანძს ორჯერ.



სურ.94. mqtt in კვანძის თვისებები

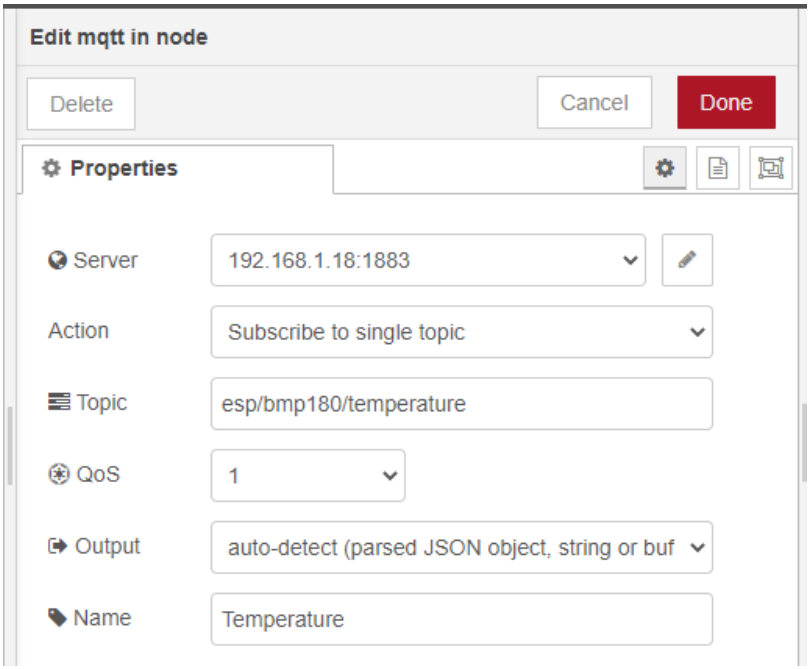
mqtt in კვანძის რედაქტირების ფანჯრის Server-ის ველთან დავაჭიროთ რედაქტირების ღილაკზე და შევავსოთ შესაბამისი ველები Connection და Security ჩანართებში

(სურ.95). Server ველში ვწერთ Mosquitto ბროკერის ip მისამართსა და პორტის ნომერს; Security ჩანართის Username და Password ველებში ვწერთ ბროკერის მომხმარებლის სახელსა და პაროლს.



სურ. 95. mqtt ბროკერის კონფიგურირება

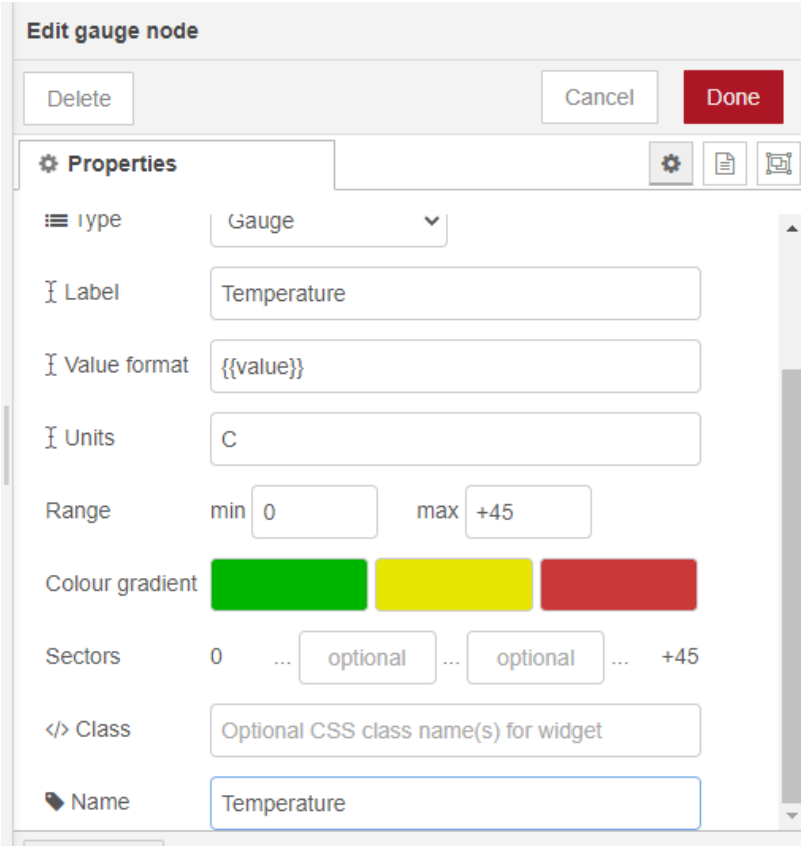
დავაჭიროთ Add ღილაკს. ამ ღილაკზე დაჭერით ვბრუნდებით mqtt in კვანძის რედაქტირების ფანჯარაში, Topic ველში ვწერთ თემის დასახელებას.



სურ.96. mqtt in კვანძის რედაქტირება

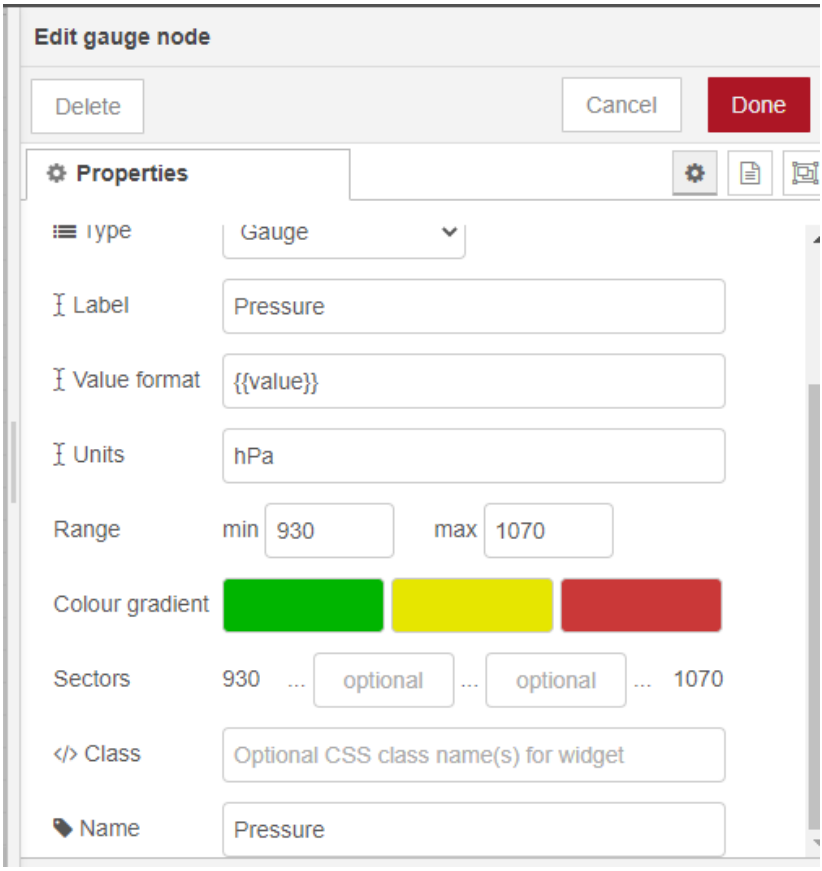
იგივე პროცედურა უნდა გავიმეოროთ მეორე mqtt in კვანძისთვის იმ განსხვავებით, რომ Topic ველში ვწერთ მეორე თემის (esp/bmp180/pressure) დასახელებას.

დავაჭიროთ gauge კვანძს და მოვახდინოთ მისი თვისებების რედაქტირება. (სურ.97)



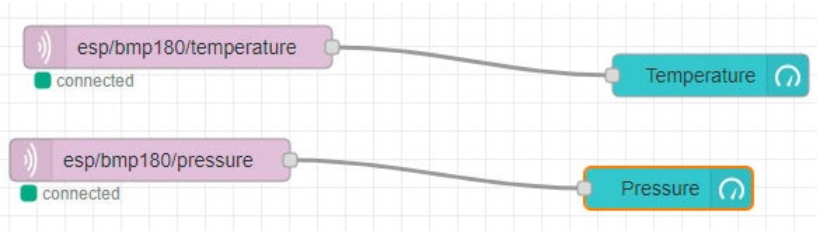
სურ. 97. gauge კვანძის რედაქტირება ტემპერატურის მონაცემებისთვის

იგივე პროცედურა გავიმეოროთ მეორე gauge კვანძისთვის (სურ. 98).



სურ. 98. gauge კვანძის რედაქტირება წნევის მონაცემებისთვის

შევაერთოთ კვანძები ერთმანეთთან ისე, როგორც მოცემულია სურ. 99-ზე.



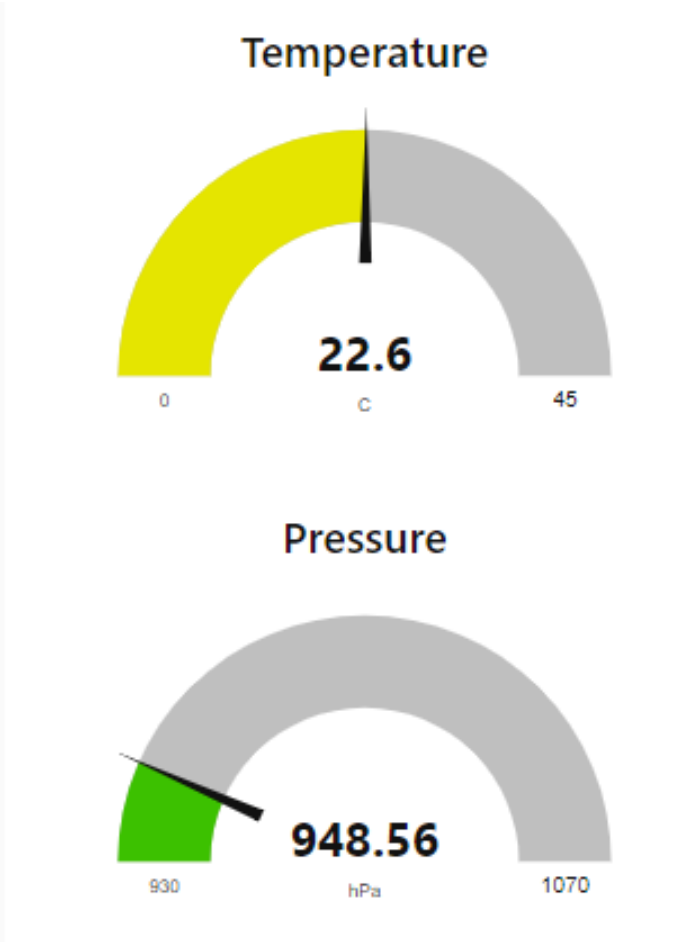
სურ. 99. კვანძების შეერთება

დავაჭიროთ ეკრანის ზედა მარჯვენა კუთხეში მოთავსებულ Deploy ღილაკს.

სურ. 100. Deploy ღილაკი

ბრაუზერის სამისამართო ველში ავკრიფოთ: http://raspberry-pi-ip-address:1880/ui. raspberry-pi-ip-address წარმოადგენს ჩვენს Raspberry pi-ს ip მისამართს.

ბრაუზერის ფანჯარაში მივიღებთ შემდეგ სურათს:

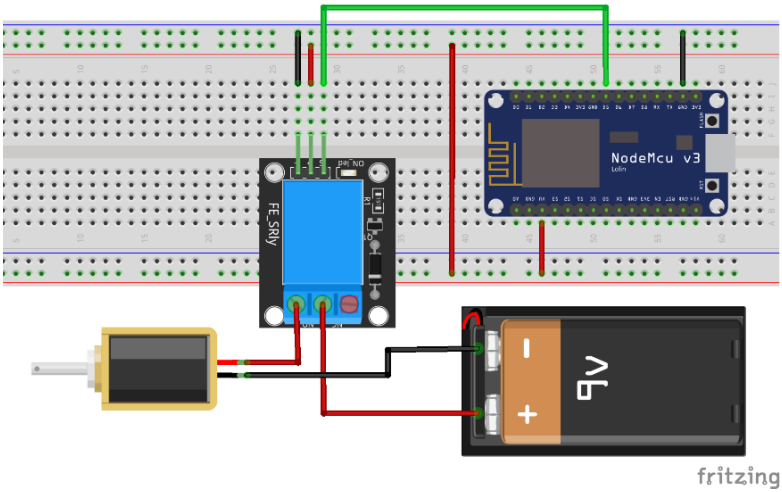


სურ. 101. ტემპერატურისა და წნევის მონაცემების ჩვენება Node-RED გარემოში

IoT-ზე დაფუძნებული კარის საკეტი

ქვემოთ განვიხილავთ მაგალითს, რომლის საშუალებითაც შესაძლებელია გადავწყვიტოთ Wi-Fi უსადენო ქსელით კარის გაღება-დაკეტვის ამოცანა. ასეთი სისტემები მომხმარებლისთვის საინტერესოა იმით, რომ დისტანციურად მართვის საშუალებას იძლევა და თავიდან გვაცილებს იმ უსიამოვნებებს, რომელიც გასაღების ძებნასა და ტარებასთან არის დაკავშირებული.

ავაგოთ სურ. 102-ზე მოცემული სქემა. რელეს მოდულის შესასვლელი გამომყვანი მივუერთოთ NodeMCU-ს D5 საკონტაქტო გამომყვანს. იმის გამო, რომ სოლენოიდი მოიხმარს დიდ დენს და სამუშაოდ სჭირდება 9V და მეტი ძაბვა, ამიტომ მისთვის აუცილებელია ცალკე ელემენტის ან აკუმულატორის გამოყენება.



სურ. 102. სოლენოიდისა და რელეს მოდულის მიერთების სქემა

ჩავწეროთ ქვემოთ მოცემული კოდი Arduino IDE-ში და ავტვირთოთ.

```
// MQTT კლიენტის პროტოკოლის ბიბლიოთეკა. ეს
ბიბლიოთეკა საშუალებას იძლევა გამოიწეროს და
გამოაქვეყნოს შეტყობინებები
#include <PubSubClient.h>
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან
დაკავშირებისთვის
#include <ESP8266WiFi.h>
//Adafruit BMP085/BMP180-ის ბიბლიოთეკა
#include <Adafruit_BMP085.h>
//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
// Mosquitto MQTT Broker-ის ip მისამართი, იგივეა რაც
Raspberry pi-ს ip მისამართი
#define MQTT_HOST "your  mosquitto ip address"
// Mosquitto MQTT Broker-ის  პორტის ნომერი
#define MQTT_PORT 1883
// Mosquitto MQTT Broker-ის  მომხმარებლის სახელი
#define MQTT_USERNAME "your mosquitto username"
// Mosquitto MQTT Broker-ის  მომხმარებლის პაროლი
#define MQTT_PASSWORD "your mosquitto password"
//MQTT კლიენტის ID, რომელიც გამოიყენება სერვერთან
დასაკავშირებლად
#define MQTT_CLIENT_ID "NodeMCUClient"
// საკონტაქტო გამომყვანი, რომელიც მიერთებულია
რელეს მოდულთან
#define solenoid_pin D5
//NodeMCU-ს WiFiClient კლასის შექმნა, რომლის საშუა-
ლებითაც ხდება MQTT ბროკერთან დაკავშირება
```

```
WiFiClient NodeMCUHome;
PubSubClient client(NodeMCUHome);
//Wi-Fi ქსელთან დაკავშირება
void setup_wifi() {
  WiFi.begin(ssid, wifiPassword);
  Serial.println("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP
Address: ");
  Serial.println(WiFi.localIP());
}
// ეს ფუნქცია სრულდება მაშინ, როდესაც ქსელში
არსებული რომელიმე მოწყობილობა აქვეყნებს შეტყობინებას
თემით, რომელიც NodeMCU-ს აქვს გამოწერილი
void callback(String topic, byte* message, unsigned
int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();
   //თუ მიღებული თემა არის door/solenoid, მაშინ
ვამოწმებთ თემის შეტყობინებას, არის ის on თუ off, ამის
შემდეგ ხდება გადაწყვეტილების მიღება სოლენოიდის
ჩართვის ან გამორთვის თაობაზე
```

```
  if (topic == "door/solenoid") {
    Serial.print("Changing door solenoid state to ");
    if (messageTemp == "on") {
      digitalWrite(solenoid_pin, HIGH);
      Serial.print("On");
    } else if (messageTemp == "off") {
      digitalWrite(solenoid_pin, LOW);
      Serial.print("Off");
    }
  }
  Serial.println();
}
// NodeMCU-ს ხელახალი დაკავშირება MQTT ბროკერთან
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(MQTT_CLIENT_ID, MQTT_USERNAME,
MQTT_PASSWORD)) {
      Serial.println("connected");
      client.subscribe("door/solenoid");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
void setup() {
  pinMode(solenoid_pin, OUTPUT);
  digitalWrite(solenoid_pin, 0);
  Serial.begin(115200);
  setup_wifi();
```

```
  //კლიენტისთვის MQTT ბროკერის ip მისამართისა და
პორტის მითითება
  client.setServer(MQTT_HOST, MQTT_PORT);
  //ეს ფუნქცია სრულდება, როდესაც მოდის შეტყობინება
MQTT ბროკერისგან
  client.setCallback(callback);
}
void loop() {
  //MQTT ბროკერთან კავშირის შემოწმება
  if (!client.connected()) {
    reconnect();
  }
  //MQTT ბროკერიდან შეტყობინების არსებობის შემოწმება
  if (!client.loop())
    client.connect(MQTT_CLIENT_ID);
}
```

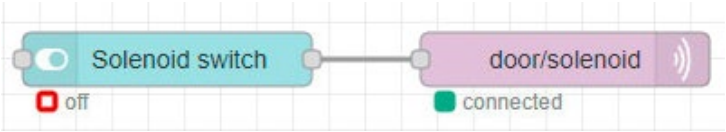
სკეტჩის გაშვების შემდეგ მიმდევრობითი მონიტორის ფანჯარაში მივიღებთ:

```
Connecting
.....
Connected to WiFi network with IP Address: 192.168.1.14
Attempting MQTT connection...connected
```

სურ. 103. პროგრამის გაშვების შედეგები მიმდევრობით მონიტორში

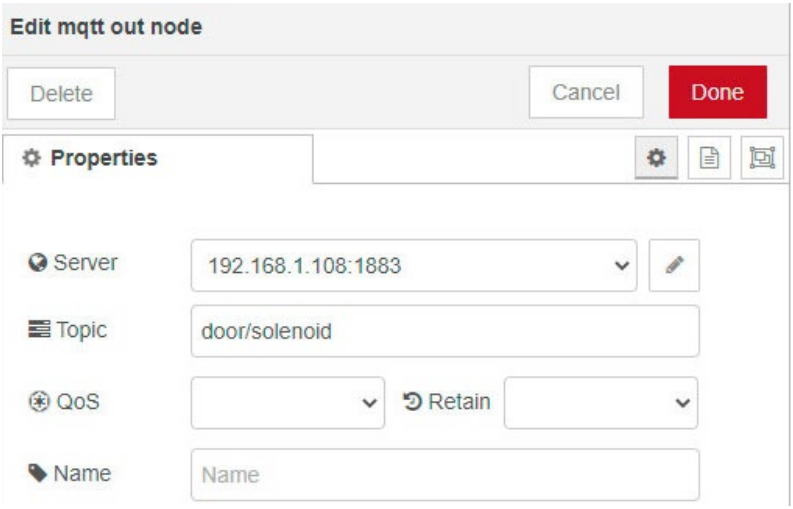
NodeMCU-სა და Node-RED-ის დასაკავშირებლად, Node-RED-ის გარემოში დავამატოთ mqtt out კვანძი. სოლენოიდის მართვისთვის დავამატოთ switch კვანძი. კვანძები

დავუკავშიროთ ერთმანეთს ისე, როგორც ეს სურ. 104-ზეა მოცემული.



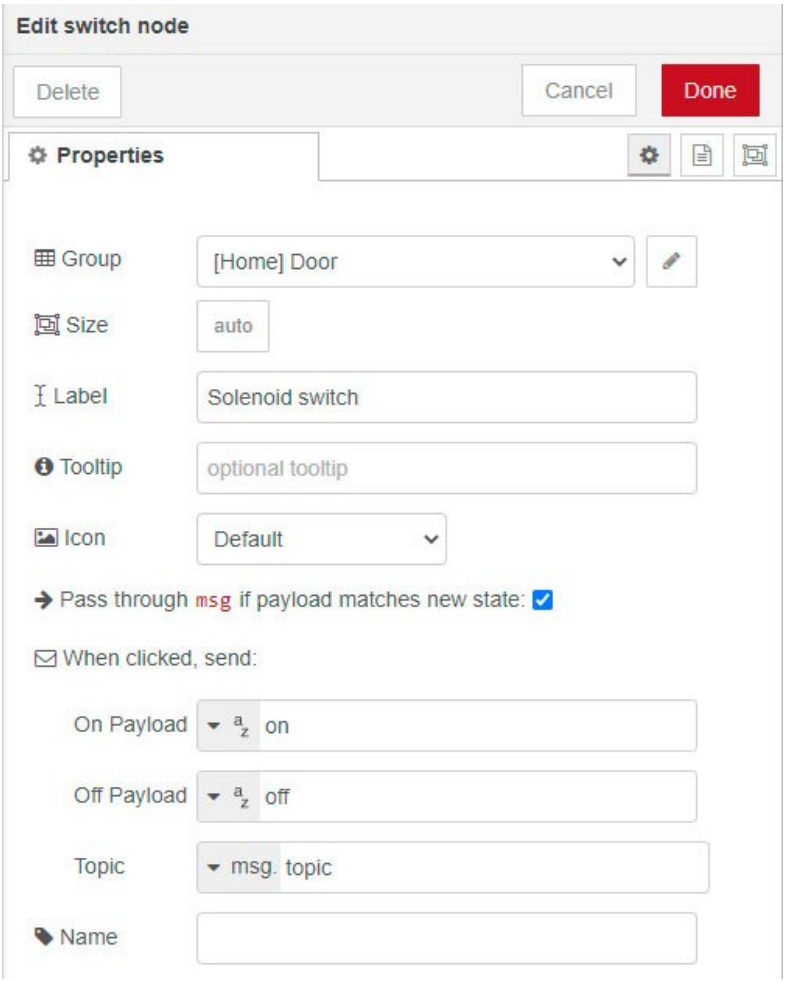
სურ. 104. კვანძების დაკავშირება

mqtt out კვანძის რედაქტირების ფანჯრის topic ველში ჩავწეროთ ის თემა (door/solenoid), რომელიც მითითებული გვაქვს სკეტჩში.



სურ. 105. mqtt out კვანძის რედაქტირება

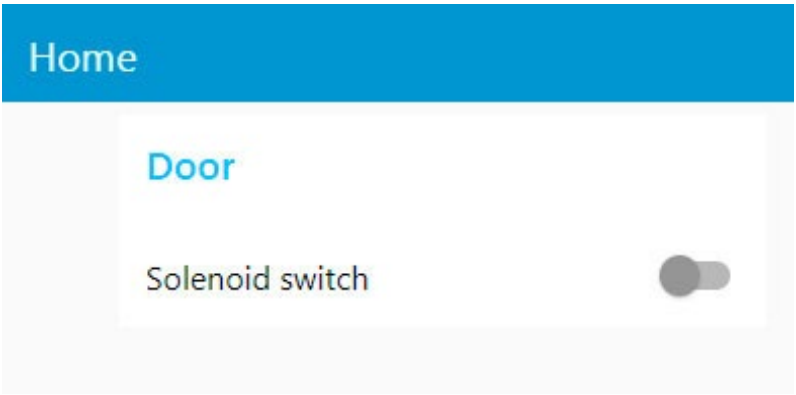
კარის გაღების ან დაკეტვის ბრძანების გადასაცემად ვიყენებთ switch კვანძს.



სურ. 106. switch კვანძის რედაქტირება

დავაჭიროთ Deploy ღილაკს და გადავიდეთ მისამართზე: http://raspberry-pi-ip-address:1880/ui.

ბრაუზერის ფანჯარაში მივიღებთ შემდეგ სურათს:

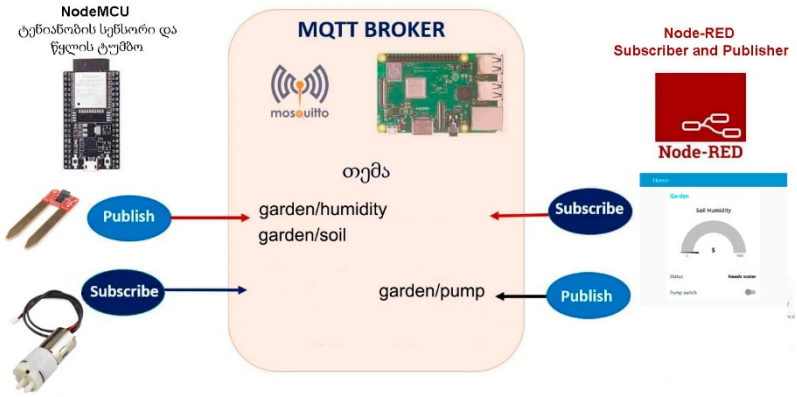


სურ. 107. კარის გაღება/დაკეტვის ვებ ინტერფეისი

საირიგაციო სისტემა MQTT პროტოკოლის გამოყენებით

MQTT პროტოკოლის გამოყენებით მარტივი IoT სარწყავი სისტემის შესაქმნელად საჭიროა ნიადაგის ტენიანობის სენსორი და რელეს მოდული. რელეს საშუალებით ხორციელდება წყლის ტუმბოს მართვა.

ქვემოთ მოცემული გამოსახულება (სურ. 108) გამოხატავს განსახილველი ამოცანის არსს.

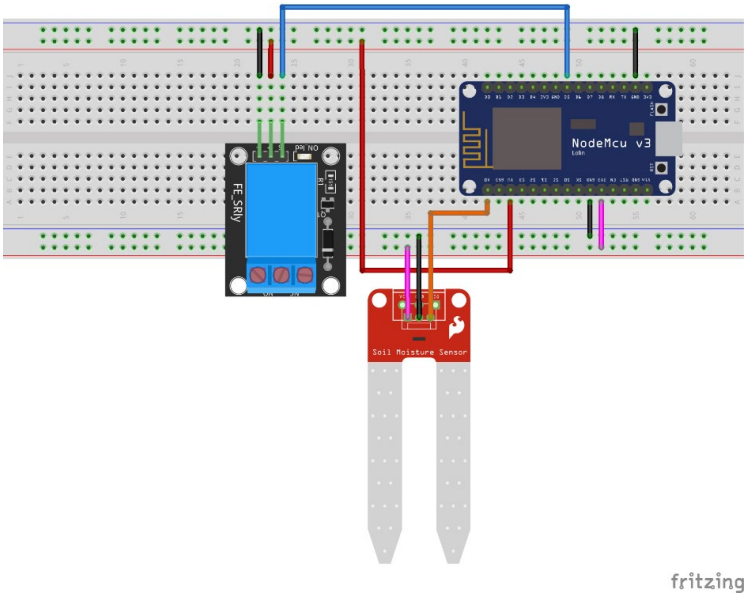


სურ. 108. სენსორებიდან და Node-RED-დან მიღებული მონაცემების გამოქვეყნება MQTT ბროკერის მეშვეობით

ავაგოთ სქემა სურ.109-ზე მოცემული ნახაზის მიხედვით.

რელეს მოდულის შესასვლელი გამომყვანი მივუერთოთ NodeMCU-ს D5 საკონტაქტო გამომყვანს. როგორც წესი, ნიადაგის ტენიანობის სენსორს სამი გამომყვანი აქვს: VCC, GND და A0.

A0 წარმოადგენს ნიადაგის ტენიანობის სენსორის ანალოგურ გამოსასვლელს, რომელზეც ძაბვა იცვლება ნიადაგში ტენიანობის პროპორციულად. ტენიანობის სენსორის გამომყვანი შევუერთოთ NodeMCU-ს A0 ანალოგურ შესასვლელს.



სურ.109 . მარტივი სარწყავი სისტემის სქემა

ქვემოთ მოცემულია პროგრამა, რომლის საშუალებითაც ხდება ნიადაგის ტენიანობის მონაცემების გადაცემა MQTT სერვერზე და ტუმბოს ჩართვა/გამორთვის ბრძანების მიღება.

```
//MQTT კლიენტის პროტოკოლის ბიბლიოთეკა. ეს ბიბლიო-
თეკა საშუალებას იძლევა გამოიწეროს და გამოაქვეყნოს
შეტყობინებები
#include <PubSubClient.h>
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან
დაკავშირებისთვის
#include <ESP8266WiFi.h>
//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
```

```
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
// Mosquitto MQTT Broker-ის ip მისამართი, იგივეა რაც
Raspberry pi-ს ip მისამართი
#define MQTT_HOST "your  mosquitto ip address"
// Mosquitto MQTT Broker-ის  პორტის ნომერი
#define MQTT_PORT 1883
// Mosquitto MQTT Broker-ის  მომხმარებლის სახელი
#define MQTT_USERNAME "your mosquitto username"
// Mosquitto MQTT Broker-ის  მომხმარებლის პაროლი
#define MQTT_PASSWORD "your mosquitto password"
//MQTT კლიენტის ID, რომელიც გამოიყენება სერვერთან
დასაკავშირებლად
#define MQTT_CLIENT_ID "NodeMCUClient"
// საკონტაქტო გამომყვანი, რომელიც მიერთებულია
რელეს მოდულთან
#define pump_pin D5
//კლასის ობიექტების შექმნა
WiFiClient NodeMCUHome;
PubSubClient client(NodeMCUHome);
// ტაიმერის დამხმარე ცვლადები
long now = millis();
long lastMeasure = 0;
void setup_wifi() {
  Serial.begin(115200);
  //უსადენო ქსელთან დაკავშირება
  WiFi.begin(ssid, wifiPassword);
  Serial.println("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
```

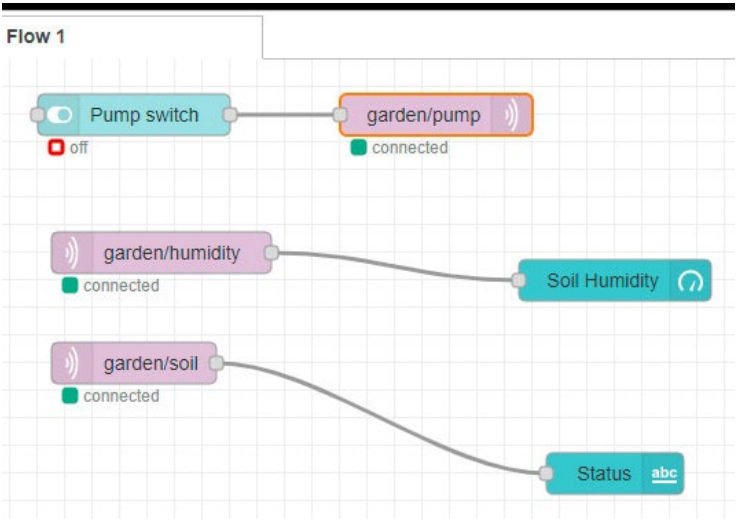
```
  Serial.print("Connected to WiFi network with IP
Address: ");
  Serial.println(WiFi.localIP());
}
//ეს ფუნქცია სრულდება მაშინ, როდესაც ქსელში არსე-
ბული რომელიმე მოწყობილობა აქვეყნებს შეტყობინებას
თემით, რომელიც NodeMCU-ს აქვს გამოწერილი
void callback(String topic, byte* message, unsigned
int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();
   //თუ მიღებული თემა არის garden/pump, მაშინ
ვამოწმებთ თემის შეტყობინებას, არის ის on თუ off, ამის
შემდეგ ხდება გადაწყვეტილების მიღება ტუმბოს ჩართვის ან
გამორთვის თაობაზე
  if (topic == "garden/pump") {
    Serial.print("Changing pump state to ");
    if (messageTemp == "on") {
      digitalWrite(pump_pin, HIGH);  //
      Serial.print("On");
    } else if (messageTemp == "off") {
      digitalWrite(pump_pin, LOW);
      Serial.print("Off");
    }
  }
```

```
  Serial.println();
}
// NodeMCU-ს ხელახალი დაკავშირება MQTT ბროკერთან
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(MQTT_CLIENT_ID, MQTT_USERNAME,
MQTT_PASSWORD)) {
      Serial.println("connected");
      //თემის გამოწერა
      client.subscribe("garden/pump");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
void setup() {
  pinMode(pump_pin, OUTPUT);
  digitalWrite(pump_pin, 0);
  Serial.begin(115200);
  setup_wifi();
  //ბროკერთან დასაკავშირებლად უნდა გამოვიძახოთ
მეთოდი setServer, რომელსაც გადავცემთ ბროკერის
მისამართს და პორტს.
  client.setServer(MQTT_HOST, MQTT_PORT);
  //ეს ფუნქცია სრულდება, როდესაც მოდის შეტყობინება
MQTT ბროკერისგან
  client.setCallback(callback);
}
```

```
void loop() {
  char msg[10];
  char msgtext[25];
  String themsg;
  //MQTT ბროკერთან კავშირის შემოწმება
  if (!client.connected()) {
    reconnect();
  }
  //MQTT ბროკერიდან შეტყობინების არსებობის შემოწმება
  if (!client.loop())
    client.connect(MQTT_CLIENT_ID);
  now = millis();
  //აქვეყნებს ნიადაგის ტენიანობის მნიშვნელობას ყოველ
30 წამში ერთხელ
  if (now - lastMeasure > 30000) {
    lastMeasure = now;
    //ტენიანობის მნიშვნელობის წაკითხვა A0 ანალოგური
შესასვლელიდან
    int soil_moisture = analogRead(A0);
    Serial.print("analog value: ");
    Serial.println(soil_moisture);
    //ტენიანობის მნიშვნელობის შემოწმება და შესაბამისი
შეტყობინების გამოტანა
    if ((soil_moisture > 300) && (soil_moisture <
700)) {
      Serial.println("Humid soil");
      sprintf(msgtext, "Humid soil");
    } else if ((soil_moisture > 700) &&
(soil_moisture < 1024)) {
      Serial.println("Moist Soil");
      sprintf(msgtext, "Moist Soil");
    } else if (soil_moisture < 300) {
      Serial.println("Needs water");
```

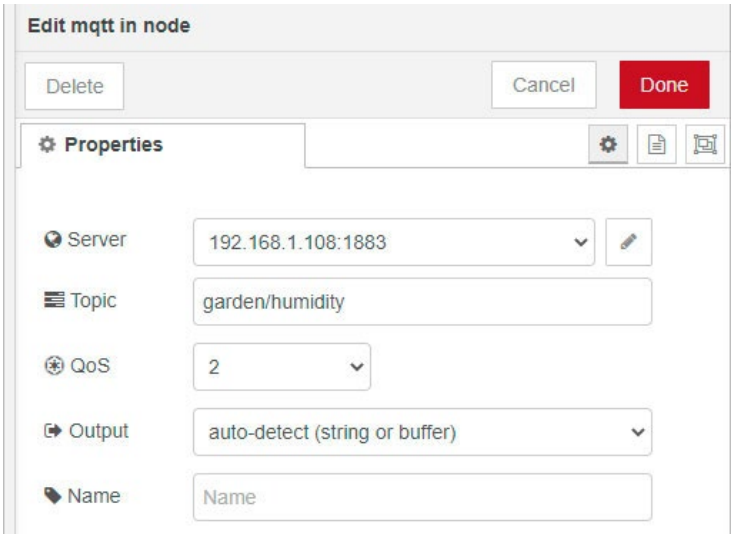
```
      sprintf(msgtext, "Needs water");
    } else {
      sprintf(msgtext, "Sensor Problem");
    }
    //ტენიანობის მნიშვნელობის ჩაწერა char ტიპის msg მასივში
    sprintf(msg, "%i", soil_moisture);
    //შეტყობინებების გამოქვეყნება შესაბამის თემებში
    client.publish("garden/humidity", msg);
    client.publish("garden/soil", msgtext);
  }
}
```

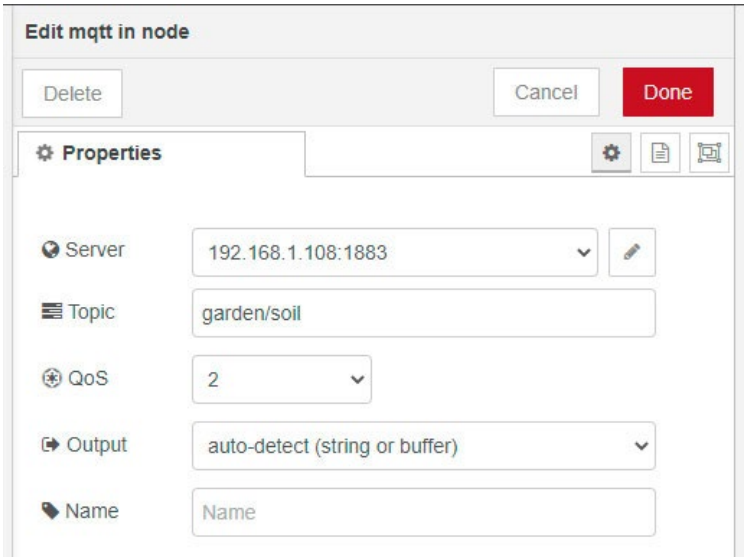
Node-RED-ის გარემოში მოვათავსოთ ორი mqtt in, ერთი mqtt out, ერთი switch და ორი gauge ტიპის კვანძები. დავაკავშიროთ ისინი ერთმანეთთან ისე, როგორც ეს სურ.110-ზეა მოცემული.

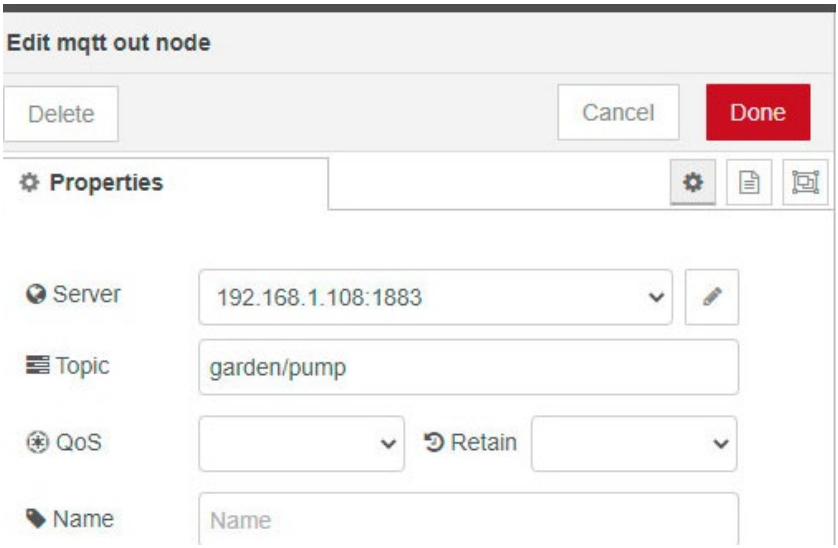


სურ. 110. კვანძების დაკავშირება

სურ.111-ზე ასახულია mqtt in და mqtt out კვანძების რედაქტირება.



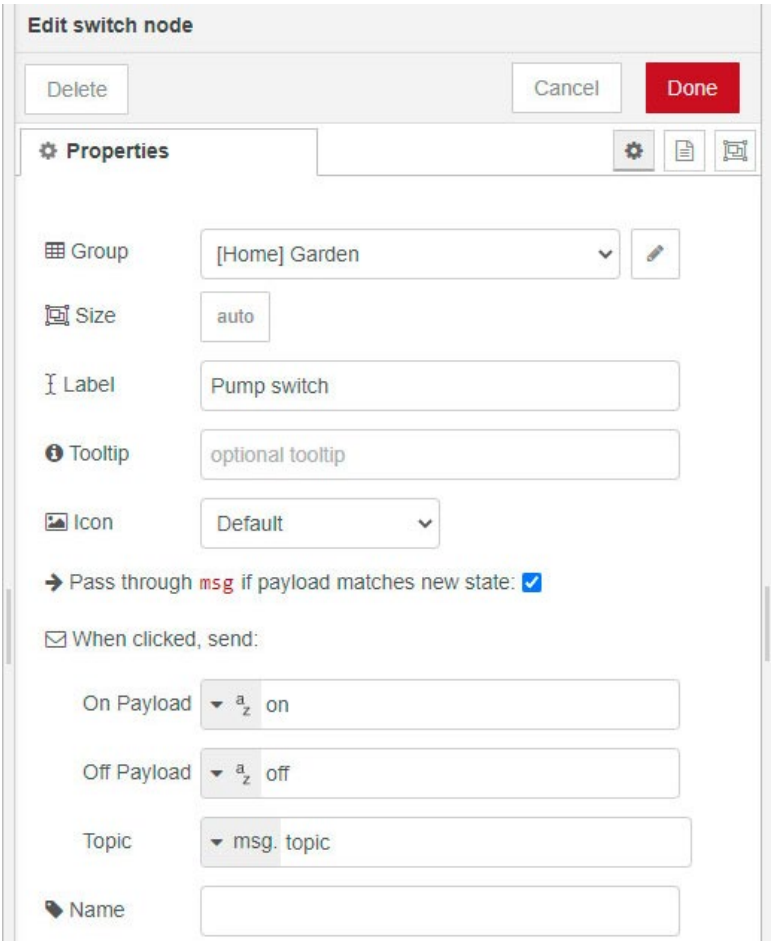




სურ.111. mqtt in და mqtt out კვანძების რედაქტირების პროცესი

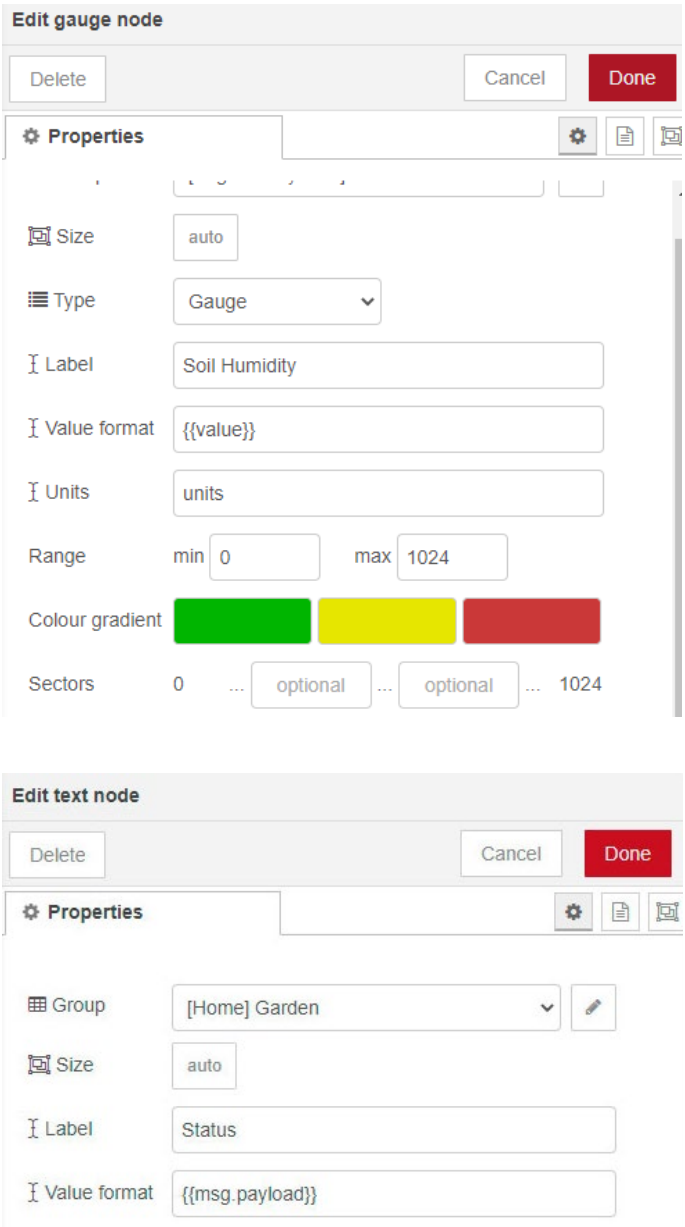
Node-RED-ის გარემოში ტუმბოს სამართავად გამოიყენება switch კვანძი.

სურ. 112-ზე ასახულია switch კვანძის რედაქტირების პროცესი.



სურ. 112. switch კვანძის რედაქტირება

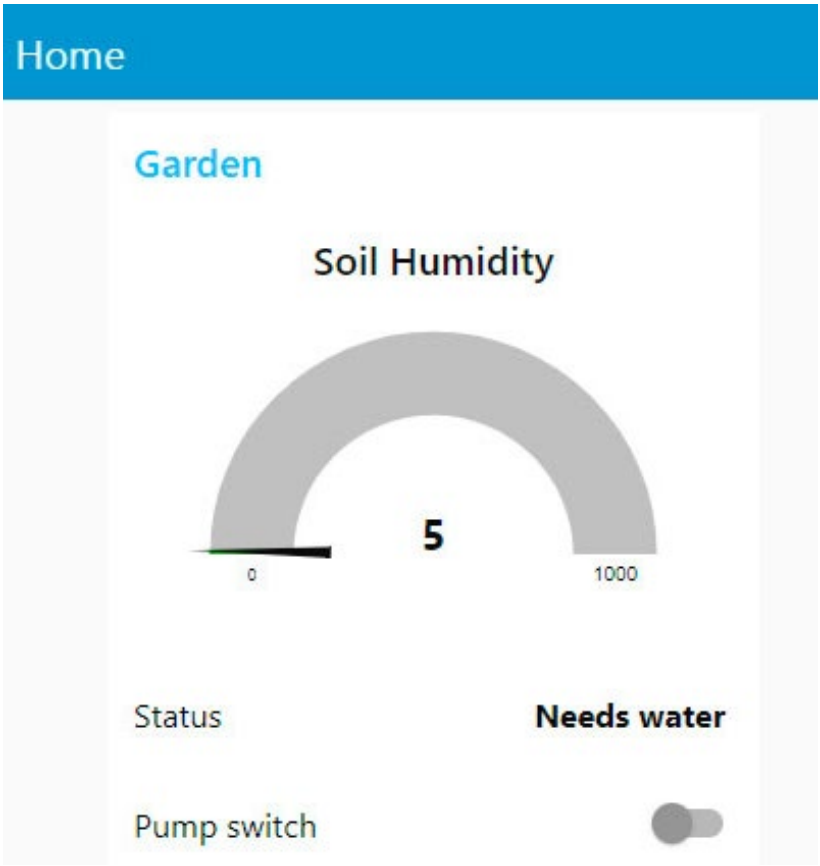
gauge ტიპის კვანძები გამოიყენება ნიადაგის ტენიანობის შესახებ ინფორმაციის გამოსატანად. სურ. 113-ზე ნაჩვენებია gauge ტიპის კვანძების რედაქტირება.



სურ. 113. gauge ტიპის კვანძების რედაქტირება

დავაჭიროთ Deploy ღილაკს და გადავიდეთ მისამართზე: http://raspberry-pi-ip-address:1880/ui

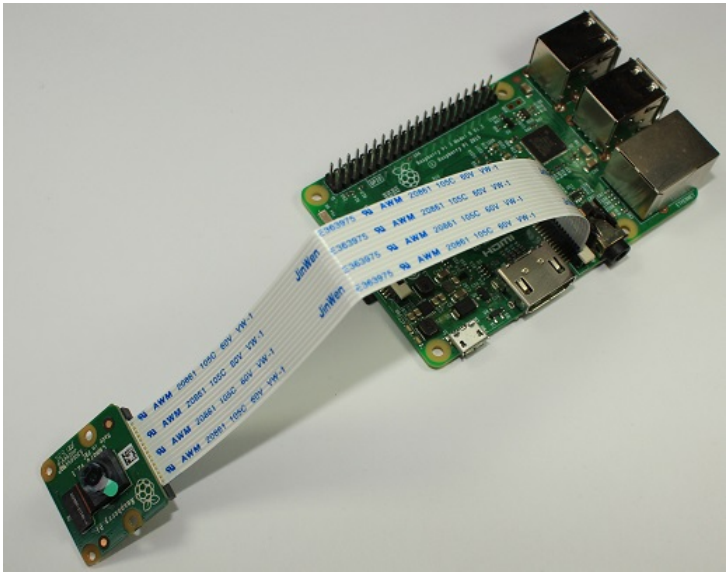
ბრაუზერის ფანჯარაში მივიღებთ შემდეგ სურათს:



სურ. 114. მარტივი საირიგაციო სისტემის ვებ ინტერფეისი

Node-RED და Raspberry Pi კამერა

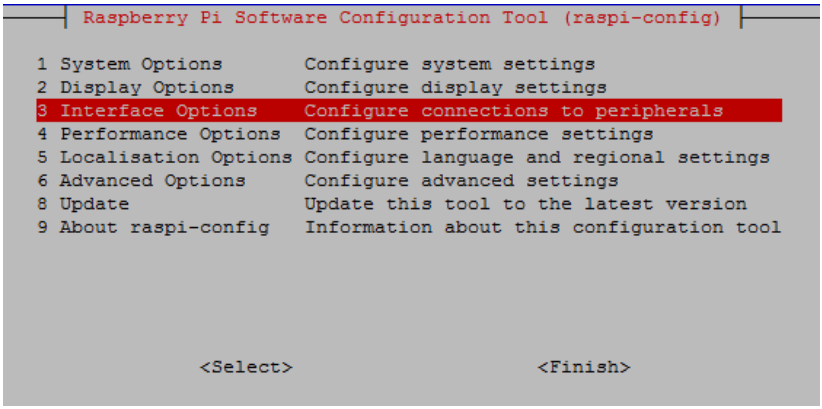
განვიხილოთ Node-RED-ისა და Raspberry Pi კამერას გამოყენებით ფოტოსურათის გადაღების მაგალითი. შევუერთოთ Raspberry Pi კამერის მოდული Raspberry Pi-ს, ისე როგორც სურ. 115-ზეა ნაჩვენები. შეერთების პროცესში Raspberry pi კვებიდან გამორთული უნდა იყოს. კამერა ისე უნდა იყოს მიერთებული Raspberry pi-სთან, რომ კამერის შლეიფის წარწერებიანი მხარე ზემოთ მოექცეს.



სურ. 115. Raspberry Pi კამერის მიერთება

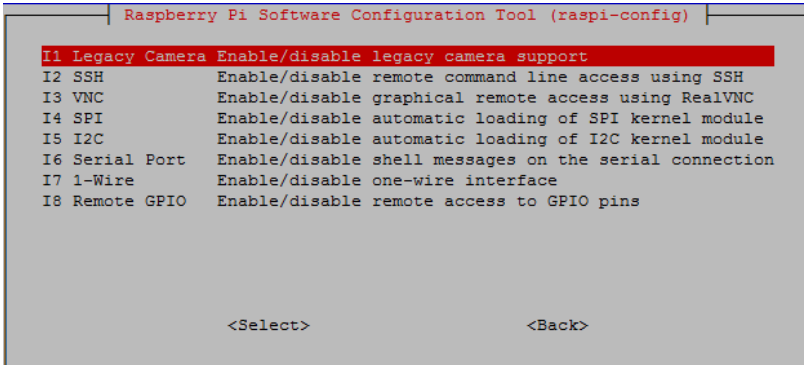
კამერის მხარდაჭერის ჩასართავად, ტერმინალის ფანჯარაში ავკრიფოთ შემდეგი ბრძანება: *sudo raspi-config*

ბრძანების აკრეფის შემდეგ ეკრანზე გამოჩნდება Raspberry Pi-ს კონფიგურაციის ფანჯარა. ავირჩიოთ ინტერფეისის პარამეტრები (სურ. 116):



სურ. 116. Raspberry Pi-ს ინტერფეისის პარამეტრები

ამ პროცედურის შემდეგ გამოჩნდება ფანჯარა, ვირჩევთ I1 Legacy Camera Enable/disable legacy camera support პარამეტრს (სურ. 117).



სურ. 117. I1 Legacy Camera Enable/disable legacy camera support პარამეტრის არჩევა

კონფიგურაციის პროცესის დასრულების შემდეგ გამოვიდეთ კონფიგურაციის პროგრამიდან და გადავტვირთოთ Raspberry pi.

დავაყენოთ Raspberry Pi კამერის კომპონენტი node-RED გარემოსთვის. ამისათვის ტერმინალის ფანჯარაში შევიტანოთ შემდეგი ბრძანებები:

cd ~/.node-red

sudo npm install node-red-contrib-camerapi

sudo reboot

გადაღებული ფოტოს შესანახად დაგვჭირდება საქაღალდე, მის ადგილმდებარეობას ვუთითებთ settings.js ფაილში. ტერმინალის ფანჯარაში ავკრიფოთ

sudo nano ~/.node-red/settings.js

ფაილში Ctrl+w ბრძანებით ვიპოვოთ httpStatic (სურ. 118). სტრიქონს წავუშალოთ კომენტარის სიმბოლო, მითითებული საქაღალდის მისამართის ნაცვლად ჩავწეროთ /home/iot/Pictures/, სადაც iot აღნიშნავს Raspberry pi-ს მომხმარებლის სახელს.

```
//    next();
//},

/** When httpAdminRoot is used to move the UI to a different root path, the
 * following property can be used to identify a directory of static content
 * that should be served at http://localhost:1880/.
 * When httpStaticRoot is set differently to httpAdminRoot, there is no nee
 * to move httpAdminRoot
 */
httpStatic: '/home/iot/Pictures/', //single static source
/* OR multiple static sources can be created using an array of objects... *
//httpStatic: [
//    {path: '/home/nol/pics/',    root: "/img/"},
//    {path: '/home/nol/reports/', root: "/doc/"},
//],
```

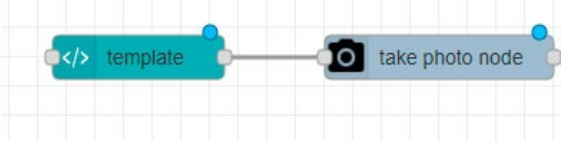
სურ.118. settings.js ფაილის რედაქტირება

რედაქტირების დასრულების შემდეგ, დავაჭიროთ Ctrl+x-ს, შემდეგ y-ს და enter-ს.

გადავიდეთ NodeRED-ის გარემოში. კვანძების ჩამონათვალიდან ავირჩიოთ camerapi takephoto კვანძი (სურ. 119).

სურ. 119. camerapi-takephoto კვანძი

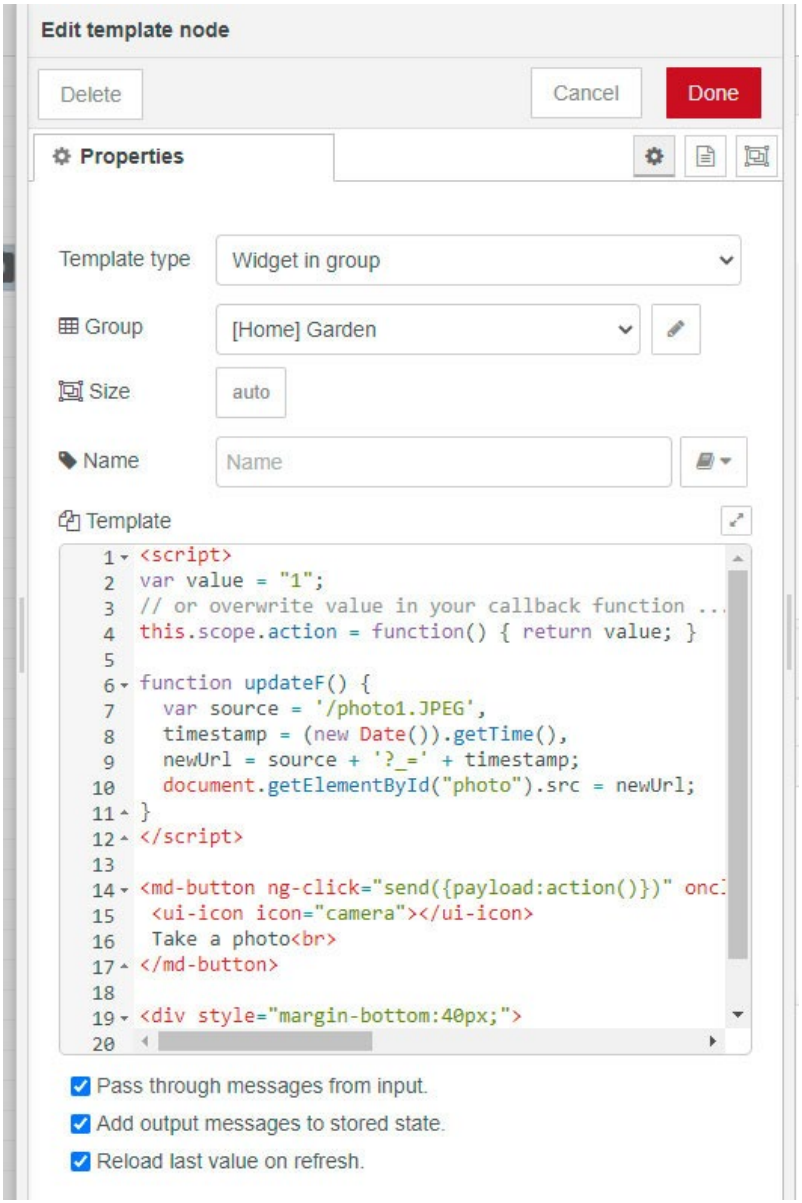
Node-RED-ის გარემოში გადმოვიტანოთ camerapi-takephoto და template კვანძები. შევაერთოთ ეს კვანძები ერთმანეთთან (სურ. 120).



სურ. 120. კვანძების შეერთება

Template კვანძის Template ველში ჩავწეროთ შემდეგი კოდი (სურ.121):

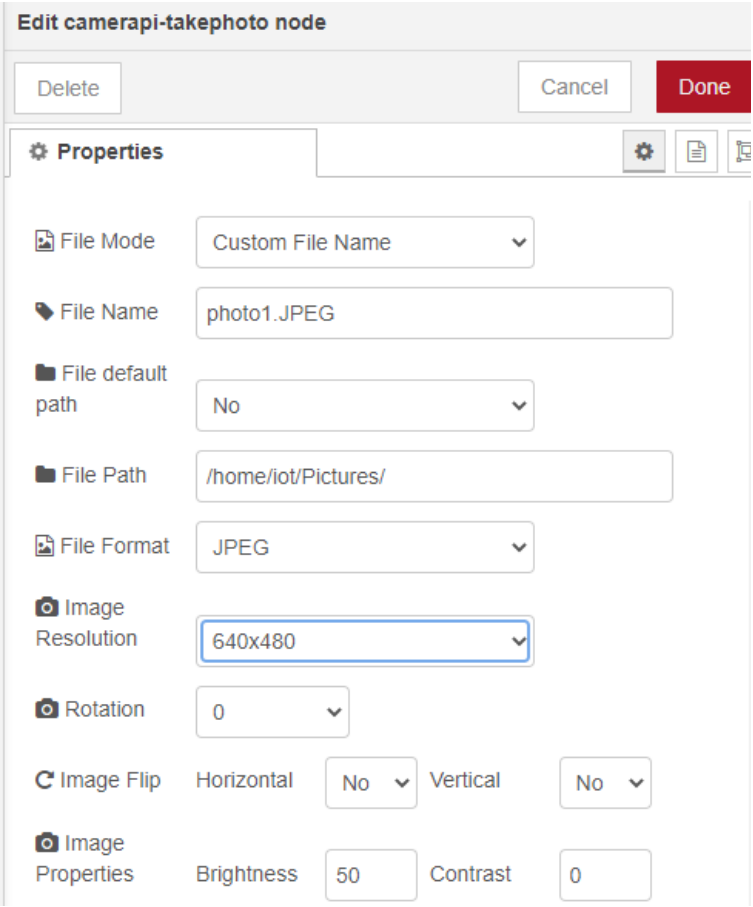
```
<script>
var value = "1";
this.scope.action = function() { return value; }

function updateF() {
  var source = '/photo1.JPEG',
  timestamp = (new Date()).getTime(),
  newUrl = source + '?_=' + timestamp;
  document.getElementById("photo").src = newUrl;
}
</script>

<md-button ng-click="send({payload:action()})"
onclick="setTimeout(updateF, 2500);"
style="padding:40px; margin-bottom: 40px;" >
 <ui-icon icon="camera"></ui-icon>
 Take a photo<br>
</md-button>

<div style="margin-bottom:40px;">
 <img src="/photo1.JPEG" id="photo" width="100%"
height="100%">
</div>
```



სურ. 121. Template კვანძის რედაქტირება

JavaScript კოდის ფუნქციაა Node-RED გვერდის განახლება ფოტოს ყოველი გადაღების შემდეგ.

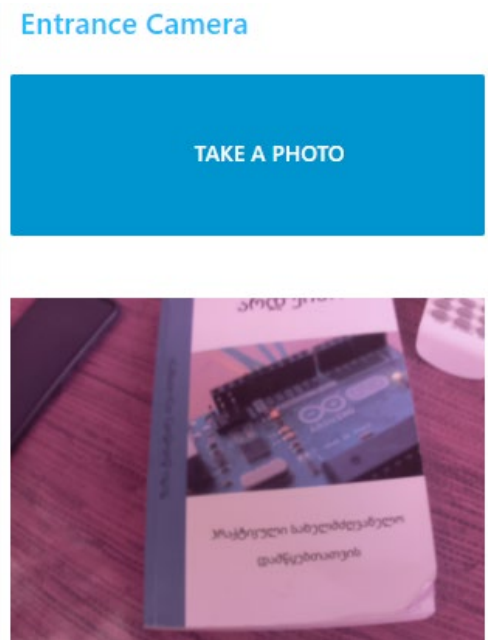
სურ. 122-ზე მოცემულია camerapi-takephoto კვანძის რედაქტირების ფანჯარა.



სურ. 122. camerapi-takephoto კვანძის რედაქტირება

camerapi-takephoto კვანძის File Path ველში სწორად უნდა იყოს მითითებული ფოტოსურათის შესანახად არჩეული საქაღალდის მისამართი (/home/iot/Pictures/). ფოტოს გადაღების შემდეგ, ფოტო შეინახება photo1.JPEG სახელით, ჩვენს მიერ მითითებულ საქაღალდეში. ყოველი ახალი ფოტოს გადაღების შემდეგ Node-RED ფოტოს ინახავს იმავე საქაღალდეში, იგივე სახელით, რაც ნიშნავს, რომ ძველი ფოტოს ჩანაცვლება ხდება ახალი ფოტოთი.

დავაჭიროთ Deploy ღილაკს და გადავიდეთ მისამართზე: http://raspberry-pi-ip-address:1880/ui. ბრაუზერის ფანჯარაში მივიღებთ შემდეგ სურათს:

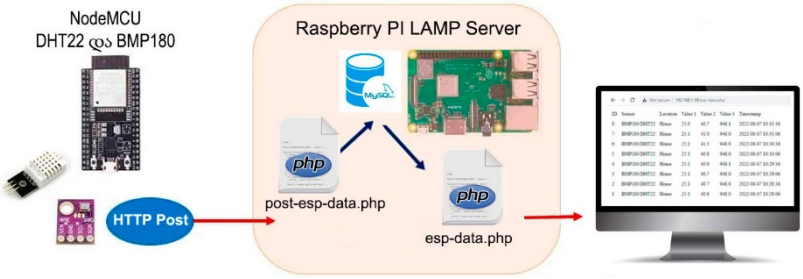


სურ. 123. კამერით გადაღებული ფოტო

სენსორის მონაცემების გამოქვეყნება Raspberry Pi LAMP სერვერზე

ამოცანის მიზანია სენსორებიდან მიღებული მონაცემების გაგზავნა სერვერზე, HTTP პროტოკოლის საშუალებით; შემდეგ ამ მონაცემების შენახვა მონაცემთა ბაზაში და მათი გამოქვეყნება ვებ საიტზე.

ვქმნით NodeMCU კლიენტს, რომელიც HTTP პროტოკოლის საშუალებით დაუკავშირდება Raspberry Pi LAMP (Linux, Apache, MySQL, PHP) სერვერს. Raspberry Pi-ზე გვექნება PHP სკრიპტი, რომლის საშუალებითაც NodeMCU-დან მიღებული სენსორების მონაცემებს შევიტანთ MYSQL მონაცემთა ბაზაში. გარდა ამისა, დაგვჭირდება კიდევ ერთი PHP სკრიპტი, რომლის საშუალებითაც მოხდება მონაცემთა ბაზაში შენახული მონაცემების ვებ საიტზე გამოტანა. ჩვენს მაგალითში გამოვიყენებთ NodeMCU დაფაზე დაერთებულ BMP180 და DHT22 სენსორებს.



სურ.124. სენსორების მონაცემების გამოქვეყნება Raspberry Pi LAMP სერვერზე

პროექტის განსახორციელებლად, გამოვიყენებთ შემდეგ ტექნოლოგიებს:

- LAMP სერვერი
  - NodeMCU
  - PHP სკრიპტი, NodeMCU-დან სენსორების მონაცემების MYSQL მონაცემთა ბაზაში შესატანად.
- MySQL მონაცემთა ბაზა

სანამ დავიწყებთ LAMP სერვერის ინსტალაციას, შევასრულოთ შემდეგი ბრძანება:

sudo apt update && sudo apt upgrade -y

Apache2 ვებ სერვერის ინსტალაციისთვის შევასრულოთ შემდეგი ბრძანება: *sudo apt install apache2 -y*

თუ ინსტალაციისას პრობლემა შეიქმნა, ასეთ შემთხვევაში დაგვჭირდება სისტემის სრული განახლება:

sudo apt full-upgrade

sudo apt -y dist-upgrade

იმისთვის, რომ შევამოწმოთ დაინსტალირდა თუ არა apache2, შევასრულოთ შემდეგი ბრძანებები:

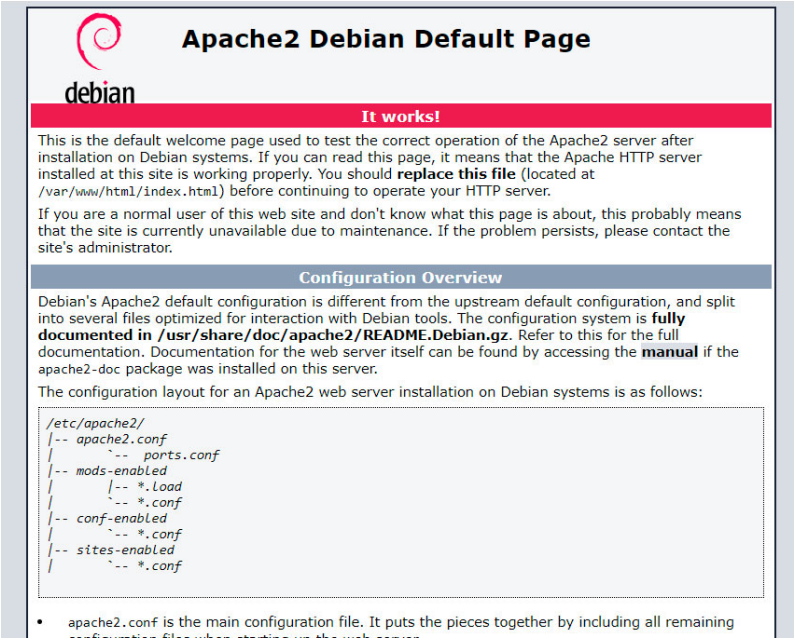
cd /var/www/html

ls -al

```
iot@raspberrypi:/var/www/html $ ls -al
total 20
drwxr-xr-x 2 root root  4096 Aug  6 15:05 .
drwxr-xr-x 3 root root  4096 Aug  6 15:05 ..
-rw-r--r-- 1 root root 10701 Aug  6 15:05 index.html
iot@raspberrypi:/var/www/html $
```

სურ. 125. Apache2-ის html საქაღალდე

Apache2-ის html საქაღალდეში უნდა გვქონდეს index.html ფაილი. იმისათვის, რომ ჩვენს ბრაუზერში გავხსნათ index.html გვერდი, ბრაუზერის სამისამართო ველში ავკრიფოთ Raspberry pi-ს ip მისამართი. ეკრანზე მივიღებთ შემდეგ გამოსახულებას (სურ. 126).



სურ. 126. Apache2 ვებ სერვერის საწყისი გვერდი

Raspberry Pi-ზე PHP-ის ინსტალაციისთვის, შევასრულოთ შემდეგი ბრძანება: *sudo apt install php -y*

PHP-ის ინსტალაციის დასრულების შემდეგ წავშალოთ index.html ფაილი და შევქმნათ index.php ფაილი. ამისათვის შევასრულოთ შემდეგი ბრძანებები:

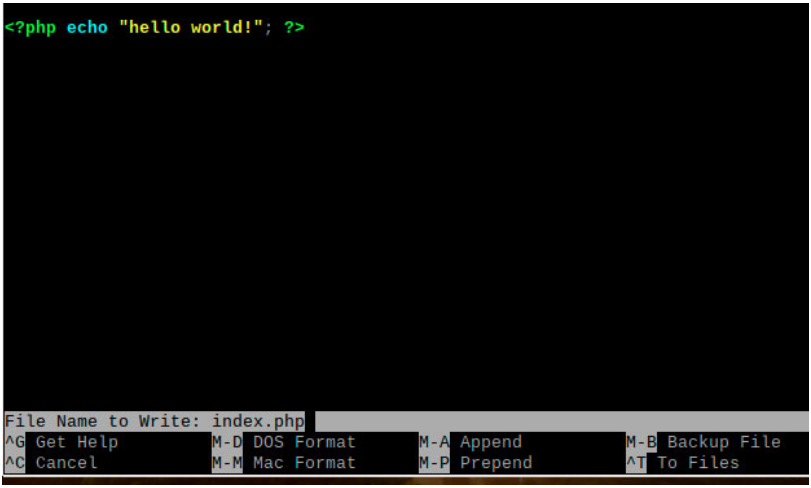
cd /var/www/html

sudo rm index.html

sudo nano index.php

index.php ფაილში ჩავამატოთ სტრიქონი:

<?php echo "hello world"; ?>



სურ. 127. index.php ფაილი

index.php ფაილის შესანახად, დავაჭიროთ Ctrl+x და შემდეგ y კლავიშს. გადავტვირთოთ Apache2 ვებ სერვერი ბრძანებით: *sudo service apache2 restart*

იმისათვის, რომ შევამოწმოთ მუშაობს თუ არა Apache2 ვებ სერვერზე PHP სკრიპტის მხარდაჭერა, ბრაუზერის

სამისამართო ველში ავკრიფოთ Raspberry pi-ს ip მისამართი. ვებ გვერდზე უნდა გამოჩნდეს index.php ფაილში ჩვენს მიერ შენახული ტექსტი - hello world!. ამის შემდეგ, წავშალოთ ჩვენს მიერ შექმნილი index.php ფაილი:

cd /var/www/html

sudo rm index.php

MySQL სერვერისა (MariaDB Server) და PHP-MySQL პაკეტების ინსტალაციისთვის შევასრულოთ შემდეგი ბრძანებები:

sudo apt install mariadb-server php-mysql -y

sudo service apache2 restart

MySQL (MariaDB Server) მონაცემთა ბაზის ინსტალაციის შემდეგ, რეკომენდებულია შევასრულოთ ქვემოთ მოცემული ბრძანება MySQL ბაზის უსაფრთხოებისთვის:

sudo mysql_secure_installation

ბრძანების შესრულების შემდეგ ტერმინალის ფანჯარაში მივიღებთ (სურ.128):

```
Processing triggers for php7.4-cli (7.4.30-1+deb11u1) ...
iot@raspberrypi:/var/www/html $ sudo service apache2 restart
iot@raspberrypi:/var/www/html $ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
```

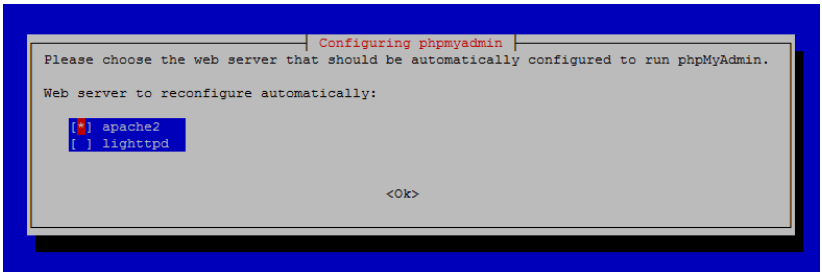
სურ. 128. MySQL სერვერის უსაფრთხოების პაკეტის ინსტალაცია

ქვემოთ აღწერილია MySQL სერვერის უსაფრთხოების პაკეტის ინსტალაციის პროცედურა:

- სისტემა გვეკითხება (სურ.128) მიმდინარე root პაროლს;
- შევიტანოთ Raspberry Pi-ს მომხმარებლის პაროლი;
- დავაჭიროთ enter-ს;
- დავაჭიროთ Y კლავიშს;
- სისტემა გვეკითხება, გვინდა თუ არა root პაროლის შეცვლა, ვაჭერთ Y-ს და შემდეგ enter-ს;
- შევიტანოთ ახალი პაროლი და დავაჭიროთ enter-ს;
- დავაჭიროთ Y-ს კითხვაზე Remove anonymous users
- დავაჭიროთ Y-ს კითხვაზე Disallow root login remotely
- დავაჭიროთ Y-ს კითხვაზე Remove test database and access to it
- დავაჭიროთ Y-ს კითხვაზე Reload privilege tables now

ინსტალაციის დასრულების შემდეგ ტერმინალის ფანჯარაში დავინახავთ შეტყობინებას: “Thanks for using MariaDB!”.

MySQL მონაცემთა ბაზის ადმინისტრირებისთვის დავაინსტალიროთ phpMyAdmin პროგრამული უზრუნ-ველყოფა. შევასრულოთ შემდეგი ბრძანება: *sudo apt install phpmyadmin -y.* ტერმინალის ფანჯარაში გამოჩნდება შეკითხვები phpMyAdmin-ის კონფიგურირებასთან დაკავ-შირებით. ავირჩიოთ apache2 და დავაჭიროთ Ok ღილაკს.



სურ.129. phpMyAdmin-ის კონფიგურირება

კითხვაზე Configure database for phpmyadmin with dbconfig-common? პასუხად დავაჭიროთ Yes-ს; შემდეგ ფანჯარაში შევიტანოთ პაროლი phpmyadmin-თვის, დავაჭიროთ enter-ს. PHP MySQLi გაფართოების ჩასართავად, შევასრულოთ შემდეგი ბრძანებები:

sudo phpenmod mysqli

sudo service apache2 restart

PhpMyAdmin-ის ინსტალაციის დასასრულებლად უნდა შეიქმნას ახალი მომხმარებელი, ამისათვის შევასრულოთ შემდეგი ბრძანება: *sudo mysql -u root -p.* ტერმინალის ფანჯარაში გამოჩნდება MySQL (MariaDB) მონიტორი (სურ.130).

```
iot@raspberrypi:/var/www/html $ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 51
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

სურ.130. MySQL (MariaDB) მონიტორი

MySQL (MariaDB) მონიტორში ჩავწეროთ შემდეგი ბრძანებები:

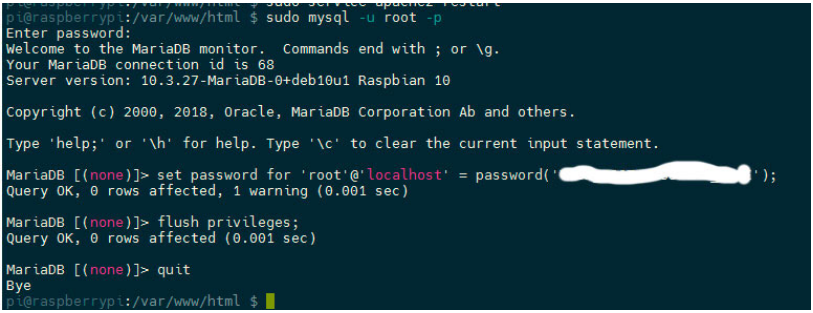
set password for 'root'@'localhost' =

password('YOUR_ROOT_PASSWORD_HERE');

flush privileges;

quit

ტექსტი 'YOUR_ROOT_PASSWORD_HERE' შევცვალოთ ჩვენთვის სასურველი პაროლით.



სურ.131. MySQL (MariaDB) მონიტორი

ამ პროცედურის დასრულების შემდეგ, მოვახდინოთ Apache2.conf ფაილის რედაქტირება ბრძანებით

sudo nano /etc/apache2/apache2.conf

ფაილის ბოლოში დავამატოთ სტრიქონი:

Include /etc/phpmyadmin/apache.conf

ფაილის შესანახად, დავაჭიროთ Ctrl+x და შემდეგ y კლავიშს.

გადავტვირთოთ Apache2 ვებ სერვერი ბრძანებით:

sudo service apache2 restart

ბრაუზერის სამისამართო სტრიქონში ავკრიფოთ http://raspberry-pi-ip-address/phpmyadmin

ბრაუზერის ფანჯარაში მივიღებთ შემდეგ სურათს:

სურ.132. phpMyAdmin-ის ავტორიზაციის გვერდი

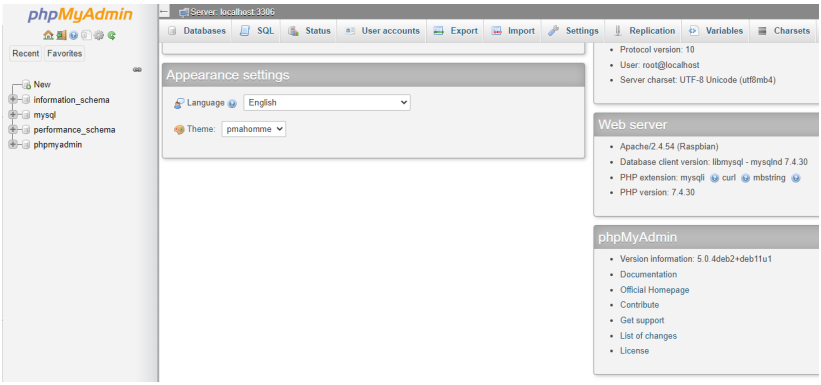
Username ველში ჩავწეროთ root, password ველში root-ის შეცვლილი პაროლი (*'YOUR_ROOT_PASSWORD_HERE'*). დავაჭიროთ Go ღილაკს. ეკრანზე გამოჩნდება phpMyAdmin-ის საწყისი გვერდი. ამით სრულდება Raspberry Pi-ზე LAMP სერვერისა და phpMyAdmin-ის ინსტალაცია.

იმისათვის, რომ ვმართოთ ჩვენი ვებ გვერდები, საჭიროა /var/www/html/ საქაღალდეზე, Raspberry Pi-ს

მომხმარებელს (ჩვენს შემთხვევაში, iot) გარკვეული უფლებები მივანიჭოთ. ტერმინალის ფანჯარაში შევასრულოთ შემდეგი ბრძანებები:

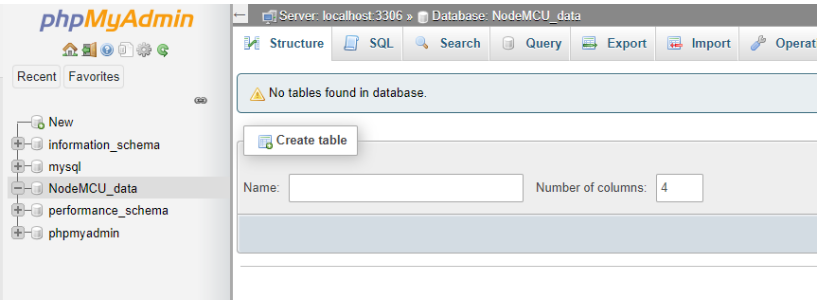
sudo chown -R iot:www-data /var/www/html/
sudo chmod -R 770 /var/www/html/

MySQL მონაცემთა ბაზის შესაქმნელად გადავიდეთ PhpMyAdmin-ის საწყისი გვერდის (სურ. 133) databases ჩანართში, Create database ველში ჩავწეროთ მონაცემთა ბაზის სახელი NodeMCU_data, დავაჭიროთ Create ღილაკს.



სურ.133. phpMyAdmin-ის საწყისი გვერდი

phpMyAdmin-ის საწყისი გვერდის მარცხენა მხარეს, ბაზების ჩამონათვალში გამოჩნდება ჩვენს მიერ შექმნილი NodeMCU_data მონაცემთა ბაზა (სურ. 134).

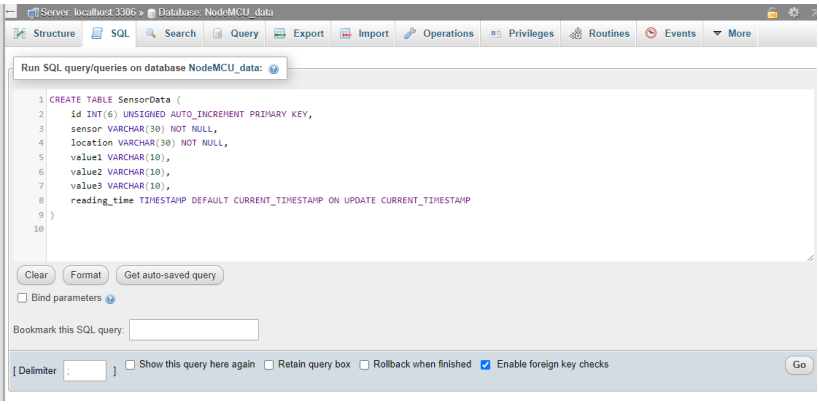


სურ.134. NodeMCU_data ბაზის შექმნა

მაუსის მარცხენა ღილაკზე დაჭერით ბაზების სიიდან ავირჩიოთ ჩვენს მიერ შექმნილი მონაცემთა ბაზა, დავაჭიროთ SQL ჩანართს, ჩავწეროთ მასში შემდეგი კოდი:

```
CREATE TABLE SensorData (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   sensor VARCHAR(30) NOT NULL,
   location VARCHAR(30) NOT NULL,
   temp VARCHAR(10),
  hum VARCHAR(10),
   pres VARCHAR(10),
   reading_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
)
```

დავაჭიროთ Go ღილაკს, შეიქმნება NodeMCU_data ბაზის ცხრილი SensorData.



სურ.135.SensorData ცხრილის შექმნა

დავწეროთ PHP სკრიპტი, რომელიც ახდენს NodeMCU-დან მიღებული ინფორმაციის შეტანას MySQL მონაცემთა ბაზაში. PHP სკრიპტის შესაქმნელად ავკრიფოთ შემდეგი ბრძანება: *sudo nano /var/www/html/post-esp-data.php*

post-esp-data.php ფაილში ჩავწეროთ შემდეგი კოდი:

```
<?php
$servername = "localhost";
 // ჩვენს მიერ შექმნილი მონაცემთა ბაზის სახელი
$dbname = "REPLACE_WITH_YOUR_DATABASE_NAME";
// მონაცემთა ბაზის მომხმარებლის სახელი
$username = "REPLACE_WITH_YOUR_USERNAME";
// მონაცემთა ბაზის მომხმარებლის პაროლი
$password = "REPLACE_WITH_YOUR_PASSWORD";

// api_key_value-ს მნიშვნელობა იგივე უნდა იყოს, რაც
NodeMCU-ში ატვირთულ კოდში.
$api_key_value = " testAPIkey";
```

```
$api_key= $sensor = $location = $value1 = $value2 =
$value3 = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $api_key = test_input($_POST["api_key"]);
    if($api_key == $api_key_value) {
        $sensor = test_input($_POST["sensor"]);
        $location = test_input($_POST["location"]);
        $value1 = test_input($_POST["temp"]);
        $value2 = test_input($_POST["hum"]);
        $value3 = test_input($_POST["pres"]);

        $conn = new mysqli($servername, $username,
$password, $dbname);
        // Check connection
        if ($conn->connect_error) {
            die("Connection failed: " . $conn-
>connect_error);
        }

        $sql = "INSERT INTO SensorData (sensor,
location, temp, hum, pres)
        VALUES ('" . $sensor . "', '" . $location .
"', '" . $value1 . "', '" . $value2 . "', '" .
$value3 . "')";

        if ($conn->query($sql) === TRUE) {
            echo "New record created successfully";
        }
        else {
            echo "Error: " . $sql . "<br>" . $conn-
>error;
        }
```

```
        $conn->close();
    }
    else {
        echo "Wrong API Key provided.";
    }

}
else {
    echo "No data posted with HTTP POST.";
}

function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
?>
```

ფაილის შესანახად, დავაჭიროთ Ctrl+x და შემდეგ y კლავიშს.

http://raspberry-pi-ip-address/post-esp-data.php მისამართზე შესვლისას ბრაუზერის ფანჯარაში გამოჩნდება: No data posted with HTTP POST.

დავწეროთ მეორე PHP სკრიპტი, რომელიც ახდენს MySQL მონაცემთა ბაზიდან მონაცემების გამოტანას ვებ საიტზე. PHP სკრიპტის ფაილის შესაქმნელად ავკრიფოთ შემდეგი ბრძანება: *sudo nano /var/www/html/esp-data.php*

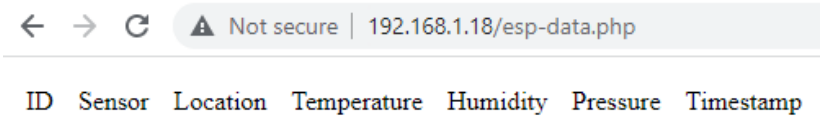
ჩავწეროთ *esp-data.php* ფაილში ქვემოთ მოცემული კოდი:

```
<!DOCTYPE html>
<html><body>
<?php

$servername = "localhost";

// ჩვენს მიერ შექმნილი მონაცემთა ბაზის სახელი
$dbname = "REPLACE_WITH_YOUR_DATABASE_NAME";
// მონაცემთა ბაზის მომხმარებლის სახელი
$username = "REPLACE_WITH_YOUR_USERNAME";
// მონაცემთა ბაზის მომხმარებლის პაროლი
$password = "REPLACE_WITH_YOUR_PASSWORD";

$conn = new mysqli($servername, $username, $password,
$dbname);
if ($conn->connect_error) {
    die("Connection failed: " . $conn-
>connect_error);
}

$sql = "SELECT id, sensor, location, temp, hum, pres,
reading_time FROM SensorData ORDER BY id DESC";

echo '<table cellspacing="5" cellpadding="5">
      <tr>
        <td>ID</td>
        <td>Sensor</td>
        <td>Location</td>
        <td>Temperature</td>
        <td>Humidity</td>
```

```
        <td>Pressure</td>
        <td>Timestamp</td>
      </tr>';

if ($result = $conn->query($sql)) {
    while ($row = $result->fetch_assoc()) {
        $row_id = $row["id"];
        $row_sensor = $row["sensor"];
        $row_location = $row["location"];
        $row_value1 = $row["temp"];
        $row_value2 = $row["hum"];
        $row_value3 = $row["pres"];
        $row_reading_time = $row["reading_time"];

        echo '<tr>
                <td>' . $row_id . '</td>
                <td>' . $row_sensor . '</td>
                <td>' . $row_location . '</td>
                <td>' . $row_value1 . '</td>
                <td>' . $row_value2 . '</td>
                <td>' . $row_value3 . '</td>
                <td>' . $row_reading_time . '</td>
              </tr>';
    }
    $result->free();
}

$conn->close();
?>
</table>
</body>
</html>
```

ფაილის შესანახად, დავაჭიროთ Ctrl+x და შემდეგ y კლავიშს.

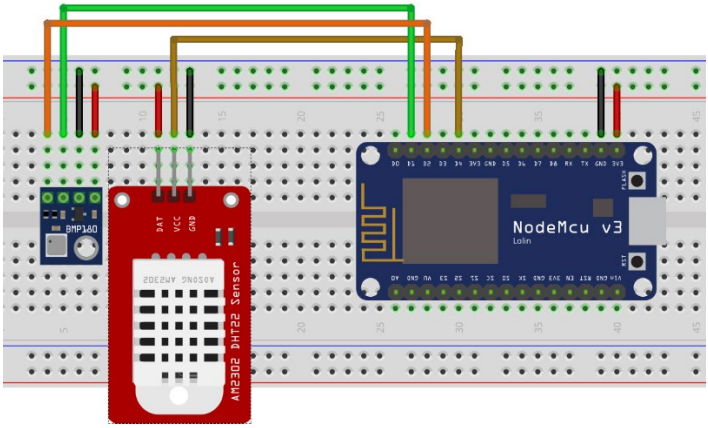
http://raspberry-pi-ip-address/esp-data.php მისამართზე შესვლისას ბრაუზერის ფანჯარაში გამოჩნდება (სურ.136):



სურ.136. NodeMCU_data მონაცემთა ბაზის ვებ ინტერფეისი

მონაცემთა ბაზის ცარიელი ცხრილის ვებ გვერდზე გამოტანა ნიშნავს იმას, რომ ყველაფერი მზად არის მონაცემების მისაღებად. ბაზისთვის მონაცემების გადასაცემად დაგვჭირდება NodeMCU და სენსორები.

ავაგოთ სურ. 137 -ზე მოცემული სქემა.



სურ.137. სენსორების დაერთება NodeMCU-ზე

ჩავწეროთ პროგრამის კოდი Arduino IDE-ში და ავტვირთოთ.

```
//ESP8266-ის ბიბლიოთეკა Wi-Fi ქსელთან დაკავშირებისთვის
#include <ESP8266WiFi.h>
//HTTP GET, POST და PUT ბიბლიოთეკა
#include <ESP8266HTTPClient.h>
//უსადენო ქსელის კლიენტის ბიბლიოთეკა
#include <WiFiClient.h>
//Adafruit BMP085/BMP180-ის ბიბლიოთეკა
#include <Adafruit_BMP085.h>
//DHT სენსორების ბიბლიოთეკა
#include "DHT.h"

#define DHTPIN D4
//წაუშალეთ კომენტარის ნიშანი თქვენს მიერ გამოყენებული სენსორის შესაბამისად
//#define DHTTYPE DHT11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
// DHT სენსორის ინიციალიზაცია
DHT myDHTSensor(DHTPIN, DHTTYPE);

//BMP180 სენსორის ინიციალიზაცია
Adafruit_BMP085 myBMPSensor;

//Wi-Fi ქსელის SSID
#define ssid "your_wifi_ssid"
//Wi-Fi ქსელის პაროლი
#define wifiPassword "your_wifi_password"
```

```
//ჩვენს მიერ შექმნილი PHP სკრიპტის მისამართი
"http://Your-Raspberry-Pi-IP-Address/post-esp-
data.php";
const char* serverName = "http://192.168.1.18/post-
esp-data.php";
//ცვლადები, სადაც ვინახავთ სენსორებიდან მიღებულ
მნიშვნელობებს
float temp;
float pres;
float hum;
float bmp_temp;
// apiKeyValue ცვლადის მნიშვნელობა შეიძლება იყოს
ნებისმიერი სტრიქონი, მხოლოდ უნდა ემთხვეოდეს esp-
data-post.php სკრიპტში არსებულ api_key_value
ცვლადის მნიშვნელობას. ეს პარამეტრი გამოიყენება
უსაფრთხოების მიზნით. მხოლოდ მათ შეუძლიათ
მონაცემების გამოქვეყნება ბაზაში, ვისთვისაც ცნობილია
აღნიშნული პარამეტრის მნიშვნელობა.

String apiKeyValue = "testAPIkey";
String sensorName = "BMP180/DHT22";
String sensorLocation = "Home";

unsigned long lastTime = 0;
// მივანიჭოთ ტაიმერს დროის რაიმე მნიშვნელობა,
რომლის გასვლის შემდეგაც მოხდება სენსორების
ჩვენებების აღება; ჩვენს შემთხვევაში, ეს დრო 30 წამია
unsigned long timerDelay = 30000;

void check_sensors() {
  // DHT სენსორიდან მონაცემების წაკითხვა
  hum = myDHTSensor.readHumidity();
  temp = myDHTSensor.readTemperature();
```

```
  // BMP180 სენსორიდან მონაცემების წაკითხვა
  pres = myBMPSensor.readPressure() / 100.0;
  bmp_temp = myBMPSensor.readTemperature();
}

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, wifiPassword);
  Serial.println("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP
Address: ");
  Serial.println(WiFi.localIP());
  if (!myBMPSensor.begin()) {
    Serial.println("Could not find a valid BMP085
sensor, check wiring!");
    while (1) {}
  }
  myDHTSensor.begin();
  Serial.println("Timer set to 30 seconds (timerDelay
variable), it will take 30 seconds before publishing
the first reading.");
}

void loop() {
  // HTTP POST მოთხოვნის გაგზავნა timerDelay დროის
შემდეგ
  if ((millis() - lastTime) > timerDelay) {
```

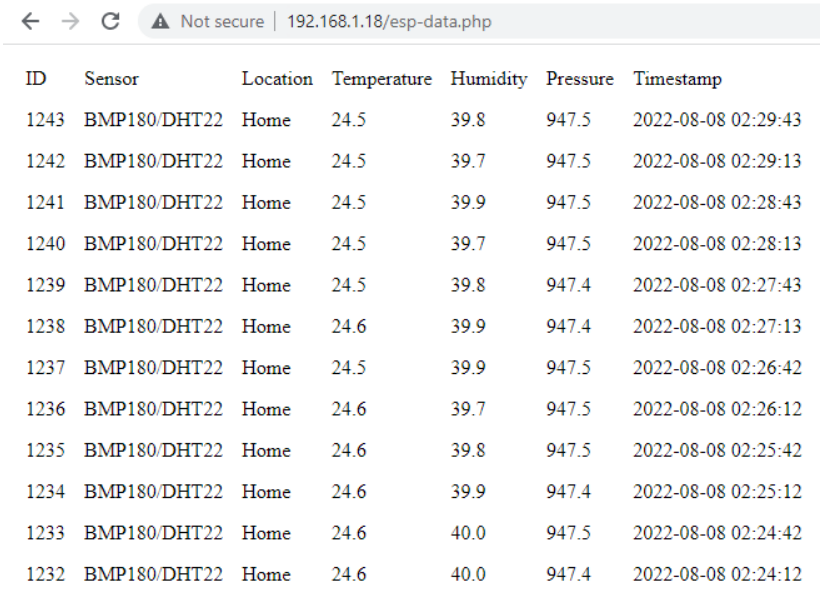
```
    // WiFi კავშირის შემოწმება
    if (WiFi.status() == WL_CONNECTED) {
      WiFiClient client;
      HTTPClient http;
      check_sensors();
      http.begin(client, serverName);
      http.addHeader("Content-Type", "application/x-
www-form-urlencoded");
      // HTTP POST-ით გასაგზავნი მონაცემები
      String httpRequestData = "api_key=" +
apiKeyValue + "&sensor=" + sensorName + "&location="
+ sensorLocation + "&temp=" + String(temp, 1) +
"&hum=" + String(hum, 1) + "&pres=" + String(pres, 1)
+ "";
      Serial.println(httpRequestData);
      int httpResponseCode =
http.POST(httpRequestData);
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      http.end();
    } else {
      Serial.println("WiFi Disconnected");
    }
    lastTime = millis();
  }
}
```

პროგრამის გაშვების შემდეგ, მიმდევრობით მონიტორში დავინახავთ (სურ. 138):

```
.......
Connected to WiFi network with IP Address: 192.168.1.14
Timer set to 30 seconds (timerDelay variable), it will take 30 seconds before publishing the first reading.
api_key=testAPIkey&sensor=BMP180/DHT22&location=Home&value1=24.8&value2=41.7&value3=946.3
HTTP Response code: 200
api_key=testAPIkey&sensor=BMP180/DHT22&location=Home&value1=24.7&value2=41.9&value3=946.2
HTTP Response code: 200
```

სურ. 138. პროგრამის შედეგები მიმდევრობითი მონიტორის ფანჯარაში

http://raspberry-pi-ip-address/esp-data.php მისამართზე შესვლისას ბრაუზერის ფანჯარაში გამოჩნდება მონაცემთა ბაზაში შეტანილი მონაცემები (სურ.139):

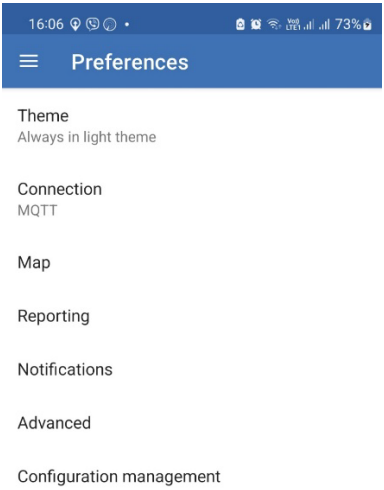


ID	Sensor	Location	Temperature	Humidity	Pressure	Timestamp
1243	BMP180/DHT22	Home	24.5	39.8	947.5	2022-08-08 02:29:43
1242	BMP180/DHT22	Home	24.5	39.7	947.5	2022-08-08 02:29:13
1241	BMP180/DHT22	Home	24.5	39.9	947.5	2022-08-08 02:28:43
1240	BMP180/DHT22	Home	24.5	39.7	947.5	2022-08-08 02:28:13
1239	BMP180/DHT22	Home	24.5	39.8	947.4	2022-08-08 02:27:43
1238	BMP180/DHT22	Home	24.6	39.9	947.4	2022-08-08 02:27:13
1237	BMP180/DHT22	Home	24.5	39.9	947.5	2022-08-08 02:26:42
1236	BMP180/DHT22	Home	24.6	39.7	947.5	2022-08-08 02:26:12
1235	BMP180/DHT22	Home	24.6	39.8	947.5	2022-08-08 02:25:42
1234	BMP180/DHT22	Home	24.6	39.9	947.4	2022-08-08 02:25:12
1233	BMP180/DHT22	Home	24.6	40.0	947.5	2022-08-08 02:24:42
1232	BMP180/DHT22	Home	24.6	40.0	947.4	2022-08-08 02:24:12

სურ. 139. სენსორებიდან მიღებული მონაცემების გამოტანა ვებ საიტზე

GPS-ის გამოყენება Node-RED გარემოში

ამოცანის მიზანია დავადგინოთ მოწყობილობის ადგილმდებარეობა და შემდეგ განვახორციელოთ შესაბამისი წინასწარ განსაზღვრული მოქმედებები, მაგალითად, სახლთან მიახლოებისას გაიღოს ავტოფარეხის კარები ან აინთოს გარე განათება. ამოცანის შესასრულებლად დაგვჭირდება სმარტფონი და მასზე დაინსტალირებული Owntracks აპლიკაცია. სმარტფონი საჭიროა GPS კოორდინატების მისაღებად, ხოლო Owntracks აპლიკაციის საშუალებით ხდება მიღებული კოორდინატების გადაცემა mqtt ან http პროტოკოლის საშუალებით. ჩვენს შემთხვევაში, კოორდინატების გადაცემა ხდება Raspberry pi-ზე დაინსტალირებული mqtt ბროკერისთვის. დავაინსტალიროთ Owntracks აპლიკაცია სმარტფონზე და მოვახდინოთ აპლიკაციის კონფიგურირება:



სურ.140. Owntracks აპლიკაციის კონფიგურირების მენიუ

Connection მენიუში შევავსოთ:

Mode - MQTT

Host - MQTT ბროკერის პარამეტრები (IP მისამართი და პორტი)

ClienT ID –

Identification მენიუში:

Username - MQTT ბროკერის მომხმარებლის სახელი

Password - MQTT ბროკერის პაროლი

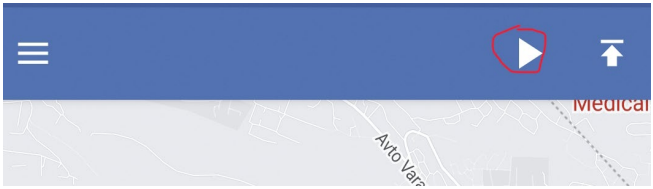
Device ID - myphone

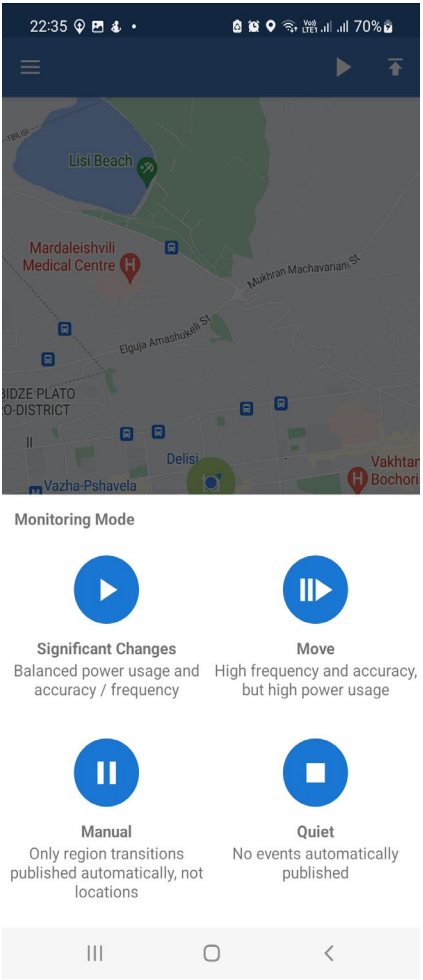
Tracker ID –

Security:

გამოვრთოთ TLS

ეკრანზე Monitoring Mode მენიუდან ავირჩიოთ Significant Changes:



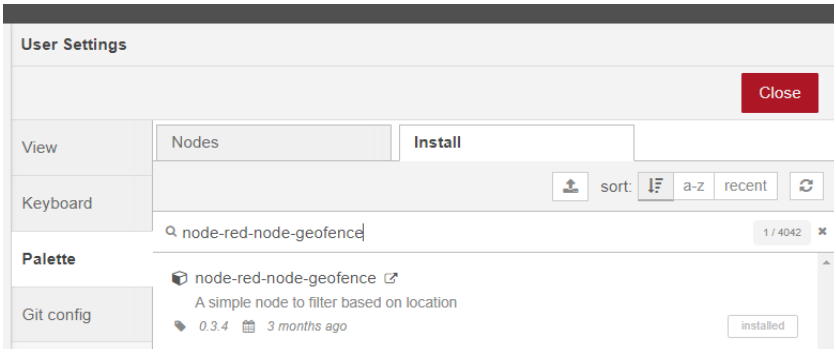


სურ.141 . მონიტორინგის რეჟიმის შერჩევა

ამით დასრულდა Owntracks აპლიკაციის კონფიგურირება.

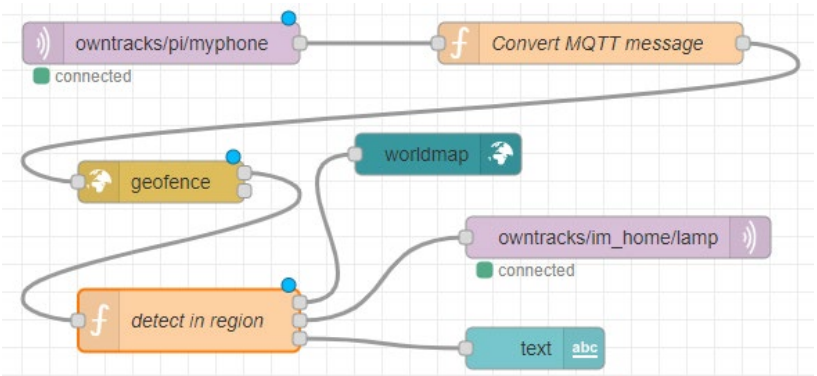
Node-RED-ის გარემოში დავაინსტალიროთ ორი კომპონენტი: node-red-node-geofence და node-red-contrib-web-

worldmap. Node-RED-ის მთავარი მენიუს Manage palette განყოფილებაზე დაჭერის შედეგად გამოჩნდება User Settings ფანჯარა, ვირჩევთ install ჩანართს. ძებნის ველში ვწერთ ზემოთ აღნიშნული კომპონენტების დასახელებას, ვირჩევთ ამ კომპონენტებს და ვაინსტალირებთ.



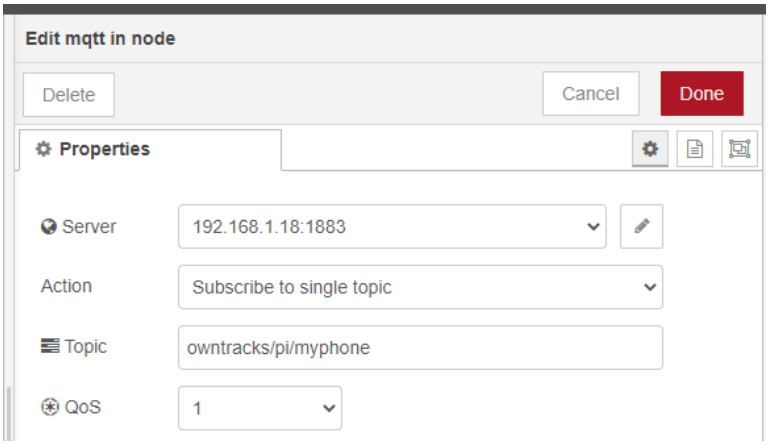
სურ. 142. Node-Red-ის კომპონენტების ინსტალაცია

Node-RED-ის გარემოში კვანძები დავუკავშიროთ ერთმანეთს ისე, როგორც ეს მოცემულია სურ. 143-ზე.



სურ. 143 . კვანძების დაკავშირება

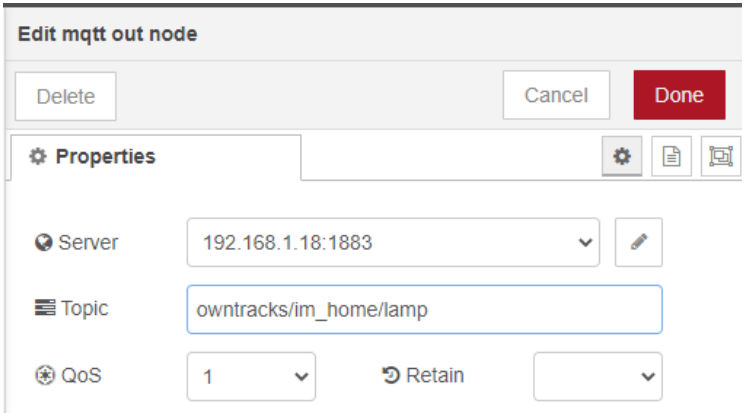
mqtt in კვანძის პარამეტრები:



სურ.144. mqtt in კვანძის პარამეტრები

მივაქციოთ ყურადღება იმას, რომ Owntracks აპლიკაცია Topic-ის დასახელებას ქმნის შემდეგი წესით: owntracks/ MQTT ბროკერის მომხმარებლის სახელი/Device ID.

mqtt out კვანძის პარამეტრები:



სურ.145. mqtt out კვანძის პარამეტრები

Convert MQTT message კვანძის კოდი:

Setup | On Start | On Message

```
var longt=0;
var lat=0;
var name="";
var data=JSON.parse(msg.payload);
if(data._type=="location"){
    lat=Number(data.lat);
    longt=Number(data.lon);
    name=data.tid;
}
msg.payload = {
    "name": name,
    "lat": lat,
    "lon": longt
}
return msg;
```

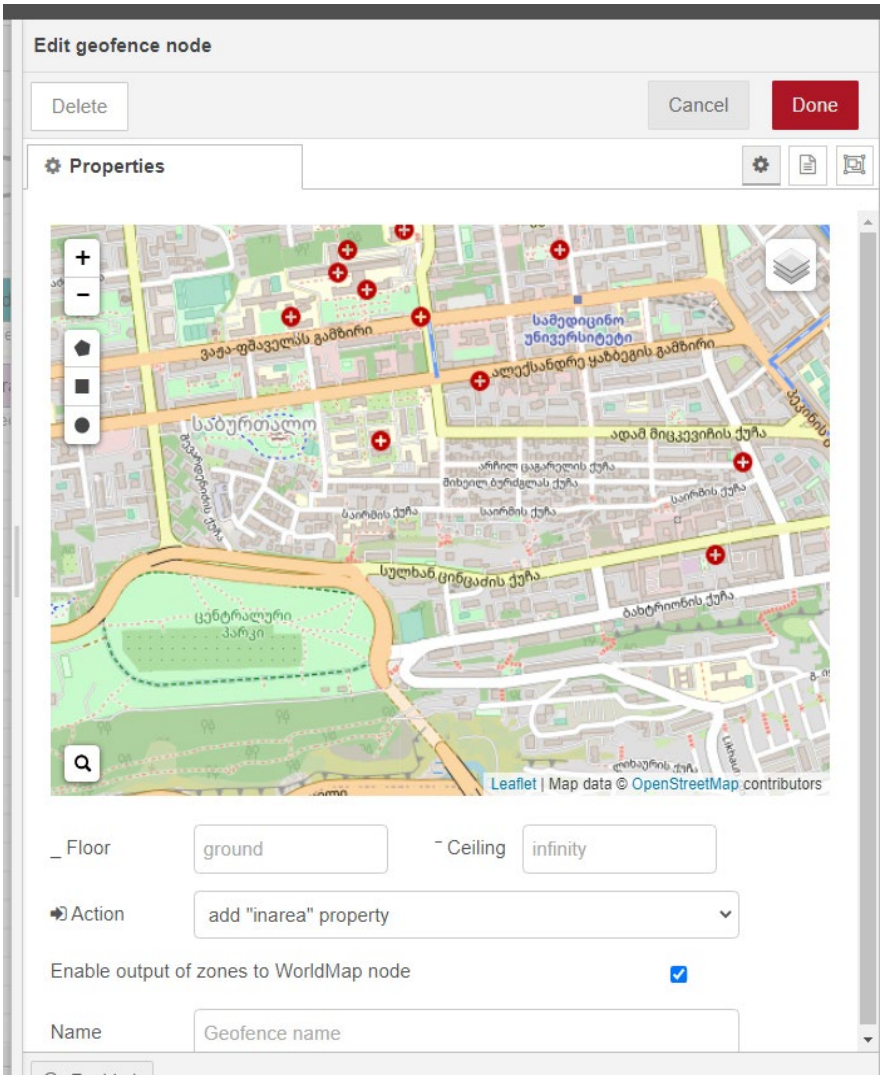
სურ.146. Convert MQTT message კვანძის კოდი

detect in region კვანძის კონფიგურაცია და კოდი:

სურ. 147. detect in region კვანძის კონფიგურაცია

```
let payload = msg.payload;
var msg1={};
var msg2={};
let location = msg.location;
let inarea_flag = false;
var im_home = 0;
var im_home_msg = "You are out of home";
if(location.inarea){
    inarea_flag = true;
}
let i_color = "blue";
msg.heating = "Off";
if(inarea_flag){
    i_color = "green";
    msg.heating = "ON";
    im_home = 1;
    im_home_msg = "You are at home";
}
msg.payload.iconColor = i_color;
msg1.payload = im_home;
msg2.payload = im_home_msg;
return [msg,msg1,msg2];
```

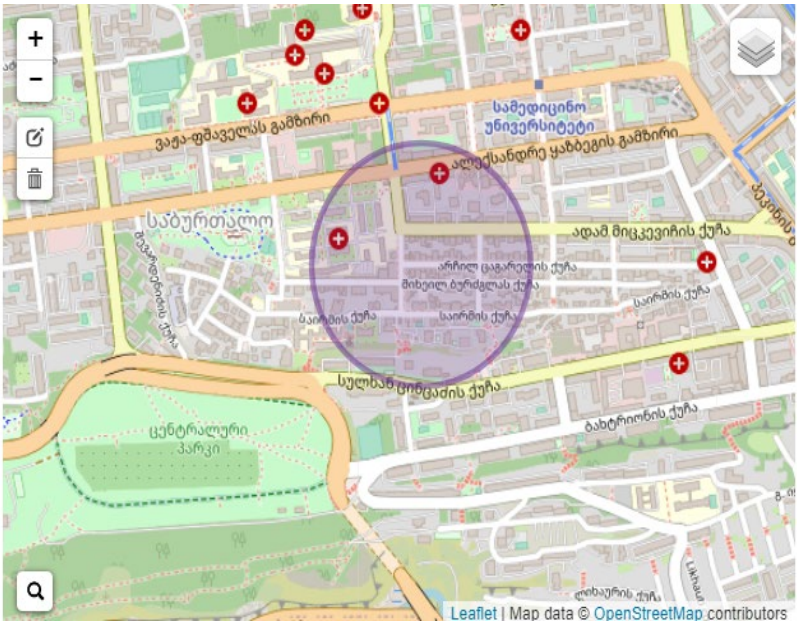
სურ. 148-ზე მოცემულია geofence კვანძის კონფიგურირების ფანჯარა. Action ჩანართში სიის ჩამონათვალიდან ავირჩიოთ add “inarea” property. ამის შემდეგ კვანძის კონფიგურირების ფანჯარაში, რუკაზე ვქმნით ზონას, რომელშიც მოხვედრის შემდეგ უნდა შესრულდეს გარკვეული მოქმედება. ზონის დასახაზად ვიყენებთ რუკის მენიუს ინსტრუმენტებს.



სურ. 148. geofence კვანძის კონფიგურირება

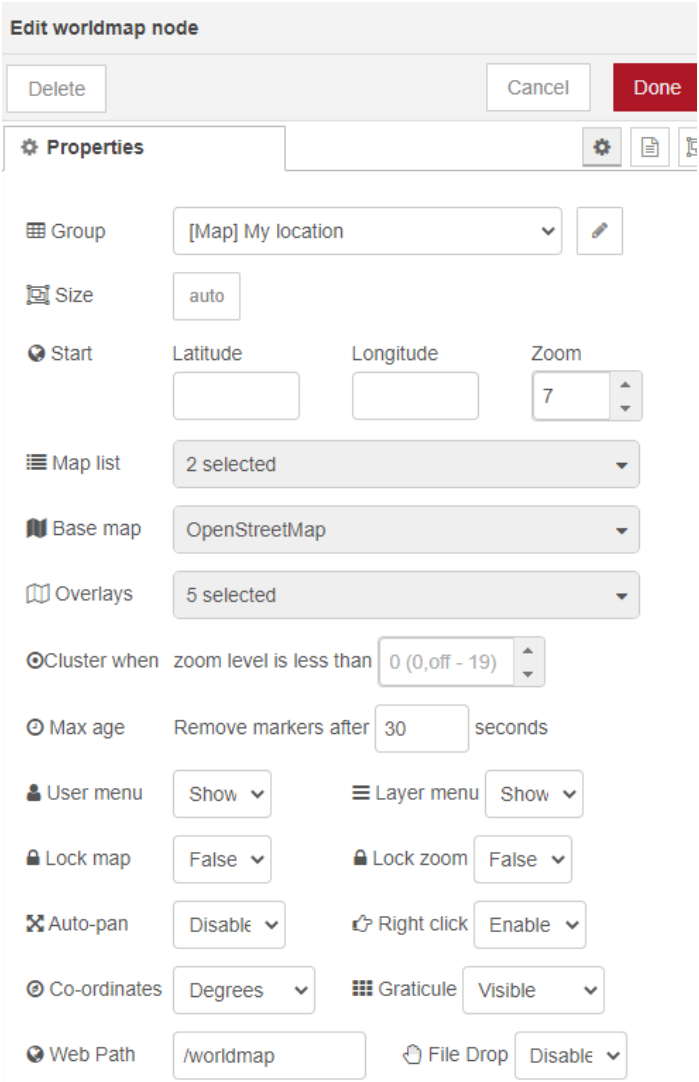
იმისათვის, რომ ზონა შევქმნათ, რუკაზე შევარჩიოთ ადგილმდებარეობა, შემდეგ მენიუს ინსტრუმენტებიდან

ავირჩიოთ ზონის ფორმა (წრე, მართკუთხედი, ხუთკუთხედი). მაუსის მარჯვენა ღილაკის საშუალებით მოვხაზოთ შესაბამისი ფორმისა და ზომის ზონა. ეს იქნება ის ადგილი, რომელშიც მოწყობილობის (ჩვენს შემთხვევაში, სმარტფონი) მოხვედრისას NodeMCU შეასრულებს გარკვეულ, წინასწარ დაპროგრამებულ მოქმედებას (მაგ. სინათლის ანთება, ავტოფარეხის კარის გაღება, კონდიციონერისა და გათბობის სისტემის, ყავის მადუღარას ჩართვა და ა.შ.).



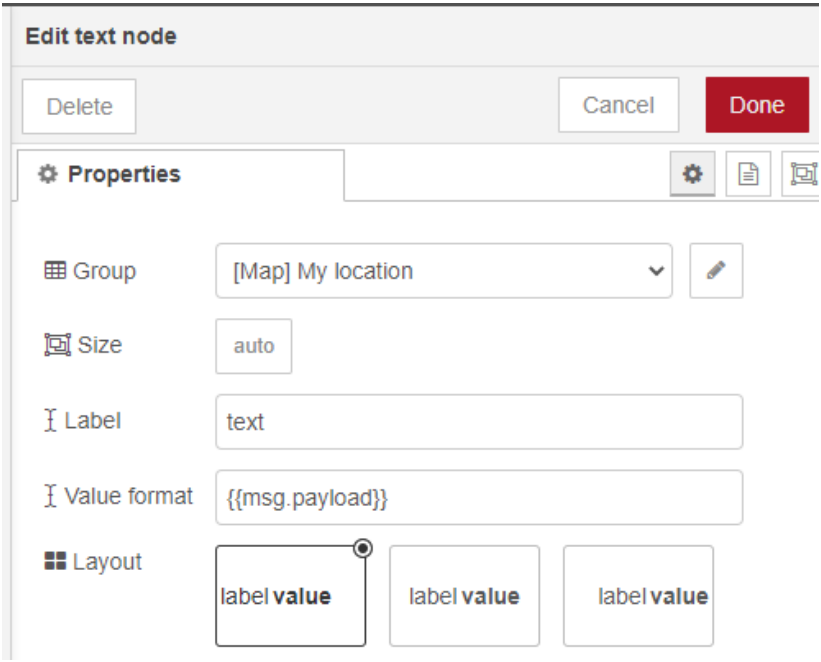
სურ.149 წრიული ზონა

Worldmap კვანძის კონფიგურირების ფანჯარა მოცემულია სურ.150-ზე:



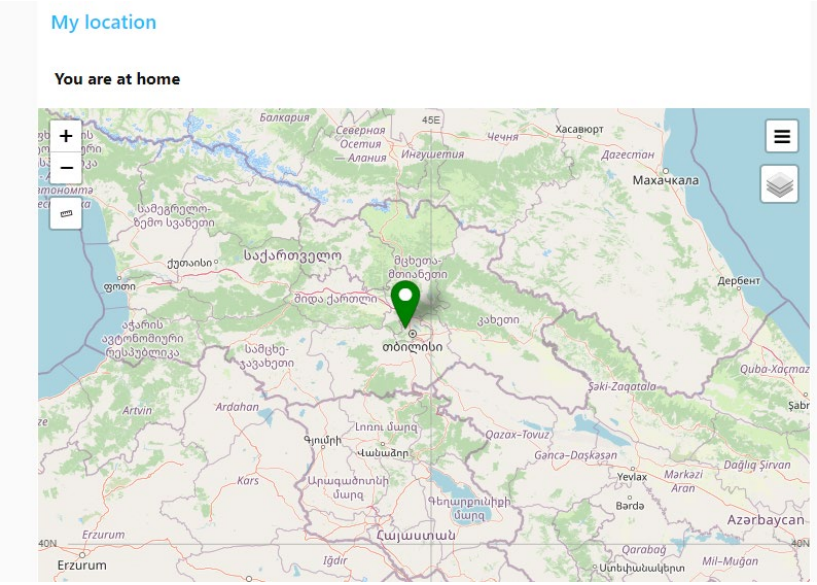
სურ.150 . Worldmap კვანძის კონფიგურირების ფანჯარა

სურ.151-ზე მოცემულია text კვანძის კონფიგურირების ფანჯარა:



სურ.151. text კვანძის კონფიგურირების ფანჯარა

კვანძების დაკავშირებისა და Deploy ღილაკზე დაჭერის შემდეგ, ბრაუზერში, Node-RED-ის UI გარემოში მივიღებთ ქვემოთ მოცემულ სურათს:

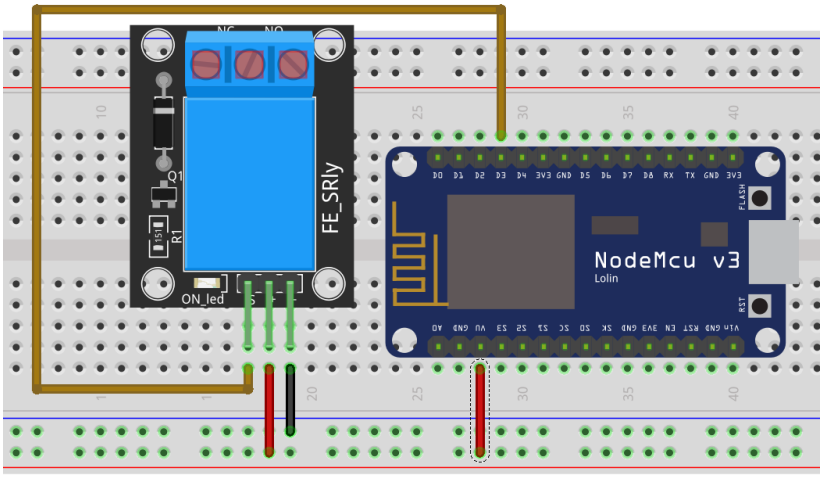


სურ.152. რუკაზე მოწყობილობის ადგილმდებარეობის ჩვენება

იმისათვის, რომ უფრო გასაგები იყოს ამოცანის მიზანი, კიდევ ერთხელ ავხსნათ, სისტემაში როგორ ხორციელდება ინფორმაციის გადაცემა და გადაცემული ინფორმაციის საფუძველზე გარკვეული მოქმედებების შესრულება. მოწყობილობა (ჩვენს შემთხვევაში, სმარტფონი) საკუთარ კოორდინატებს mqtt პროტოკოლის (თემა: owntracks/pi/myphone) საშუალებით უგზავნის Raspberry Pi-ზე დაინსტალირებულ mqtt ბროკერს; Node-RED-ის შესაბამისი კვანძების საშუალებით ხდება მიღებული შეტყობინების დამუშავება და ვებ ინტერფეისზე გამოტანა; გარდა ამისა, mqtt პროტოკოლის გამოყენებით Node-RED

უგზავნის ბრძანებას NodeMCU-ს. NodeMCU-ს საშუალებით ხდება შესაბამისი ბრძანების შესრულება.

ავაგოთ სურ. 153 -ზე მოცემული სქემა.



სურ.153. NodeMCU-სა და რელეს მოდულის შეერთების სქემა

სქემის აგების შემდეგ, NodeMCU მივაერთოთ კომპიუტერთან. გავხსნათ Arduino IDE და ჩავწეროთ შიგნით შემდეგი კოდი:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define MQTT_USERNAME "your mosquitto username"
#define MQTT_PASSWORD "your mosquitto password"
#define MQTT_HOST "your  mosquitto ip address"
#define ssid "your_wifi_ssid"
#define wifiPassword "your_wifi_password"
#define lamp_pin D3
```

```
WiFiClient espClient;
PubSubClient client(espClient);
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, wifiPassword);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("WiFi connected - ESP IP address: ");
  Serial.println(WiFi.localIP());
}
void callback(String topic, byte* message, unsigned
int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();
  if (topic == "owntracks/im_home/lamp") {
    Serial.print("Changing lamp state to ");
    if (messageTemp == "1") {
      digitalWrite(lamp_pin, HIGH);
      Serial.print("On");
    } else if (messageTemp == "0") {
```

```
      digitalWrite(lamp_pin, LOW);
      Serial.print("Off");
    }
  }
  Serial.println();
}
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266Client",
MQTT_USERNAME, MQTT_PASSWORD)) {
      Serial.println("connected");
      client.subscribe("owntracks/im_home/lamp");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
void setup() {
  pinMode(lamp_pin, OUTPUT);
  digitalWrite(lamp_pin, 0);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(MQTT_HOST, 1883);
  client.setCallback(callback);
}
void loop() {
  if (!client.connected()) {
    reconnect();
  }
```

```
  if (!client.loop())
    client.connect("NODEMCU");
}
```

როდესაც მოწყობილობა მოხვდება ჩვენს მიერ წინასწარ განსაზღვრულ ზონაში, ჩაირთვება რელეს მოდული. რელეს მოდულზე შეიძლება მიბმული იყოს განათების, გაგრილების, სისტემები, ელექტრომაგნიტური საკეტები და ა.შ.

Deep Sleep რეჟიმის გამოყენება NodeMCU-ში

Deep Sleep ტერმინი გულისხმობს მოწყობილობების მუშაობას ენერგიის დაბალი მოხმარების რეჟიმში. განვიხილოთ, როგორ ვამუშაოთ NodeMCU დაფა Deep Sleep რეჟიმში.

სურ. 154. Deep Sleep რეჟიმი

თუ NodeMCU-ს კვება ხდება აკუმულატორების გამოყენებით, გარკვეული პერიოდის შემდეგ აკუმულატორებზე ძაბვა მცირდება, განსაკუთრებით თუ დაფა

იყენებს უსადენო ქსელს. სწორედ ამიტომ არის ძალიან აქტუალური Deep Sleep-ის გამოყენება.

NodeMCU-ს Deep Sleep რეჟიმში გადაყვანა ნიშნავს ზოგი აქტიური პროცესის შეწყვეტას, რომელიც მოითხოვს დიდ სიმძლავრეს (მაგალითად, Wi-Fi), რჩება მხოლოდ ის აუცილებელი პროცესები, რომლებიც საჭიროა პროცესორის ძილის რეჟიმიდან გამოსაყვანად, საჭირო დროს ან საჭირო მომენტში.

ენერგიის მოხმარების თვალსაზრისით, NodeMCU-ს გააჩნია მუშაობის ოთხი რეჟიმი:

- Active Mode
- Modem–Sleep Mode
- Light–Sleep Mode
- Deep–Sleep Mode

	Modem – Sleep	Light – Sleep	Deep – Sleep
CPU	ON	Pause	OFF
System Clock	ON	OFF	OFF
WI-Fi	OFF	OFF	OFF
RTC	ON	ON	ON
მოხმარებული დენი	15 mA	0.9 mA	20 uA

ცხრილი 2. NodeMCU-ს მუშაობის რეჟიმების შედარება (Espressif systems-ის მონაცემების მიხედვით)

Active Mode წარმოადგენს NodeMCU-ს მუშაობის ჩვეულ რეჟიმს. ამ დროს პროცესორი მუშაობს სრული დატვირთვით, ჩართულია უსადენო ქსელის მოდული, ისე რომ NodeMCU გადასცემს, იღებს და აყურადებს უსადენო ქსელის სიგნალებს.

Active Mode-გან განსხვავებით, Modem–Sleep Mode-ში პროცესორი მუშაობს ნაკლები დატვირთვით იმის გამო, რომ გამორთულია უსადენო ქსელის მოდული. ეს რეჟიმი გამოიყენება იმ შემთხვევაში, როდესაც გვჭირდება მხოლოდ და მხოლოდ GPIO (ADC, PWM ან I2C) საკონტაქტო გამომყვანები.

Light-Sleep Mode-ში პროცესორი, RTC ბლოკი და სხვა პერიფერიული მოწყობილობები გამორთულია, ასეთ დროს Wi-Fi მოდულიც გადასულია ენერგიის ნაკლები მოხმარების რეჟიმში. Light-Sleep Mode-დან გამოსასვლელად გარე GPIO ტრიგერია საჭირო, თუმცა აქვე შევნიშნავთ, რომ GPIO16-ის გამოყენება ამ მიზნით არ შეიძლება.

Deep-Sleep Mode-ში პროცესორი, პერიფერიული მოწყობილობები და უსადენო ქსელის მოდული ენერგიის მინიმალური მოხმარების რეჟიმშია, მუშაობს მხოლოდ RTC ბლოკი.

Modem-Sleep Mode-სა და Light-Sleep Mode-ის გააქტიურება სისტემის მიერ ხდება ავტომატურად, Deep Sleep Mode-ს რთავს მომხმარებელი.

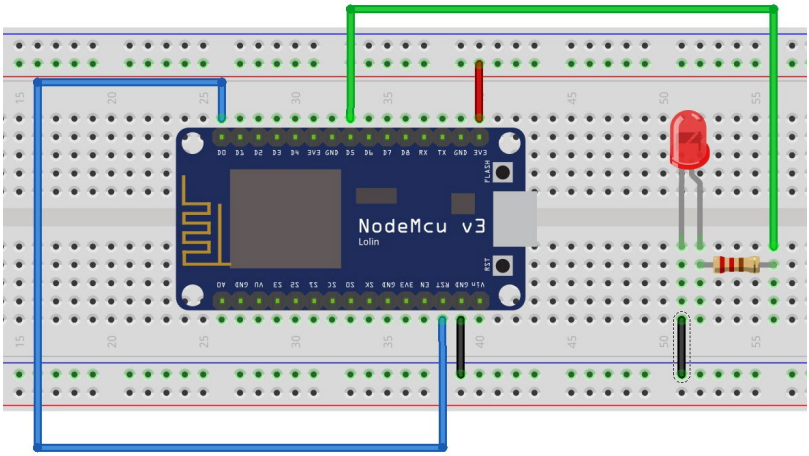
Deep Sleep Mode-ის ჩასართავად, GPIO16 (D0) და RST გამომყვანები უნდა მივუერთოთ ერთმანეთს.

NodeMCU-ს გადასაყვანად Deep Sleep Mode-ში გამოვიძახოთ ESP.deepSleep() ფუნქცია. ფუნქციის არგუმენტად შეიძლება მივუთითოთ ის დრო (მიკროწამებში), რომლის განმავლობაშიც მოწყობილობა ძილის რეჟიმში იმყოფება. ძილის რეჟიმიდან გამოყვანა შესაძლებელია ორი გზით: ავტომატურად ან გარე ტრიგერით.

ძილის რეჟიმიდან ავტომატურად გამოსვლა გულისხმობს იმას, რომ NodeMCU გაიღვიძებს იმ დროის გასვლის შემდეგ, რომელიც ESP.deepSleep() ფუნქციის არგუმენტად გვაქვს მითითებული. გარე ტრიგერით ძილის რეჟიმიდან გამოსვლისთვის საჭიროა დავაჭიროთ RST ღილაკს ან RST გამომყვანს წამიერად მივანიჭოთ დაბალი დონის სიგნალი.

განვიხილოთ მაგალითები ორივე შემთხვევისთვის.

Deep Sleep Mode-დან NodeMCU-ს ავტომატური გამოსვლის მაგალითის სადემონსტრაციოდ ავაწყოთ სქემა სურ. 155-ის მიხედვით. მივუერთოთ D0 და RST საკონტაქტო გამომყვანები ერთმანეთს, შუქდიოდი დავუკავშიროთ D5 საკონტაქტო გამომყვანს.



სურ. 155. Deep Sleep Mode-დან NodeMCU-ს ავტომატური გამოსვლა

D0 და RST გამომყვანები ერთმანეთთან შევაერთოთ მხოლოდ და მხოლოდ კოდის ატვირთვის შემდეგ.

ჩვეულებრივ რეჟიმში, RST გამომყვანს მუდმივად მინიჭებული აქვს მაღალი დონის სიგნალი. იმისათვის, რომ გადავტვირთოთ NodeMCU, RST გამომყვანს წამიერად უნდა მივანიჭოთ დაბალი დონის სიგნალი. ESP.deepSleep() ფუნქციის არგუმენტად თუ მივუთითებთ გარკვეულ დროს, ამ დროის გავლის შემდეგ, D0-ზე წამიერად გაჩნდება დაბალი დონის სიგნალი, იმის გამო, რომ D0 მიერთებულია RST გამომყვანთან, დაბალი დონის სიგნალი გაჩნდება RST-ზეც, ეს გამოიწვევს პროცესორის გამოღვიძებას (გადატვირთვას). მაგალითად, თუ გვაქვს ESP.deepSleep(10e6) ფუნქცია, ეს ნიშნავს, რომ პროცესორი იძინებს 10 წმ-ის

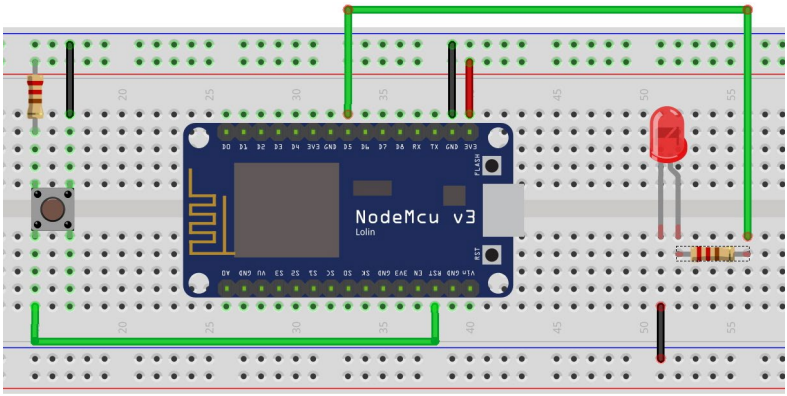
განმავლობაში. ამ დროის გავლის შემდეგ, პროცესორი დაბალი დონის სიგნალს უგზავნის D0 გამომყვანს.

ავტვირთოთ ქვემოთ მოცემული კოდი Arduino IDE-ში. კოდის ატვირთვის შემდეგ D0 საკონტაქტო გამომყვანი მივაერთოთ RST საკონტაქტო გამომყვანს. ეს შეერთება უზრუნველყოფს NodeMCU-ს Deep Sleep რეჟიმის აქტივაციას. როგორც პროგრამის კოდიდან ჩანს, შუქდიოდი 5 წამის განმავლობაში იმყოფება ჩართულ მდგომარეობაში, ამ დროის გავლის შემდეგ NodeMCU გადადის Deep Sleep რეჟიმში 10 წამით, ამ დროს შუქდიოდი გამოირთვება; 10 წამის შემდეგ NodeMCU ავტომატურად გამოდის Deep Sleep რეჟიმიდან, ამ დროს შუქდიოდი ინთება. ასე მეორდება პროცესი უსასრულოდ.

```
#define ledPin D5
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(5000);
  ESP.deepSleep(10e6);
}
```

განვიხილოთ მეორე შემთხვევა: Deep Sleep Mode-დან NodeMCU-ს გამოსვლა გარე ტრიგერის საშუალებით. როგორც ზემოთ ავღნიშნეთ, RST გამომყვანს მუდმივად მინიჭებული აქვს მაღალი დონის სიგნალი. RST გამომყვანის LOW

მდგომარეობაში გადასაყვანად, გამოვიყენოთ ღილაკი. იგივე დანიშნულებით შეგვიძლია გამოვიყენოთ NodeMCU დაფის RST ღილაკი. ავაგოთ სქემა სურ. 156-ის მიხედვით. მივაერთოთ ღილაკი და RST საკონტაქტო გამომყვანი ერთმანეთს, შუქდიოდი დავუკავშიროთ D5 საკონტაქტო გამომყვანს.



სურ. 156. Deep Sleep Mode-დან NodeMCU-ს გამოსვლა გარე ტრიგერის გამოყენებით

წინა მაგალითის კოდისგან განსხვავებით, ამ მაგალითში ESP.deepSleep() ფუნქციის არგუმენტი ნულის ტოლია. ასეთ შემთხვევაში, Deep Sleep რეჟიმი უსასრულოდ გრძელდება. ძილის რეჟიმიდან გამოსასვლელად, RST გამომყვანს ღილაკის საშუალებით წამიერად მივანიჭოთ დაბალი დონის სიგნალი.

ავტვირთოთ ქვემოთ მოცემული კოდი Arduino IDE-ში.

```
#define ledPin D5
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(5000);
  ESP.deepSleep(0);
}
```

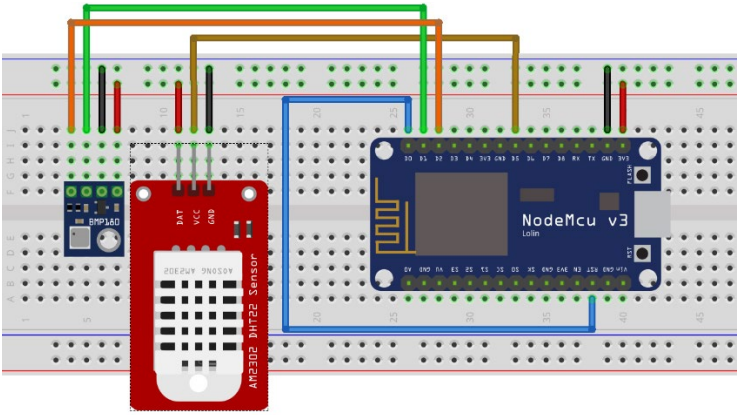
კოდის გაშვების შემდეგ შუქდიოდი ინთება 5 წმ-ის განმავლობაში, რის შემდეგაც NodeMCU გადადის უსასრულო Deep Sleep რეჟიმში, ამ დროს შუქდიოდი ჩაქრება. ღილაკზე წამიერად დაჭერის შედეგად, შუქდიოდი ისევ აინთება 5 წამის განმავლობაში, შემდეგ პროცესორი ისევ გადადის უსასრულო Deep Sleep რეჟიმში, მანამ სანამ კვლავ არ მოხდება ღილაკზე დაჭერა.

განვიხილოთ NodeMCU დაფის Deep Sleep Mode-დან ავტომატურად გამოსვლის პრაქტიკული გამოყენების მაგალითი სენსორებიდან მიღებული მონაცემების გამოსაქვეყნებლად.

ქვემოთ განხილულ ამოცანაში სრულდება შემდეგი მოქმედებები:

- NodeMCU უკავშირდება უსადენო ქსელს;
- NodeMCU კითხულობს სენსორების მონაცემებს და აქვეყნებს MQTT ბროკერში;

- NodeMCU გადადის Deep Sleep-ში, წინასწარ მითითებული დროით;
- NodeMCU იღვიძებს წინასწარ განსაზღვრული დროის გავლის შემდეგ;
- პროცესი მეორდება უსასრულოდ.



სურ. 157. სენსორების მიერთება NodeMCU დაფასთან და Deep Sleep-ის აქტივაციის სქემა

ავაგოთ სქემა სურ. 157-ის მიხედვით. ავტვირთოთ ქვემოთ მოცემული კოდი Arduino IDE-ში.

```
#include <PubSubClient.h>
#include <ESP8266WiFi.h>
#include <Adafruit_BMP085.h>
#include "DHT.h"
#define ssid "your_wifi_ssid"
#define wifiPassword "your_wifi_password"
#define MQTT_HOST "your  mosquitto ip address"
```

```
#define MQTT_PORT 1883
#define MQTT_USERNAME "your mosquitto username"
#define MQTT_PASSWORD "your mosquitto password"
#define MQTT_CLIENT_ID "NodeMCUClient"
#define MQTT_PUB_PRES "room/pressure"
#define MQTT_PUB_HUM "room/humidity"
#define MQTT_PUB_BMP_TEMP "room/temperature_bmp"
#define MQTT_PUB_TEMP "room/temperature_bmp"
WiFiClient espHomeAuto;
PubSubClient client(espHomeAuto);
Adafruit_BMP085 bmp;
#define DHTPIN D5
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22  // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
DHT myDHTSensor(DHTPIN, DHTTYPE);
float temp;
float pres;
float hum;
float bmp_temp;
char msg[10];
void setup_wifi() {
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, wifiPassword);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("WiFi connected - ESP IP address: ");
  Serial.println(WiFi.localIP());
```

```
}
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(MQTT_CLIENT_ID, MQTT_USERNAME,
MQTT_PASSWORD)) {
      Serial.println("connected");
      client.subscribe(MQTT_PUB_BMP_TEMP);
      client.subscribe(MQTT_PUB_PRES);
      client.subscribe(MQTT_PUB_HUM);

    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
void check_sensors() {
  Serial.print("H=");
  hum = myDHTSensor.readHumidity();
  Serial.print(hum, 1);
  Serial.print("% \t");
  Serial.print("T=");
  temp = myDHTSensor.readTemperature();
  Serial.print(temp, 1);
  Serial.print("C \t");
  Serial.print("P=");
  pres = bmp.readPressure() / 100.0;
  Serial.print(pres, 0);
  Serial.print("Hpa \t");
  Serial.print("Tt=");
```

```
  bmp_temp = bmp.readTemperature();
  Serial.print(bmp_temp, 1);
  Serial.println("C");
  sprintf(msg, "%.1f", temp);
  client.publish(MQTT_PUB_TEMP, msg);
  sprintf(msg, "%.1f", pres);
  client.publish(MQTT_PUB_PRES, msg);
  sprintf(msg, "%.1f", hum);
  client.publish(MQTT_PUB_HUM, msg);
  sprintf(msg, "%.1f", bmp_temp);
  client.publish(MQTT_PUB_BMP_TEMP, msg);
}
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, wifiPassword);
  while (WiFi.waitForConnectResult() != WL_CONNECTED)
{
    Serial.println("Connection Failed!
Rebooting...");
    delay(5000);
    ESP.restart();
  }

  if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP085
sensor, check wiring!");
    while (1) {}
  }
  client.setServer(MQTT_HOST, MQTT_PORT);
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
```

```
  myDHTSensor.begin();
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  if (!client.loop()) {
    client.connect(MQTT_CLIENT_ID);
  }
  check_sensors();
  delay(200);
  Serial.println("I'm awake, but I'm going into deep
sleep mode for 60 seconds");
  ESP.deepSleep(60e6);
}
```

Arduino IDE-ს გარემოდან NodeMCU-ში პროგრამის ჩაწერა უსადენო ქსელის გამოყენებით (OTA)

OTA (Over the Air) განახლება გულისხმობს პროგრამული უზრუნველყოფის ჩაწერას NodeMCU-ში, Wi-Fi ქსელის გამოყენებით, usb პორტის გვერდის ავლით. ამ ფუნქციის ხიბლი მდგომარეობს იმაში, რომ მთელი პროცედურა ხორციელდება NodeMCU მოდულთან ფიზიკური წვდომის გარეშე.

იმისათვის, რომ NodeMCU მოვამზადოთ OTA-ს მხარდაჭერისთვის, შევასრულოთ შემდეგი მოქმედებები:

- დავაკავშიროთ NodeMCU კომპიუტერთან USB კაბელის მეშვეობით;
- Arduino IDE-ს მენიუდან შევარჩიოთ დაფის ტიპი და პორტი.
- Arduino IDE-ს მაგალითების საქაღალდიდან (File->examples) გავხსნათ BasicOTA ფაილი;
- ფაილში ჩავწეროთ ჩვენი უსადენო ქსელის მონაცემები;
- ავტვირთოთ ფაილი.

ქვემოთ მოცემულია BasicOTA.ino ფაილი.

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#ifndef STASSID
#define STASSID "your-ssid"
#define STAPSK "your-password"
#endif
const char* ssid = STASSID;
const char* password = STAPSK;
void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED)
{
    Serial.println("Connection Failed!
Rebooting...");
    delay(5000);
```

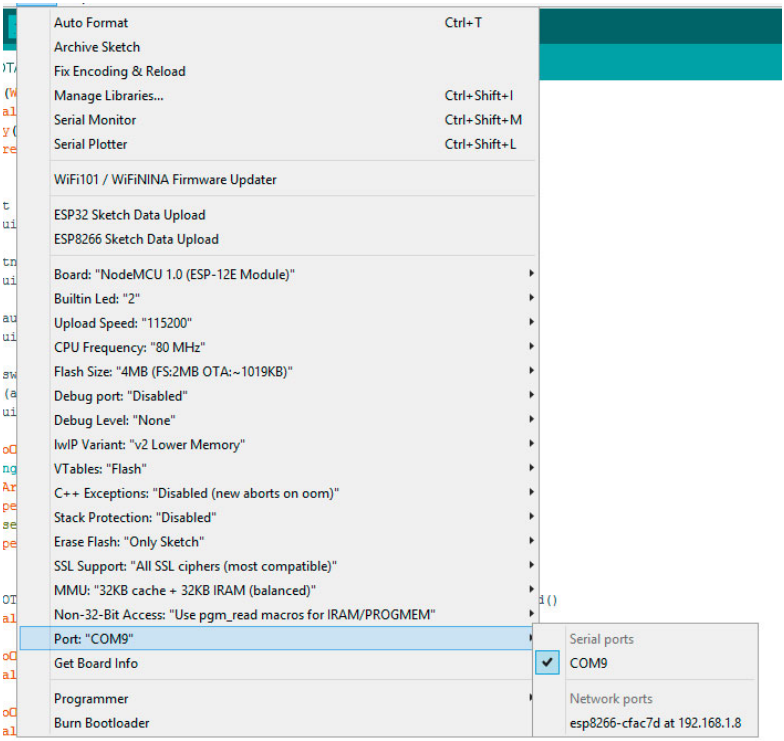
```
    ESP.restart();
  }

  // Port defaults to 8266
  // ArduinoOTA.setPort(8266);

  // Hostname defaults to esp8266-[ChipID]
  ArduinoOTA.setHostname("NODEMCU#1");

  // No authentication by default
  // ArduinoOTA.setPassword("admin");

  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  //
ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4
a801fc3");

  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
      type = "sketch";
    } else {  // U_FS
      type = "filesystem";
    }

    // NOTE: if updating FS this would be the place
to unmount FS using FS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
```

```
  ArduinoOTA.onProgress([](unsigned int progress,
unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress /
(total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) {
      Serial.println("Auth Failed");
    } else if (error == OTA_BEGIN_ERROR) {
      Serial.println("Begin Failed");
    } else if (error == OTA_CONNECT_ERROR) {
      Serial.println("Connect Failed");
    } else if (error == OTA_RECEIVE_ERROR) {
      Serial.println("Receive Failed");
    } else if (error == OTA_END_ERROR) {
      Serial.println("End Failed");
    }
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}
```

Arduino IDE-ს მიმდევრობითი ტერმინალის ფანჯარაში მივიღებთ NodeMCU-ს ip მისამართს.

ზემოთ აღნიშნული პროცედურების შესრულების შემდეგ უკვე შეგვიძლია ავტვირთოთ ახალი სკეტჩი OTA-ს

გამოყენებით. ამისათვის პირველ რიგში, NodeMCU-ს დაფა გამოვაერთოთ კომპიუტერის USB პორტიდან, ჩავრთოთ ის ჩვეულებრივ კვების წყაროში (მაგალითად, ეს შეიძლება იყოს ტელეფონის დამტენი). შემდეგ ეტაპზე, Arduino IDE-ს გარემოში, Tools მენიუდან ავირჩიოთ პორტი, პორტების ნუსხაში გამოჩნდება esp8266-xxxxxx at your_esp_ip_address ტიპის წარწერა. ვირჩევთ ამ მოწყობილობას (სურ. 158).



სურ. 158. პორტების ნუსხა

იმისათვის, რომ ჩვენს სკეტჩს OTA-ს მხარდაჭერა ჰქონდეს, პროგრამის ტანში დავამატოთ შემდეგი სტრიქონები:

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "your-ssid";
const char* password = "your-password";
void setup() { -ში :
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED)
{
    Serial.println("Connection Failed!
Rebooting...");
    delay(5000);
    ESP.restart();
  }

  // Port defaults to 8266
  // ArduinoOTA.setPort(8266);

  // Hostname defaults to esp8266-[ChipID]
  // ArduinoOTA.setHostname("myesp8266");

  // No authentication by default
  // ArduinoOTA.setPassword("admin");
```

```
  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  //
ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4
a801fc3");

  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else  // U_SPIFFS
      type = "filesystem";
    // NOTE: if updating SPIFFS this would be the
place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
  ArduinoOTA.onProgress([](unsigned int progress,
unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress /
(total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth
Failed");
    else if (error == OTA_BEGIN_ERROR)
Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR)
Serial.println("Connect Failed");
```

```
    else if (error == OTA_RECEIVE_ERROR)
Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR)
Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  void loop()-ში დავამატოთ
  ArduinoOTA.handle();
```

დავაჭიროთ Upload ღილაკს და დაველოდოთ „Done uploading" შეტყობინებას.

მაგალითისთვის განვიხილოთ NodeMCU-ს გამოსასვლელზე მაღალი და დაბალი დონის სიგნალების მიხედვით შუქდიოდის მდგომარეობის ცვლილების ამოცანა (სურ. 33). სკეტჩს დავამატოთ OTA-ს მხარდაჭერა, კოდის ეს ნაწილი მოცემულია განსხვავებული ფერით.

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "your ssid";
const char* password = "your wifi password";
#define led_pin D5
void setup() {
  pinMode(led_pin, OUTPUT);
  Serial.begin(115200);
```

```
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED)
{
    Serial.println("Connection Failed!Rebooting...");
    delay(5000);
    ESP.restart();
  }
  // Port defaults to 8266
  // ArduinoOTA.setPort(8266);
  // Hostname defaults to esp8266-[ChipID]
  // ArduinoOTA.setHostname("myesp8266");
  // No authentication by default
  // ArduinoOTA.setPassword("admin");
  // Password can be set with it's md5 value as well
  // MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
  //
ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4
a801fc3");
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else  // U_SPIFFS
      type = "filesystem";
    // NOTE: if updating SPIFFS this would be the
place to unmount SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
  });
```

```
  ArduinoOTA.onProgress([](unsigned int progress,
unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress /
(total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth
Failed");
    else if (error == OTA_BEGIN_ERROR)
Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR)
Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR)
Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR)
Serial.println("End Failed");
  });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {
  ArduinoOTA.handle();
  digitalWrite(led_pin, HIGH);
  delay(1000);
  digitalWrite(led_pin, LOW);
  delay(1000);
}
```

ამ გზით შეგვიძლია ნებისმიერი პროგრამის ჩაწერა უსადენო ქსელის გამოყენებით Arduino IDE-ს გარემოში.

გამოყენებული ლიტერატურა

1. ზ. ტაბატაძე, თ. თოდუა. არდუინო. პრაქტიკული სახელმძღვანელო დამწყებთათვის. თბილისი, 2019;
2. J. Shovic. Raspberry Pi IoT Projects. Prototyping Experiments for Makers. Second edition. Apress. 2021;
3. T. Hagino. Practical Node-RED Programming. Packt Publishing. 2021;
4. R. Singh, A, Gehlot, L. R. Gupta, B. Singh, M. Swain. Internet of Things with Raspberry Pi and Arduino. CRC Press. Taylor and Francis Group. 2020.
5. D. Norris. Home Automation with Raspberry Pi: Projects Using Google Home, Amazon Echo, and Other Intelligent Personal Assistants. McGraw-Hill Education TAB. 2019.
6. Tripathy B.K., Anuradha J. Internet of Things (IoT). Technologies, Applications, Challenges, and Solutions. CRC Press, 2017;
7. https://randomnerdtutorials.com/
8. https://icircuit.net/
9. https://lastminuteengineers.com/
10. https://www.tomshardware.com/
11. https://microcontrollerslab.com/
12. https://iotdesignpro.com/
13. https://www.element14.com/
14. http://www.steves-internet-guide.com/

სარჩევი