Final Examination Work

Automated Feeder Controlled by a Web Application

Subject: Microcontroller Architecture

Mentor:

dr Zoran Milivojević

Student:

Ilija Milojković
Rer 21/20

July 2024.

THE ACADEMY OF APPLIED TECHNICAL AND PRESCHOOL STUDIES

Final Examination Work

Automated Feeder Controlled by a Web Application

Subject: Microcontroller Architecture

Committee Members:

1. dr Nataša Nešić
2. dr Dejan Blagojević
3. dr Zoran Milivojević

Student:

Ilija Milojković
Rer 21/20

July 2024.

Acknowledgment

I want to express my sincere gratitude to all the professors at the Academy of Applied Tehnicial and Preschool Studies in Niš, especially to my mentor, Professor Dr. Zoran Milivojević, for his professional and high-quality teaching, effort, and great dedication to providing us with the best possible knowledge.

Contents

1. INTRODUCTION

In this paper, the project task is presented and described in detail, aimed at allowing the user to control an automated feeder and feed their pet in a very simple way via a web application.

Additionally, this paper will provide a deeper understanding of the components that are crucial for the realization of this project. The role and functionality of each component will be thoroughly analyzed, exploring how their integration contributes to the overall functional aspect of the device.

For the realization of this project, the Arduino UNO R3 microcontroller system, based on the ATmega328P microcontroller, was used. The system includes a Wi-Fi module ESP-01, two weight measurement sensors with the HX-711 driver, two DC/DC step-down converters, a stepper motor with the ULN2003 driver, a submersible water pump, a relay, two LEDs, a DC connector, a button, and a switch.

The final paper consists of seven chapters:

a) The first chapter is dedicated to the introduction, detailing the components used in this project, along with a clear presentation of the purpose of this paper.
b) The second chapter provides information about the microcontroller, its structure, and functionality, along with an explanation of the difference between microcontrollers and microprocessors.
c) The third chapter focuses on the ATmega328P microcontroller, presenting the pinout diagram and thoroughly explain aspects of its architecture.
d) The fourth chapter is dedicated to the Arduino UNO R3 microcontroller development system, explaining its appearance and pinout diagram. The functions of the pins and their specific roles are also detailed, as well as the Arduino IDE software environment for programming.
e) The fifth chapter is dedicated to the Node.js platform, which was used for developing the server side of the application.
f) The sixth chapter is dedicated to the React library, which was used to create the client side of the application.
g) The seventh chapter outlines the functional requirements of the project task, along with a detailed hardware and software implementation of the project. The appearance of the project and the algorithms applied for its realization are presented.

2. MICROCONTROLLER

A microcontroller is an integrated chip that receives input information and based on the programmed instructions or set criteria, processes this information and generates corresponding output signals. Unlike a microprocessor, which has a similar role, a microcontroller has built-in memory and input-output units, making it practically a microcomputer in a single integrated circuit.

In 1971, Intel introduced its first microprocessor, the 4004. At that time, there was already a demand for microcontrollers: the TMS1802 from Texas Instruments, designed for use in calculators, was already being advertised for applications in cash registers, clocks, and measuring instruments by the end of 1971. The TMS 1000, introduced in 1974, already had built-in RAM, ROM, and I/O components on the chip itself and can be considered one of the first microcontrollers, although it was then called a microcomputer. The first controllers to truly become widespread were the Intel 8048, integrated into PC keyboards, and its successor, the Intel 8051, as well as the 68HCxx series of microcontrollers from Motorola [1].

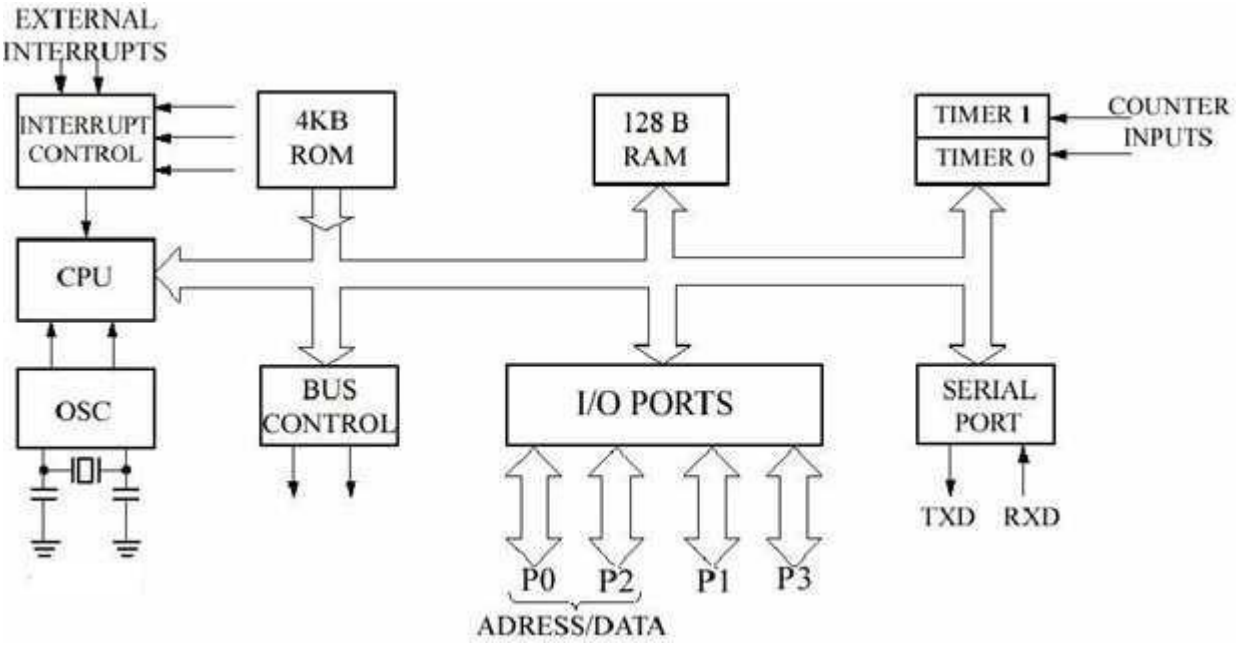


Figure 2.1. Structure of the INTEL 8051 Microcontroller Chip [2].

In Figure 2.1, the structure of the INTEL 8051 microcontroller chip is shown. The following list contains the modules commonly found in microcontrollers:

a) **Processor Core**: The CPU of the controller. It contains the arithmetic logic unit, control unit, and registers (stack pointer, program counter, accumulator, register file, and others).

b) **Memory**: Memory is divided into program memory and data memory. In larger controllers, the DMA (Direct Memory Access) controller manages data transfer between peripheral components and memory.

c) **Interrupt Controller**: Interrupts are useful for interrupting the normal flow of the program in case of external or internal events.

d) **Timer/Counter**: Most controllers have at least one, more often 2-3 timers/counters that can be used for timing events, measuring intervals, or counting events.

e) **Digital I/O (Inputs/Outputs)**: Parallel digital I/O ports are one of the main features of microcontrollers. The number of I/O pins varies from 3-4 to over 90, depending on the controller family and type.

f) **Analog I/O (Inputs/Outputs)**: Except for a few smaller controllers, most microcontrollers have integrated analog/digital converters, which vary in the number of channels (2-16) and their resolution (8-12 bits).

g) **Interfaces**: Controllers usually have at least one serial interface that can be used for downloading programs and communicating with a computer in general. Since serial interfaces can also be used for communication with external peripheral devices, most controllers offer several different interfaces such as SPI and SCI.

h) **Watchdog Timer**: Given that safety-critical systems are a significant area of application for microcontrollers, it is important to protect against software and/or hardware errors. The watchdog timer is used to reset the controller in case of a software crash.

i) **Debugging Unit**: Some controllers are equipped with additional hardware that allows for remote debugging of the chip via a computer. The main advantage of this unit is that it enables developers to identify and correct errors without the need for special debugging software.

2.1. DIFFERENCE BETWEEN MICROCONTROLLERS AND MICROPROCESSORS

A microprocessor, which is a combination of the words "micro" and "processor," is an integrated chip that performs arithmetic and logical operations in computers and similar electronic devices. Today, it is often simply referred to as a "processor" to describe this crucial component of a computer. It plays a central role in data processing as the CPU (Central Processing Unit). This is the basic element of any electronic computer.

A microcontroller, in terms of architecture, is similar to a microcomputer, but in appearance, it resembles a microprocessor because it is physically a single chip. The main difference between a microprocessor and a microcontroller is that the architecture of a microcontroller is optimized for integrating various electronic circuits, real-time process control, low cost, and low power consumption, whereas the emphasis for microprocessors is on speed and performance.

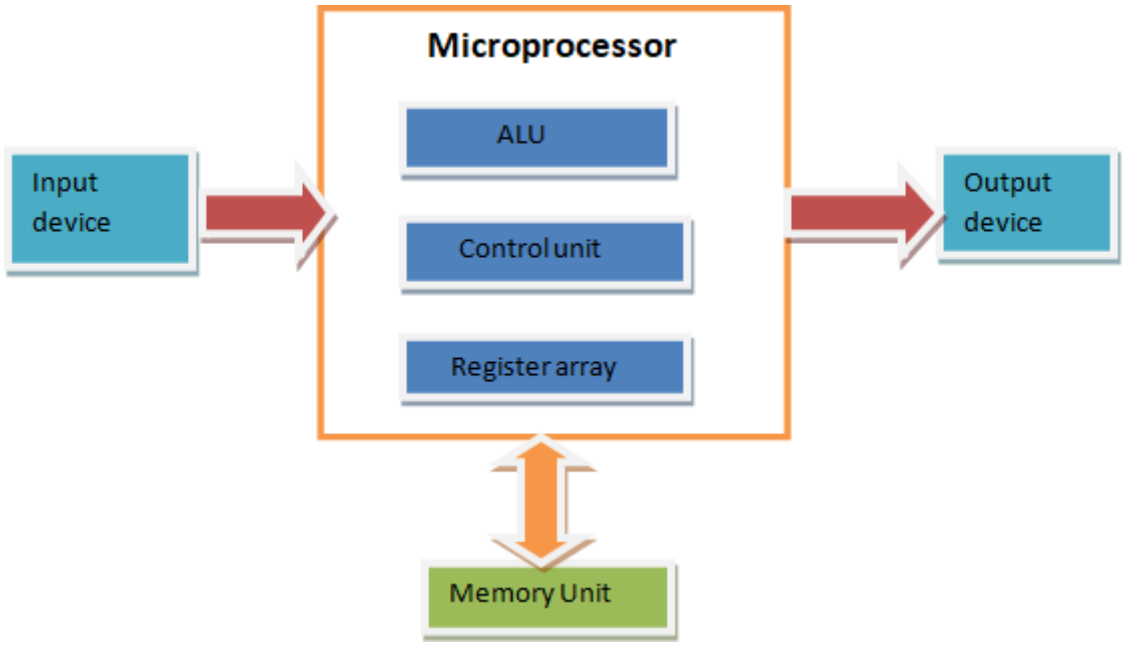


Figure 2.1.1. Block Diagram of a Microprocessor [3].

Difference between microcontrollers and microprocessors:

Microprocessor	Microcontroller
The microprocessor is the heart of a computer system.	The microcontroller is the heart of an embedded system.
Memory and input/output components must be connected externally.	The microcontroller has a processor along with internal memory and input/output components.
Due to external components, the overall power consumption is high. Therefore, it is not suitable for use with devices that operate on stored energy, such as batteries.	Since external components are minimal, the overall power consumption is lower and it can be used with devices that operate on stored energy, such as batteries.
The microprocessor has a smaller number of registers, which means a larger number of memory-oriented operations.	The microcontroller has a larger number of registers, which makes writing programs easier.
Microprocessors are based on the von Neumann architecture where program and data are stored in the same memory module.	Microcontrollers are based on the Harvard architecture where program memory and data memory are separate.
Functional blocks are ALU (Arithmetic Logic Unit), registers, timing, and control units.	Includes the functional blocks of a microprocessor, but additionally contains a timer, parallel input/output (I/O) ports, RAM, EPROM, ADC (Analog to Digital Converter), and DAC (Digital to Analog Converter).
Few types of bit manipulation instructions.	There are many types of bit manipulation instructions.

3. MICROCONTROLLER ATmega328P

The ATmega328P is a high-performance controller characterized by low power consumption. This microcontroller is 8-bit and uses the AVR RISC architecture. It is particularly valued as it is the most widespread among all AVR controllers, mainly due to its extensive application in ARDUINO boards [5].

The ATmega328P offers the following features: 32 kB of programmable Flash memory with read-while-write capability, 1 kB of EEPROM, 2 kB of SRAM memory, 23 general-purpose I/O lines, 32 general-purpose registers, three flexible timers/counters with compare modes, 1 serial programmable USART, 6-channel 10-bit ADC, a programmable watchdog timer with an internal oscillator, SPI (Serial Peripheral Interface) serial port, and six selectable software power-saving modes [6].

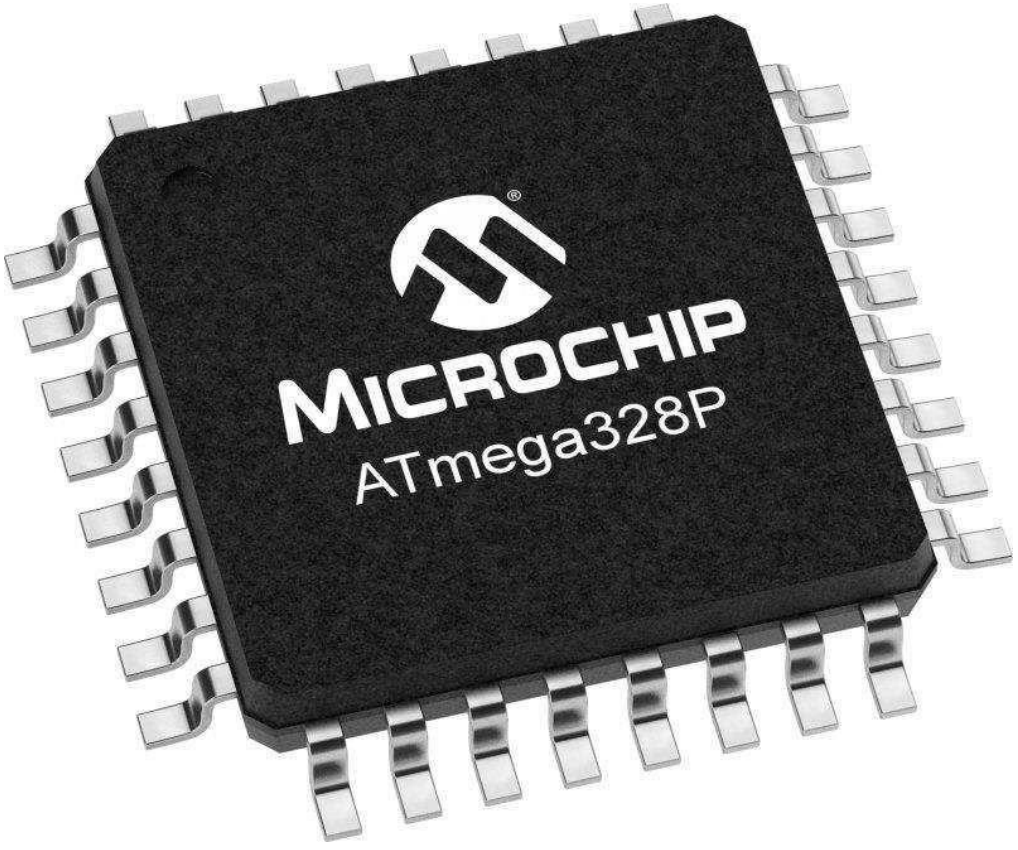


Figure 3.1. The ATmega328P Microcontroller.

ATmega328P offers a wide range of power-saving modes and features, making it an ideal choice for various applications in the embedded world. In Idle mode, the CPU halts while SRAM, timers/counters, SPI port, and interrupt system continue to operate. Power-Down mode preserves register contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset. Power-Save mode allows the asynchronous timer to continue operating, maintaining a time base while the rest of the device is idle. Analog-to-Digital Converter Noise Reduction mode stops the CPU and all I/O modules except the asynchronous timer and ADC to minimize noise during ADC conversion. Standby mode keeps the crystal/resonator oscillator running while the rest of the device sleeps, enabling fast startup with low power consumption. In Extended Standby mode, the main oscillator and asynchronous timer remain active [6].

With its flexibility, high energy efficiency, and cost-effectiveness, ATmega328P provides a solution applicable to many control and sensor systems. Its versatility, low power consumption, and extensive collection of peripheral devices make it reliable and widely applicable across various industries and projects.

3.1. PINOUT DIAGRAM

ATmega328P is housed in a 32-pin package. The pinout diagram of this microcontroller is shown in Figure 3.1.1.

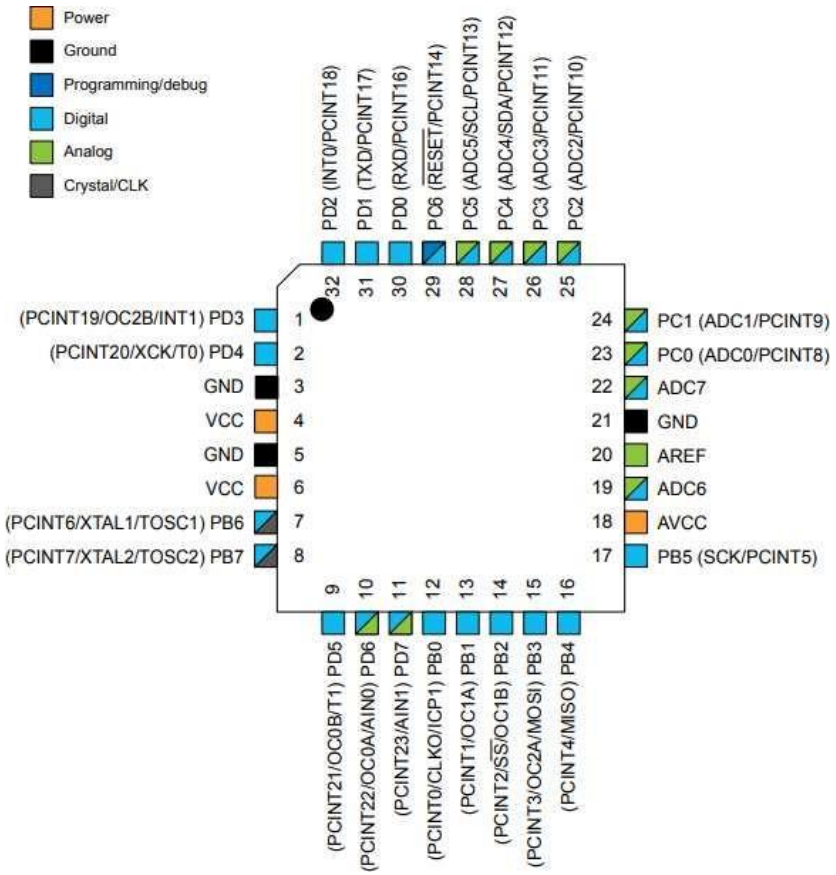


Figure 3.1.1. Pinout diagram of ATmega328P [6].

Pin description of microcontroller ATmega328P [6]:

a) VCC - Digital pin for supplying voltage

b) GND - Ground

c) AVCC - Pin for supplying voltage to the analog-to-digital converter (ADC)

d) AREF - Analog reference pin for the analog-to-digital converter (ADC)

e) ADC6 and ADC7 - Serve as analog inputs to the analog-to-digital converter (ADC). These pins are powered by an analog source and are used as 10-bit ADC channels.

f) PC6 (RESET) - If RSTDISBL is programmed, PC6 is used as an input/output pin. If RSTDISBL is not programmed, PC6 is used as a reset input. A low level on this pin lasting longer than the minimum pulse length will generate a reset. Shorter pulses do not guarantee a reset.

g) PD0 (RXD) - Input pin for serial communication.

h) PD1 (TXD) - Output pin for serial communication.

i) PD2 (INT0) - External interrupt source 0.

j) PD3 (INT1) - External interrupt source 1.

k) PD4 (XCK/T0) - Can serve as the count source for timer/counter 0, and as an external clock source for USART (Universal Synchronous and Asynchronous Receiver-Transmitter).

l) PD5 (T1/PWM/OC0B) - Can serve as the count source for timer/counter 1, as an external output for Output Compare Match B for timer/counter 0.

m) PD6 (AIN0/PWM/OC0A) - Can serve as the positive input for analog comparator 0, and as an external output for Output Compare Match A for timer/counter 0.

n) PD7 (AIN1) - Negative input for analog comparator 1.

o) PB0 (CLK0/ICP1) - Can serve as the Input Capture Pin for timer/counter 1, and as an output pin for Divided System Clock.

p) PB1 (OC1A/PWM) - Can serve as an external output for Output Compare Match A for timer/counter 1.

q) (OC1B/PWM/SS) - Can serve as an external output for Output Compare Match B for timer/counter 1, as a Slave Select input.

r) PB3 (OC2A/MOSI) - Can serve as the SPI Master data output, slave data input for SPI channel, as an external output for Output Compare Match A for timer/counter 2.

s) PB4 (MISO) - Master data input, data output for slave for SPI channel.

t) PB5 (SCK) - This pin serves as the master output clock and as the slave input clock for the SPI channel.
u) PB6 (XTAL1/TOSC1) - This pin is connected to one pin of the oscillator to provide an external clock pulse. When used as a clock pin, it cannot be used as an input/output pin.
v) PB7 (XTAL2/TOSC2) - This pin is connected to the second pin of the oscillator to provide an external clock pulse.
w) PC0-3 (ADC0-3) - These pins are used as ADC input channels.
x) PC4 (ADC4/SDA) - Serves as a two-wire serial interface for data input/output, and as an ADC input channel.
y) PC5 (ADC5/SCL) - Serves as a two-wire serial clock interface, and as an ADC input channel.

3.2. Architecture of ATmega328P Microcontroller

ATmega328P is an 8-bit microcontroller with RISC (Reduced Instruction Set Computing) architecture. This architecture is characterized by simple and efficient instructions, enabling faster program execution.

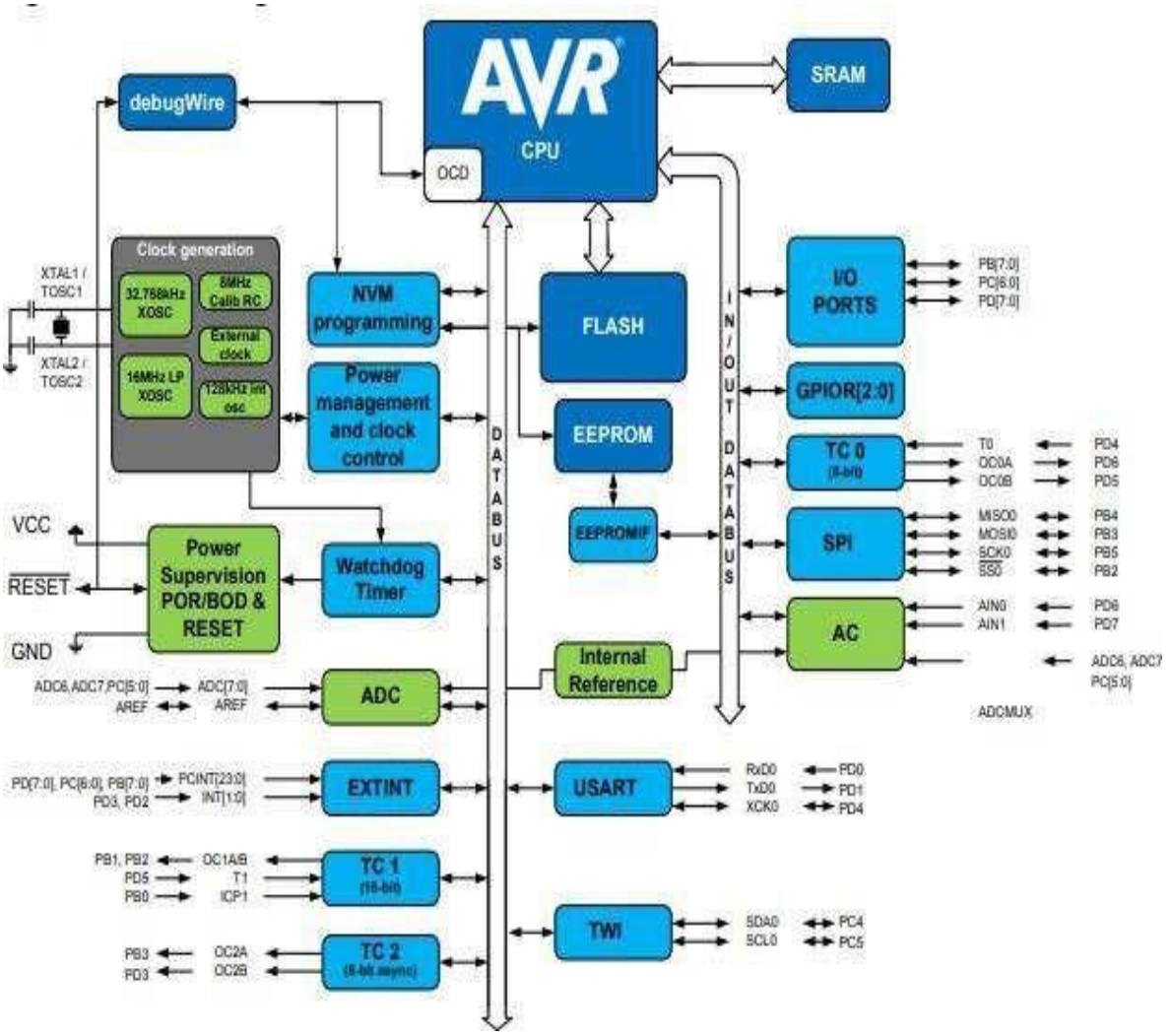


Figure 3.2.1. The architecture of the ATmega328P microcontroller [6].

3.2.1. Architecture of the AVR Core Central Processing Unit

The primary role of the CPU core is to ensure the proper execution of programs. To achieve this, the CPU must be able to access memory, perform arithmetic operations, manage peripheral devices, and handle interrupts.

AVR architecture employs the Harvard architecture, meaning that program and data are stored in separate memory spaces. This architecture allows data to be read from the program memory using specific instructions. During the execution of one instruction, the next instruction is already being fetched from the program memory. This concept enables instructions to execute in every clock cycle. The architecture of the AVR processor core is depicted in Figure 3.2.1.1.

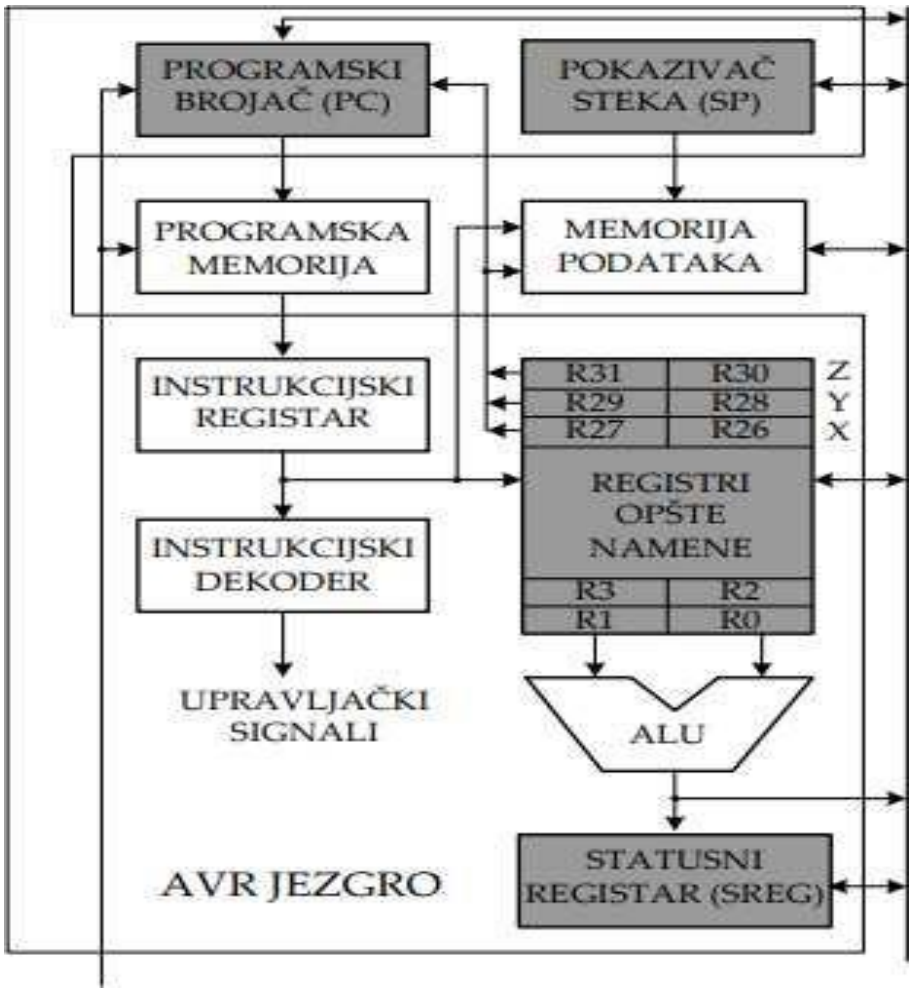


Figure 3.2.1.1. The architecture of the AVR processor core [7].

AVR uses a LOAD/STORE architecture where the ALU (Arithmetic Logic Unit) performs operations only on data (operands) stored in general-purpose registers. Instructions can address up to two operands located in general-purpose registers, with the result of the instruction execution stored at the address of the first operand. Alternatively, the second operand can be an immediate value (constant) from the current instruction. The general-purpose registers consist of 32 mapped registers R0 to R31 with single-cycle access time, enabling ALU operations to execute in a single clock cycle [7].

During interrupts and subroutine calls, the return Program Counter (PC) address is stored on the stack. The stack is efficiently allocated in the general data SRAM space, and therefore, the size of the stack is limited only by the total size of SRAM and its usage. All user programs must initialize the Stack Pointer (SP) in the reset routine (before subroutine calls or interrupts are executed) [6].

3.2.2. Memory in AVR Architecture

AVR architecture features two main memory spaces: Data Memory and Program Memory where instructions are stored.

Program Memory, also known as Flash memory, is used to store program instructions. Flash memory is non-volatile, meaning its data remains preserved even when the microcontroller loses power. Unlike data memory space, Flash memory is not as fast to access, and individual bytes cannot be easily modified, making it unsuitable for storing frequently changing variables. However, besides storing application code, Flash memory is suitable for storing constants.

Data memory space, unlike flash memory, is not composed of a single type of memory. Instead, it consists of a continuous mapping of addresses across multiple types of memory.

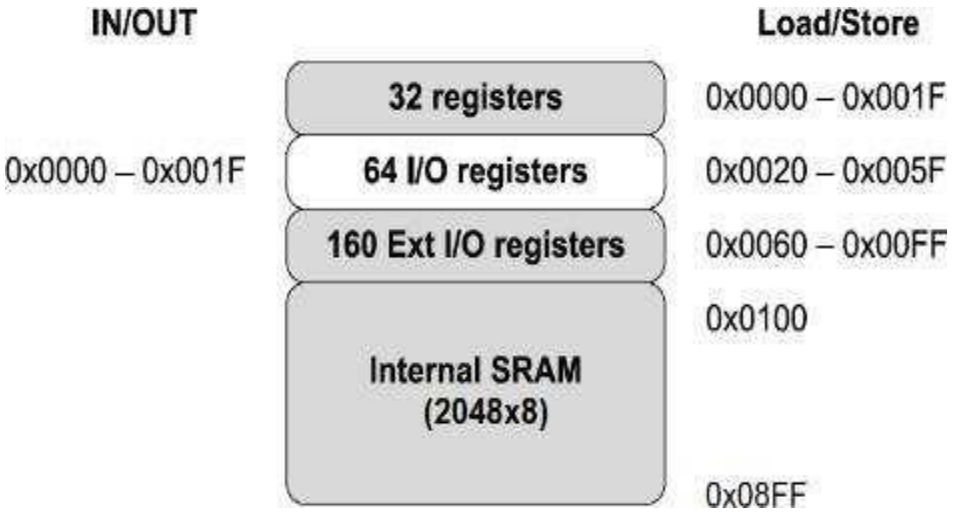


Figure 3.2.2.1. Structure of Data Memory Space [8].

Data Memory Space consists of four parts:

1) Register File starts at the beginning of the data memory space and contains 32 general-purpose registers directly connected to the ALU. These registers serve as operands for most instructions.

2) I/O Registers are locations within the data memory space that control specific functions of the microcontroller. All data direction registers and ports are housed in the I/O Registers section. Because they are frequently accessed, special instructions allow rapid access and modification of I/O Registers.

3) Extended I/O Registers differ only in that they do not support all the same set, reset, or copy instructions as standard I/O Registers. However, their function and purpose are identical.

4) SRAM (Static Random Access Memory) represents the final part of the data memory space. SRAM is a volatile memory, so its contents are not guaranteed upon system startup. However, unlike flash memory, individual bytes can be written to SRAM, and access times are faster, making it more suitable for storing temporary values.

The AVR architecture also includes EEPROM (Electrically Erasable Programmable Read-Only Memory), which is non-volatile. Unlike flash memory, EEPROM allows individual byte writes, although memory access is slower. EEPROM is best suited for storing data that needs to be available upon startup but may be modified during program execution and will be used the next time the chip is restarted.

4. ARDUINO

Arduino is an open-source hardware and software platform designed for anyone interested in developing interactive devices and applications within a specific interactive development environment. This board features a microcontroller that can be programmed to control objects in the physical world. By using sensors and input signals, Arduino can react and communicate with various output devices such as LEDs, motors, and displays. Arduino has become an extremely popular choice among creators who want to develop interactive hardware projects, thanks to its flexibility and affordability.

4.1. ARDUINO UNO R3 DEVELOPMENT SYSTEM

Arduino UNO R3 is an ideal board for gaining basic knowledge in electronics and programming. This flexible development board is equipped with the well-known ATmega328P and ATmega16U2 processors.

Figure 4.1.1. Arduino UNO R3

The board is equipped with 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. What particularly sets it apart from previous models is the fact that it does not use the FTDI USB-to-serial driver chip; instead, it integrates the Atmega16U2 programmed as a USB-to-serial converter.

4.1.1. POWER SUPPLY

The Arduino Uno can be powered via a USB connection or external power. The power source is automatically selected. External (non-USB) power can come from an AC-to-DC adapter or a battery. When using a battery, cables can be connected to the Gnd and Vin pins on the POWER connector. The board can operate with external power ranging from 6V to 20V. However, if the supply voltage is less than 7V, the 5V output pin may deliver less than 5V, potentially causing instability. Using more than 12V can overheat the voltage regulator and damage the board. The recommended voltage range is from 7V to 12V.

4.1.2. I/O PINS

Each of the 14 digital pins on the Uno board can function as either input or output using functions like pinMode(), digitalWrite(), and digitalRead(). They operate at a voltage of 5V. Each pin can supply or receive a maximum of 40 mA and has an internal pull-up resistor (default off) with a value between 20-50 kΩ.

The Uno also features 6 analog inputs, labeled A0 to A5, each providing 10-bit resolution. By default, they measure from ground (GND) to 5V, but you can adjust the upper limit of their range using the AREF pin and the analogReference() function.

4.1.3. ARDUINO IDE

DE, an abbreviation for "Integrated Development Environment," represents the official software introduced by Arduino.cc. Its primary purpose is to edit, compile, and upload code to Arduino devices. The software can be downloaded from the main Arduino website at https://www.arduino.cc/en/software. Figure 4.1.3.1 shows the download window for the Arduino IDE software environment.

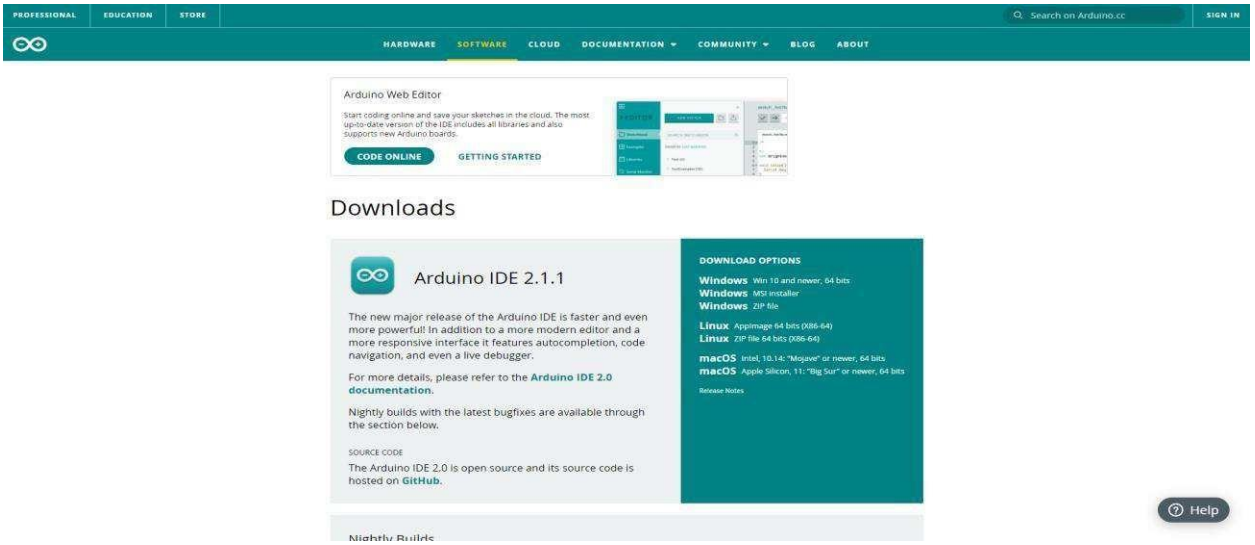


Figure 4.1.3.1. Display of the Arduino IDE software environment download window.

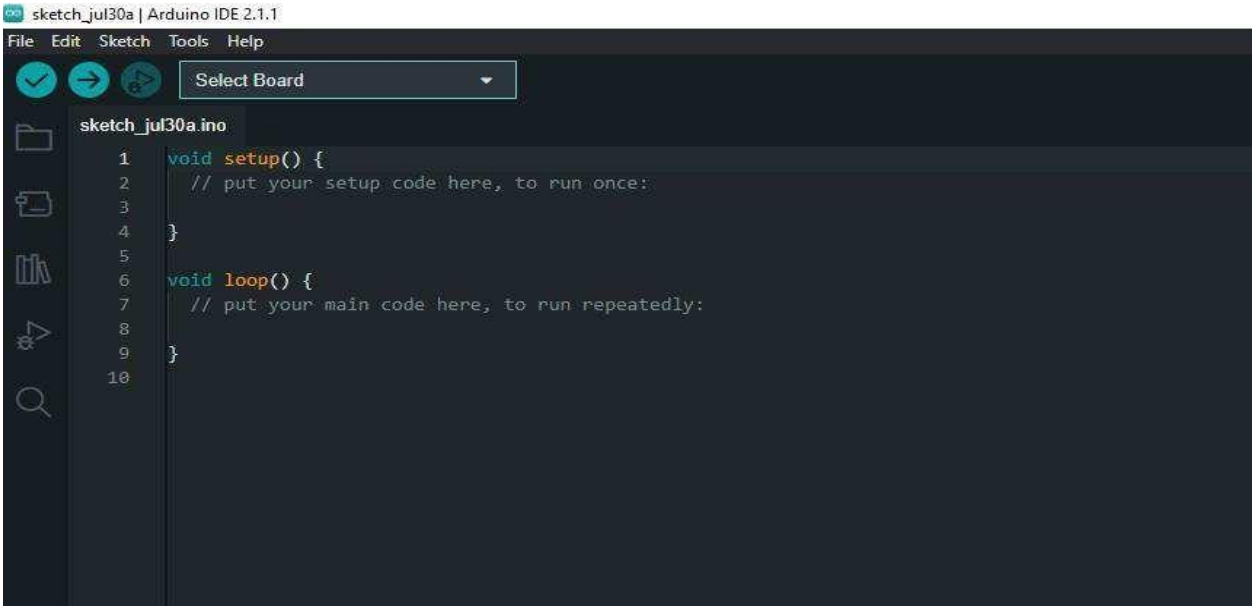


Figure 4.1.3.2. Arduino IDE software environment.

Nearly all Arduino modules are compatible with this software environment, which is open-source and easily available for installation, enabling quick code compilation even on the go. Arduino IDE is used for writing and compiling code for Arduino modules, streamlining this process even for individuals without prior technical knowledge. The main code, known as a "sketch," created on the IDE platform, ultimately generates a Hex file that is uploaded to the controller on the board. This environment supports the C and C++ languages.

5. Node.js

Node.js, short for "Node JavaScript," is an open-source platform that allows executing JavaScript code on the server-side. Introduced by the Node.js Foundation, Node.js is a powerful tool for developing server applications. The software can be downloaded from the main Node.js website at https://nodejs.org/. Figure 5.1.1. shows the Node.js software package download window.

Node.js is compatible with numerous libraries and modules, making it highly flexible for developing various types of applications. This platform is open-source and readily available for installation, enabling fast code execution and scalable server applications. Node.js utilizes the V8 JavaScript engine, the same engine that powers Google Chrome, ensuring high performance and speed.

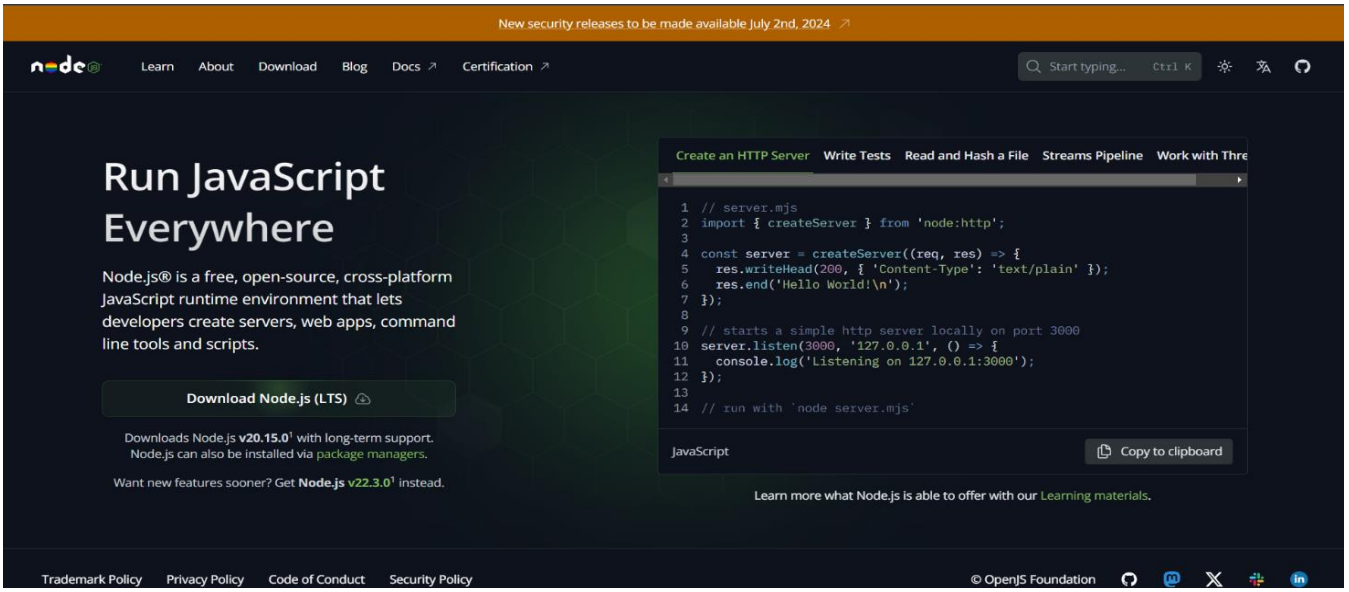


Figure 5.1.1. Display of the Node.js software package download window.

6. React

React, s druge strane, je biblioteka za izgradnju korisničkih interfejsa, razvijena od strane Facebooka. React omogućava razvoj interaktivnih korisničkih interfejsa sa komponentnom arhitekturom, koja olakšava upravljanje stanjem i logikom aplikacije. Softver se može preuzeti sa glavne React web stranice https://reactjs.org/. Na slici 6.1.1. prikazan je prozor za preuzimanje React softverske biblioteke.

React is compatible with various tools and libraries, enabling the development of complex user interfaces with minimal code. React uses JSX, a JavaScript extension that allows writing HTML within JavaScript code, making component creation easier. React also supports the development of applications that are easily scalable and maintainable, facilitating the creation of interfaces that are fast and responsive.

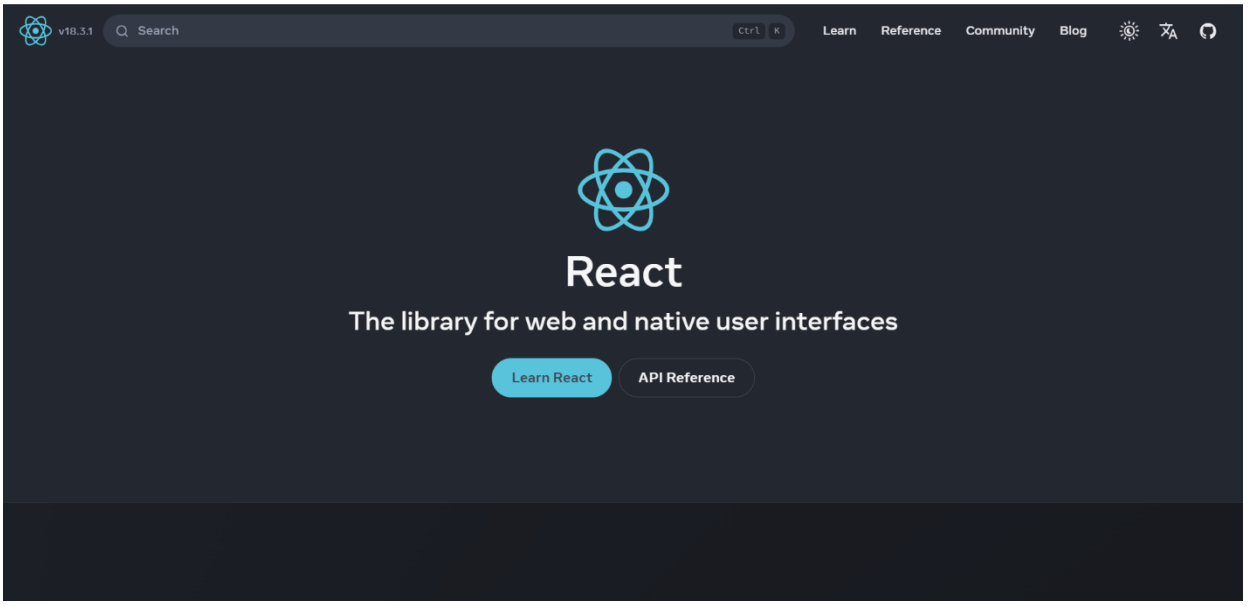


Figure 6.1.1. Display of the React software library download window.

7. PROJECT ASSIGNMENT

In the following section, the device functionality process will be presented, along with detailed analyses of implementation both in hardware and software terms.

7.1. FUNCTIONAL REQUIREMENTS

To consider the project functional, the following conditions need to be met:

a) When the device is powered on, wait a few seconds until the blue LED on the front of the device lights up (indicating successful Wi-Fi connection and readiness of weight sensors).

b) Water check and refill:

- Every 30 seconds, check the water weight in the bowl.
- If the water weight is less than one-third of the total expected amount in the bowl, refill the water to the expected amount (200g).
- There is also a safety button below the bowl; if the bowl is not in place, skip the water refill and turn on the red LED until the bowl is returned to its place.

c) Selection of food quantity, first meal time, and meal repeat interval via web application:

- The user enters the desired amount of food in grams via the keyboard.
- Chooses feeding time and repeat interval.
- Then confirm the request via a confirmation button.

d) Serving the first meal:

- When the user-selected time arrives, the stepper motor rotates 360° and dispenses a portion of food (8-12g), repeating the process until the food weight sensor measures the selected quantity.

e) Serving subsequent meals:

- Depending on the user-selected time interval, calculate the time for the next meal.
- When it's time for the next meal, first check the current amount of food in the bowl. If it's less than the selected amount, activate the motor again to refill the food. Otherwise (food is not eaten), skip that meal and wait for the next one.

7.2. SCHEME AND HARDWARE IMPLEMENTATION

The following components are required for the hardware implementation of the project:

- Arduino UNO R3 development board
- WiFi module ESP-01
- Two weight measurement sensors (up to 1kg) with HX-711 drivers
- Two DC/DC converters STEPDOWN LM259
- Self-priming pump
- Relay module HW-279
- Step motor 28BYJ-48 with ULN2003 driver
- D/C connector
- Push button
- Two LEDs (red and blue)
- Three resistors (2x 220Ω, 1x 330Ω)
- Switch
- Protoboard and connecting wires (jumper wires).

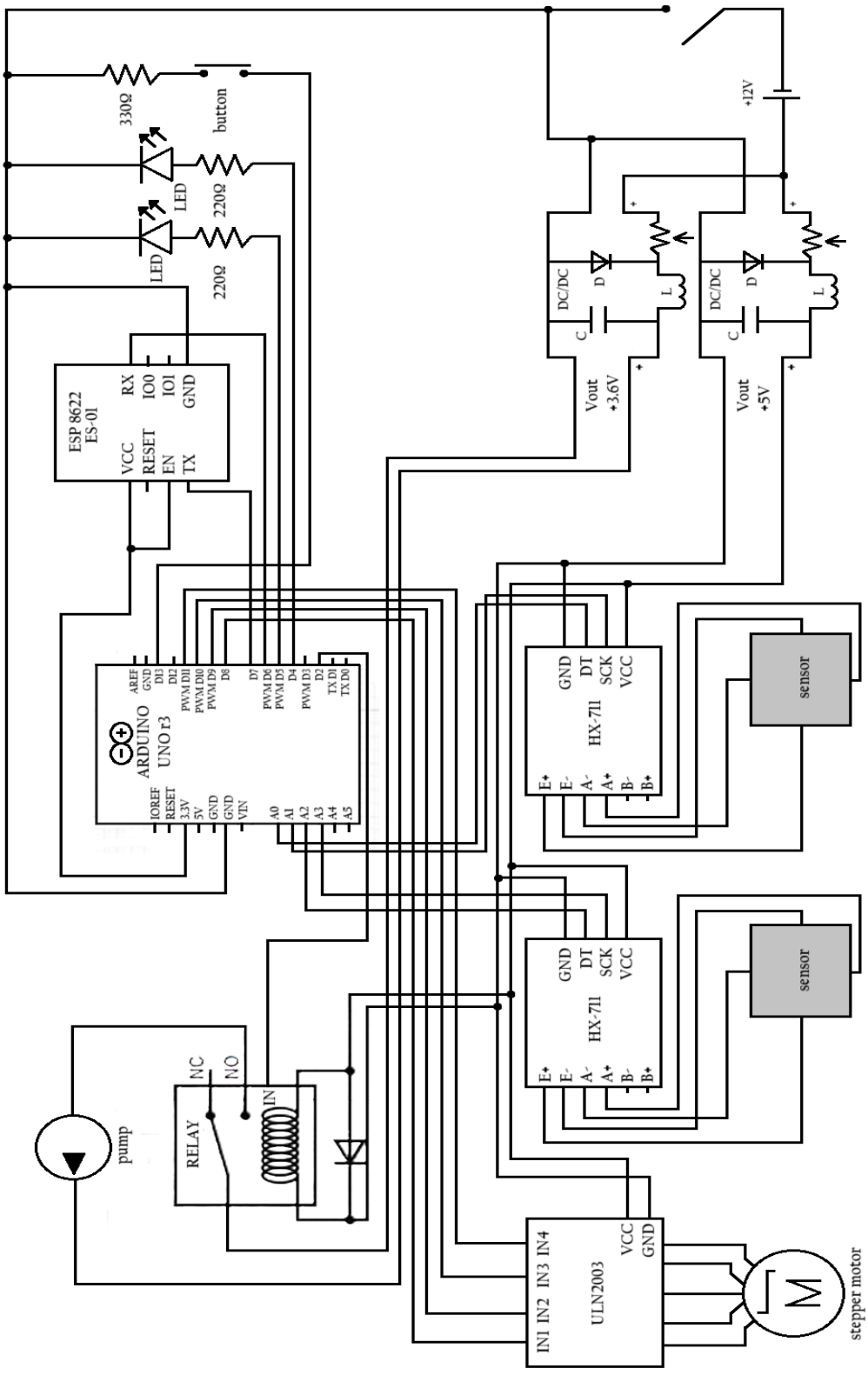


Figure 7.2.1. An electrical schematic of the project assignment.

7.2.1. The WiFi module ESP-01 and the blue LED.

The ESP-01 Wi-Fi module is a popular wireless module often used in Arduino Uno projects for connecting to the internet. It utilizes the ESP8266 chip, enabling wireless communication via Wi-Fi networks. This module is small and compact, making it ideal for integration into various projects requiring wireless connectivity.

ESP-01 can be used for various purposes such as collecting sensor data and sending it to a server, controlling devices over the internet, or creating a wireless network for communication between different devices. Modules like ESP-01 are popular due to their low cost, ease of use, and the large community that provides support and resources for project development.

In this project, the ESP-01 module is used to connect the Arduino Uno device to a Wi-Fi network. Once successfully connected, the module can send and receive data over the internet, allowing the user to remotely control the project or collect real-time data.

The blue LED activates to indicate a successful Wi-Fi connection.

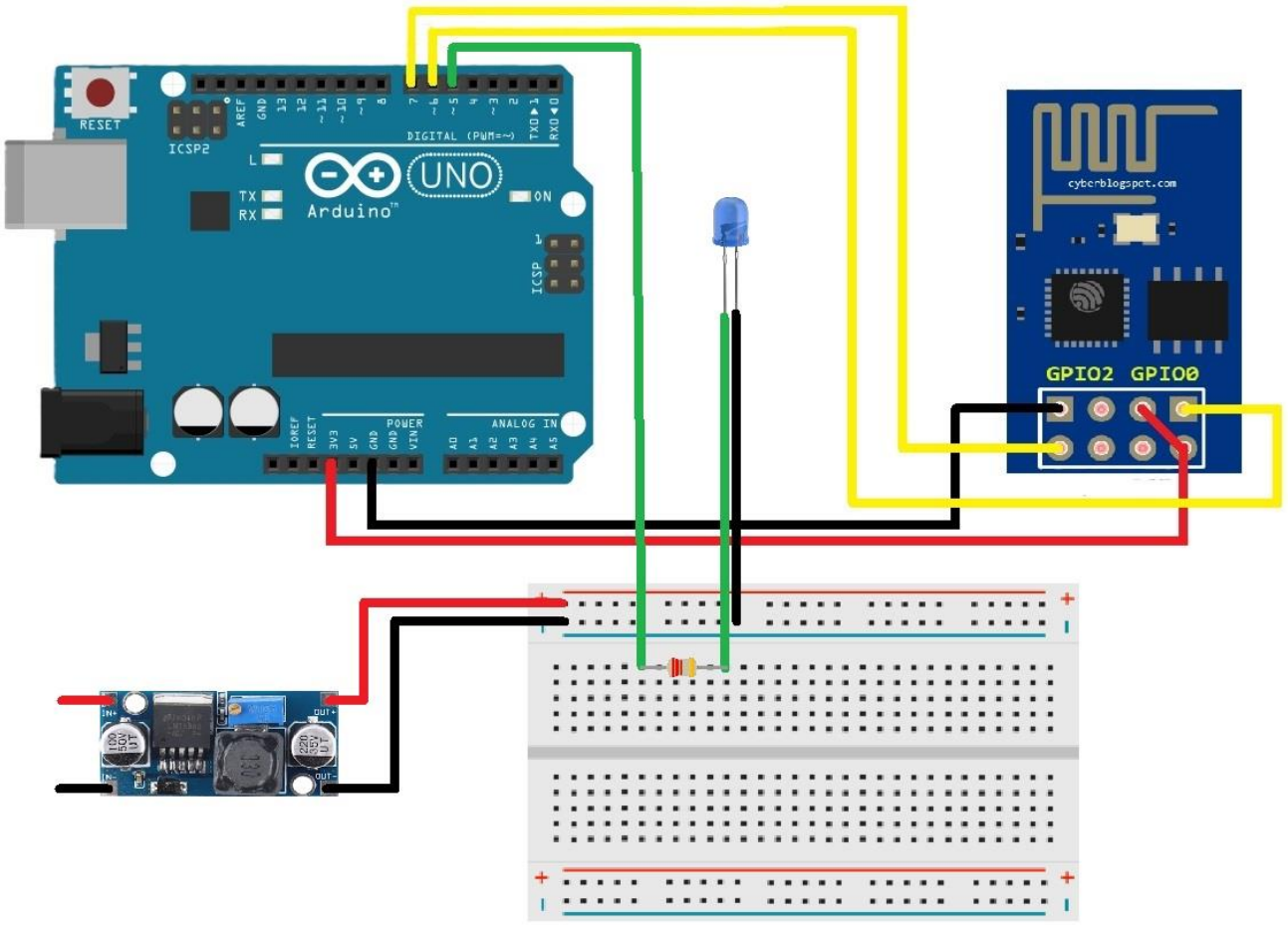


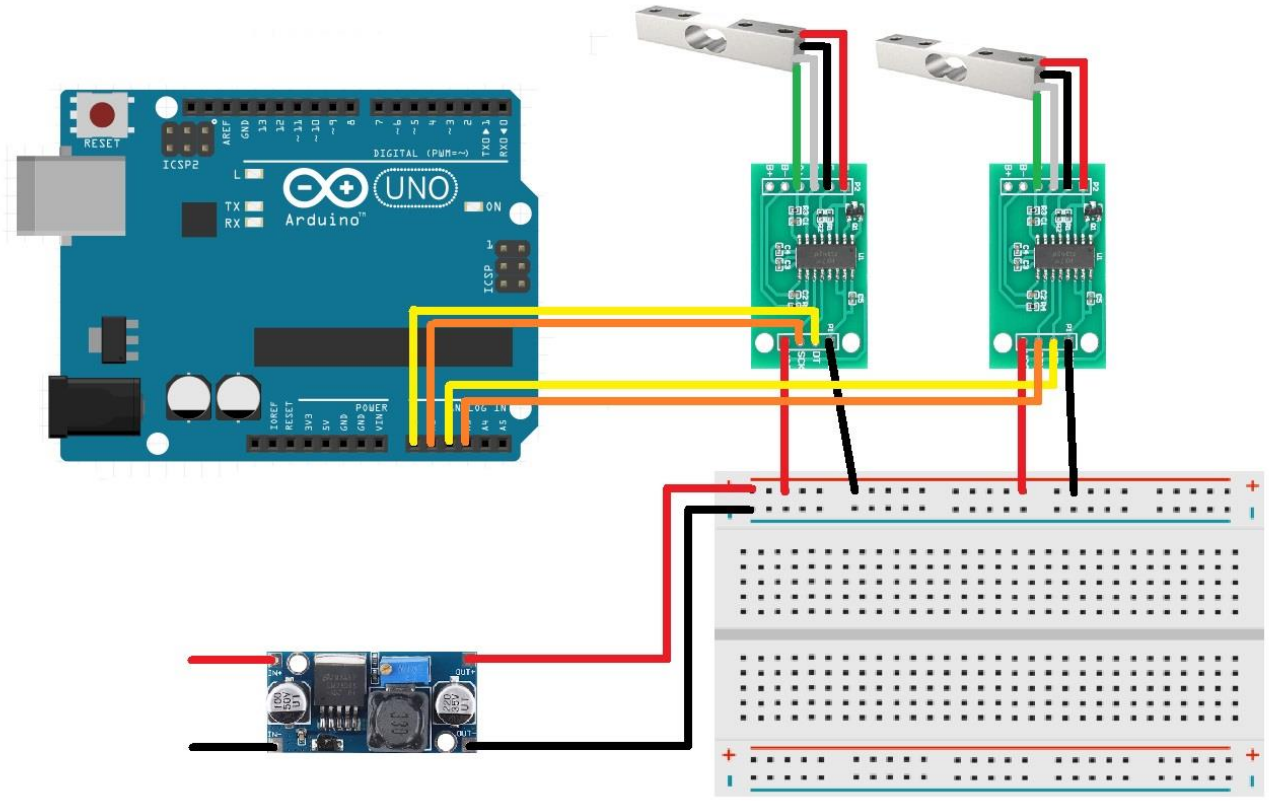
Figure 7.2.1.1. Schematic diagram of connecting the WiFi module ESP-01 to Arduino UNO R3.

7.2.2. Two weight sensors (up to 1kg) with HX-711 drivers

Two weight sensors (up to 1kg) with HX-711 drivers are key components in Arduino Uno projects that require precise weight measurement. These sensors are designed to detect small changes in weight with a high degree of accuracy.

The sensors are connected to HX-711 drivers, which are analog-to-digital converters specifically designed for such sensors. HX-711 drivers take analog signals from the weight sensors and convert them into digital signals that Arduino Uno can process. These drivers enable high measurement precision with minimal noise, which is crucial for accurate results.

In this project, they are used to measure the total weight of water and food in the bowls.



Slika 7.2.2.1. Schematic diagram of connecting two weight sensors and HX-711 drivers to Arduino UNO R3.

7.2.3. Safety button and red LED

The button is a simple electronic device used to interrupt or establish an electrical circuit. When pressed, the button either allows or interrupts the flow of current, sending a signal to the electronic system to perform a specific action.

In this project, this button functions as a switch that is activated when the bowl is properly placed. When the bowl is not in its place or has been moved, the button breaks the circuit, signaling the Arduino to stop the water replenishment process. This setup protects the system from potential spills or empty refills, enhancing the efficiency and safety of the entire system.

The red LED activates when the button is released, indicating that the bowl is not in its designated position.

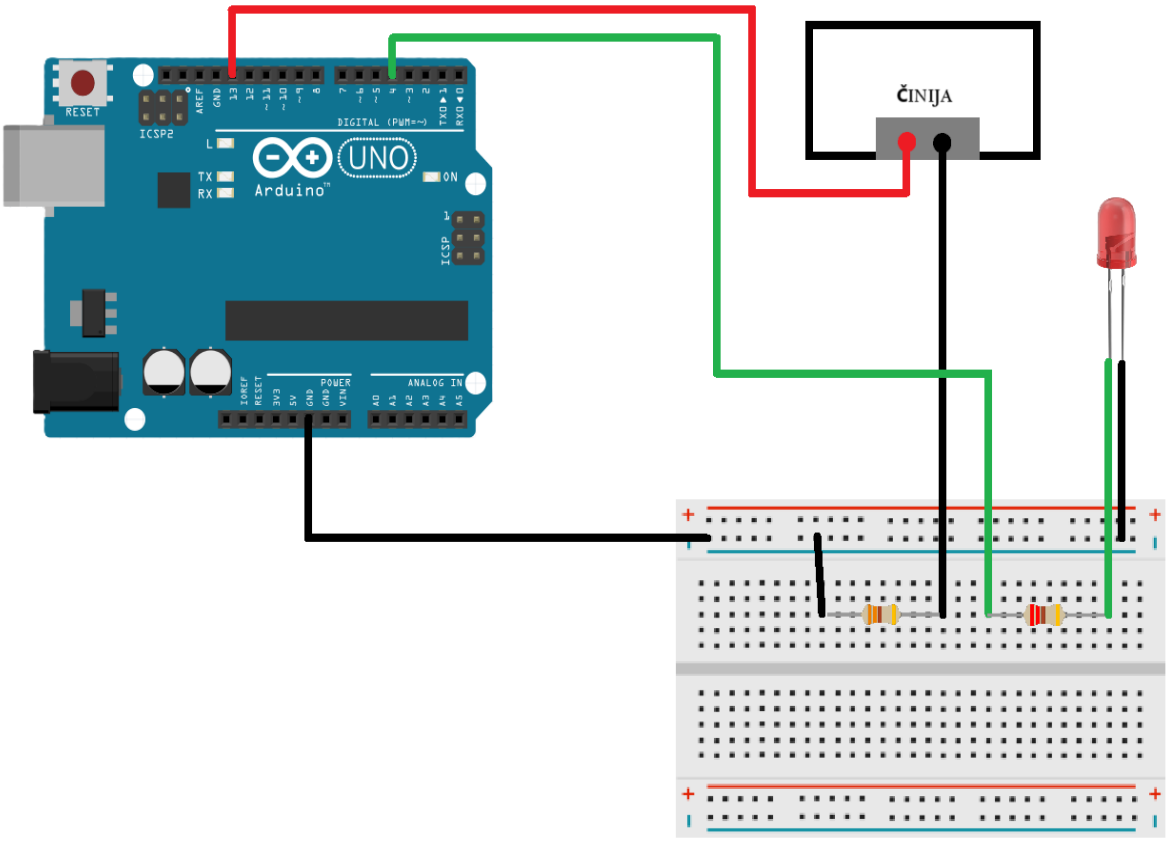


Figure 7.2.3.1. Schematic diagram of safety button to Arduino UNO R3.

7.2.4. Stepper Motor 28BYJ-48 with ULN2003 Driver

The 28BYJ-48 Stepper Motor with ULN2003 Driver is commonly used in Arduino Uno projects due to its precision and ability to control position. A stepper motor converts electrical impulses into discrete rotational steps, enabling precise positioning.

The ULN2003 driver facilitates easy connection and control of the stepper motor via Arduino. It provides the necessary power and enables control over the motor's direction and speed.

In this project, the motor is used to control a food dispensing disc. The disc, attached to the motor shaft, rotates to precisely dispense food.

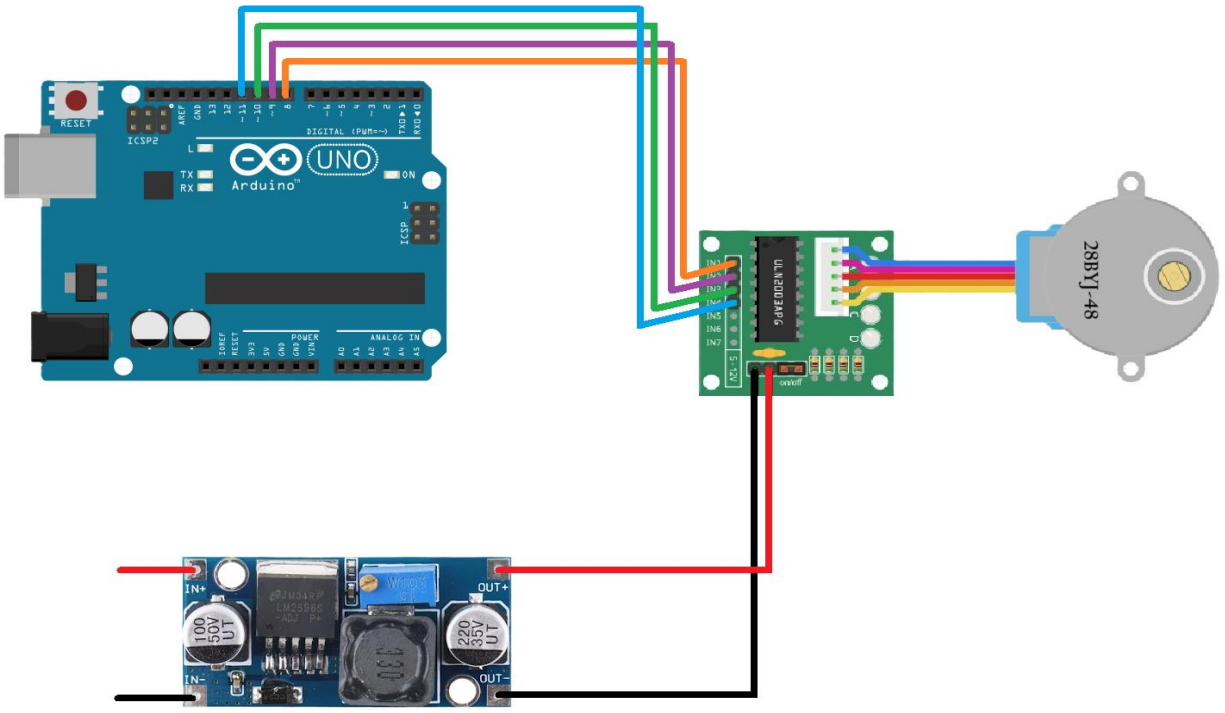


Figure 7.2.4.1. Schematic diagram of Step motor 28BYJ-48 with ULN2003 driver to Arduino UNO R.

7.2.5. Relay and self-priming water pump

A relay is an electronic switch used to control high currents or voltages using a low-voltage signal from a microcontroller such as Arduino Uno. It enables the control of various devices and components that require higher currents than what the microcontroller can directly provide.

A self-priming water pump is a device designed to be submerged in water and used to transfer liquids from one reservoir to another. This pump is activated when the relay closes the circuit, allowing current to flow to the pump.

In this project, the relay is used to control a self-priming water pump that operates at 3.6V.

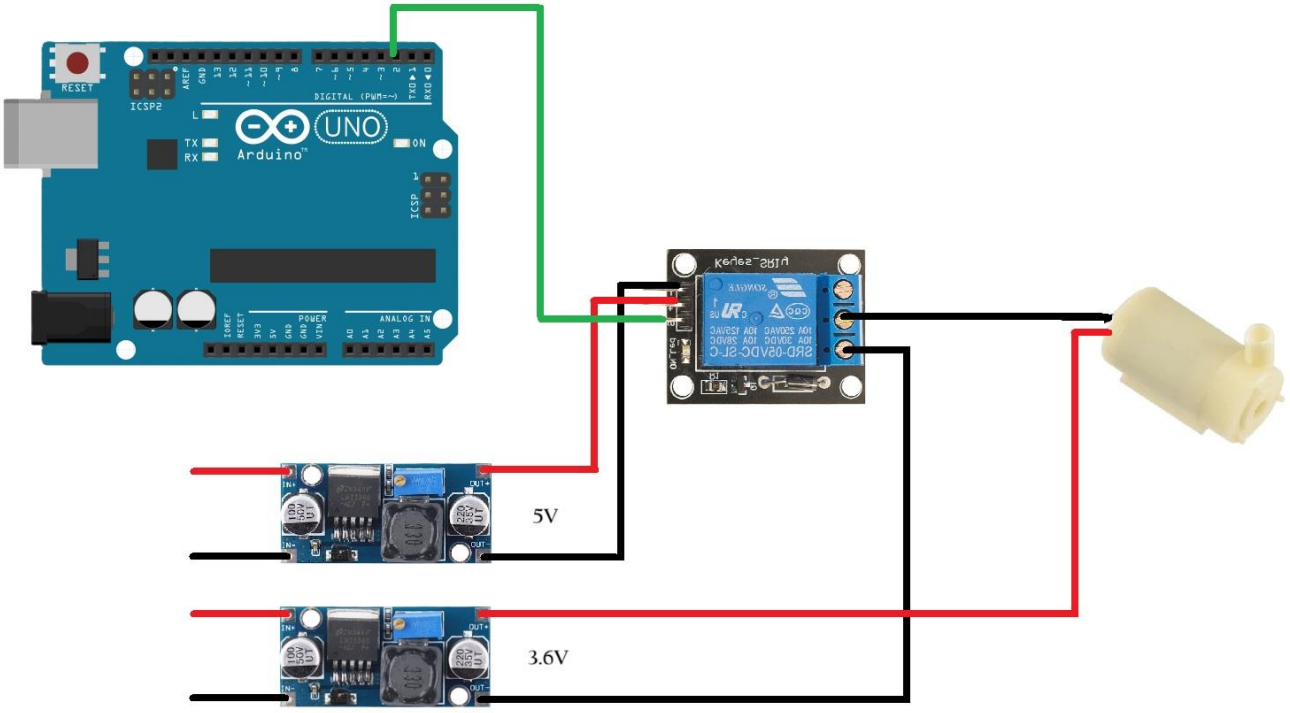


Figure 7.2.5.1. Schematic diagram of relay self-priming water pump to Arduino UNO R.

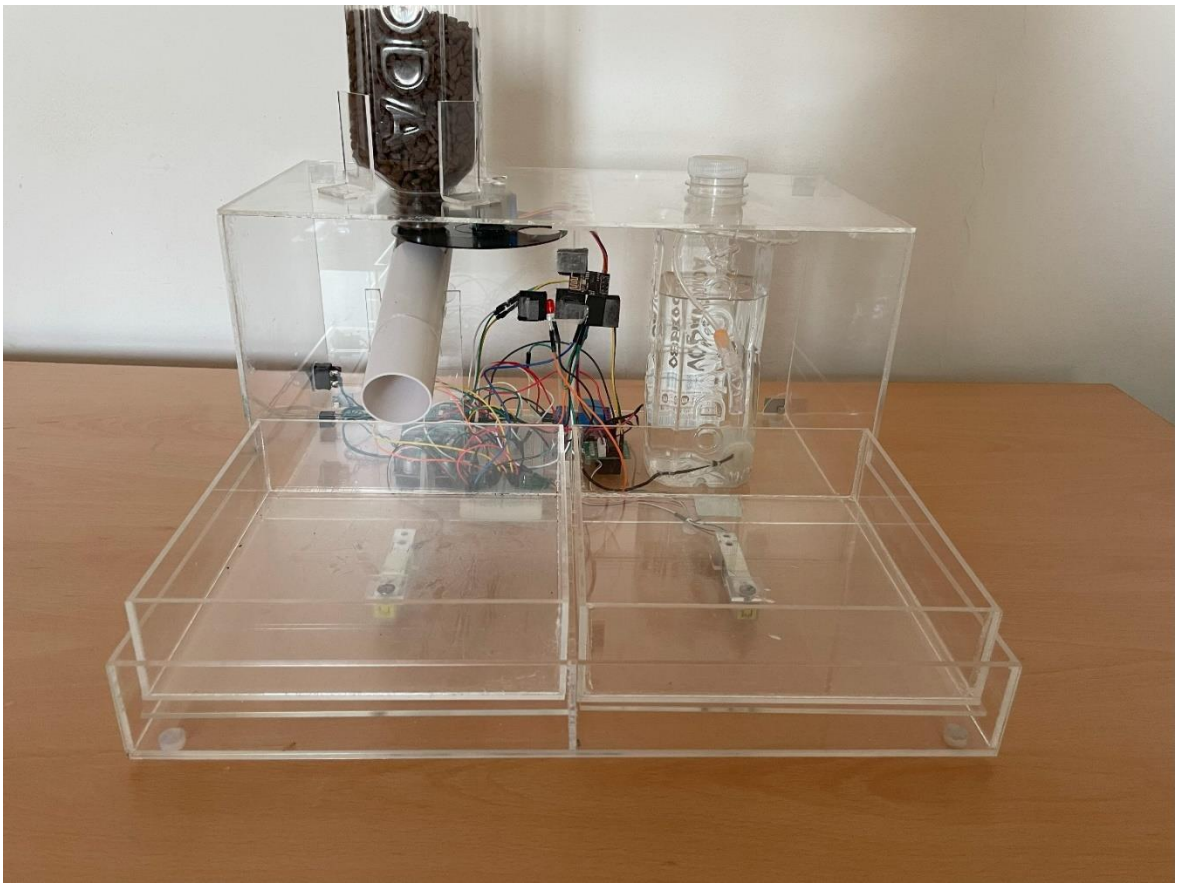


Figure7.2.6. Hardware implementation of the project.

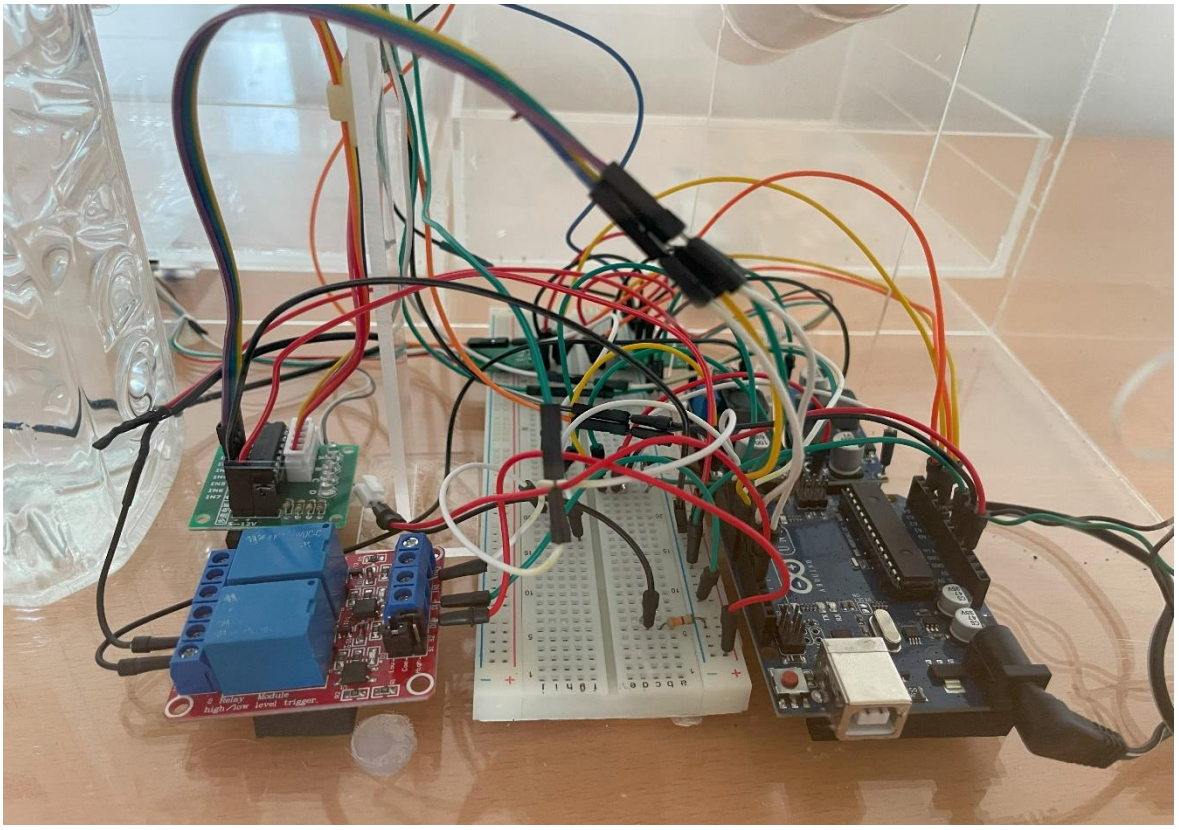


Figure 7.2.7 Hardware implementation of the project.

7.3. SOFTWARE IMPLEMENTATION

7.3.1. Arduino Software Implementation

In the following sections, some of the most important algorithms used in the software implementation of the project will be illustrated.

In the setup() part of the code, the initial state of the relay, motor, LEDs, button, scales, and the operating mode of the WiFi module (as a server) is set up. This is shown by the following algorithm.

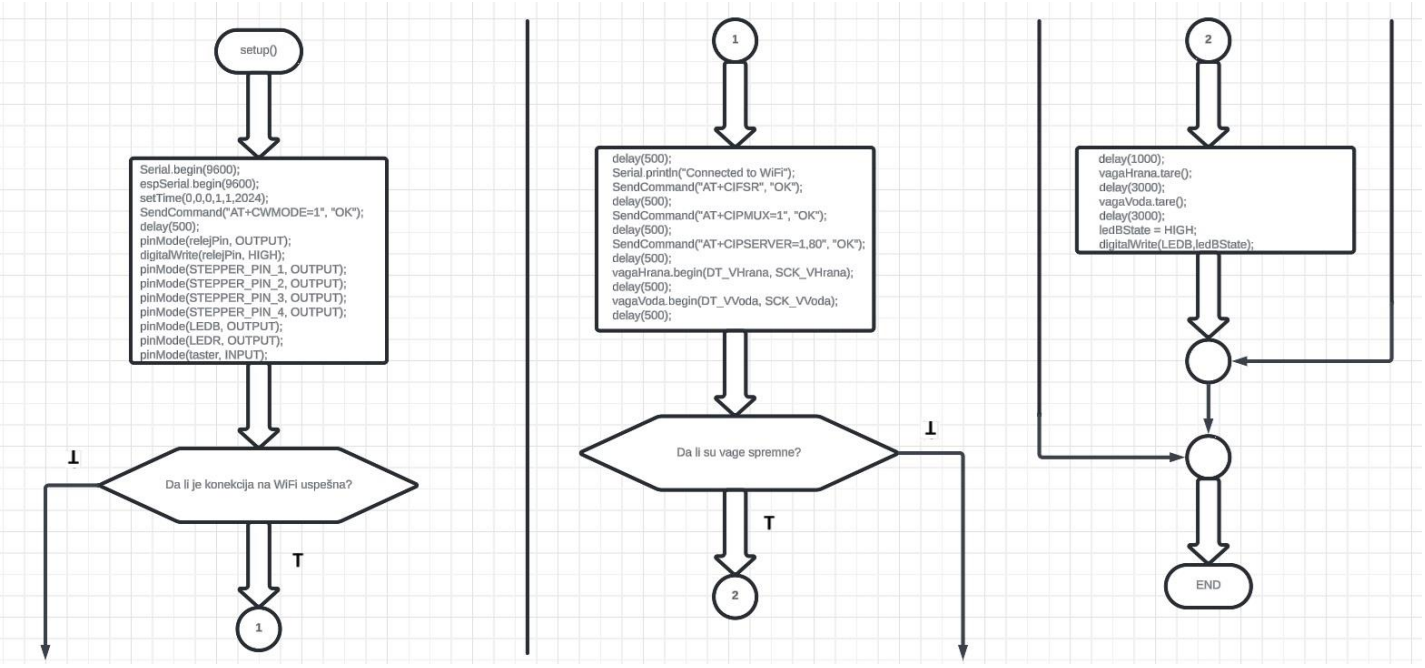


Figure 7.3.1.1. Algorithm outline of the setup() function.

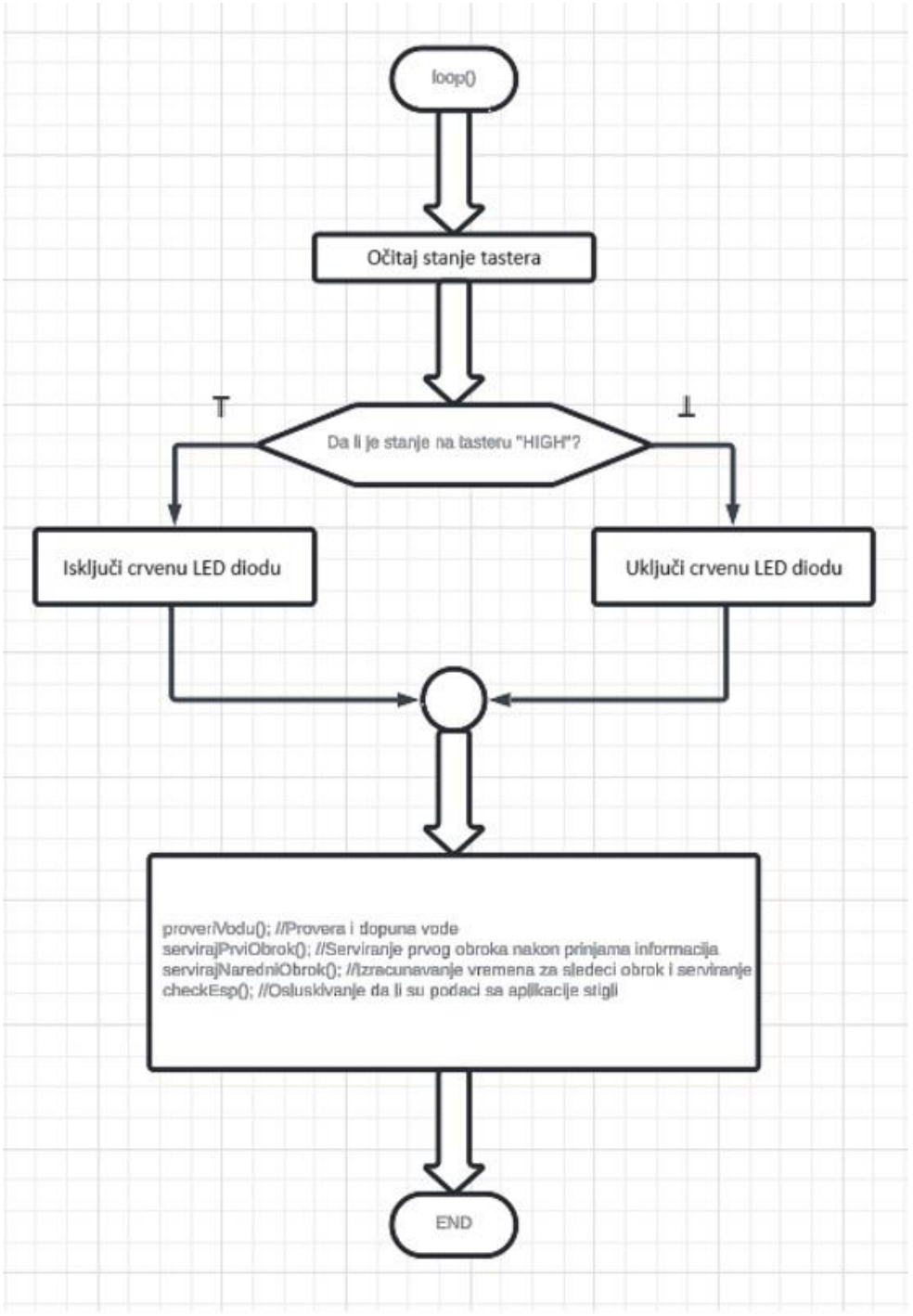


Figure 7.3.1.2. Algorithm outline of the main function.

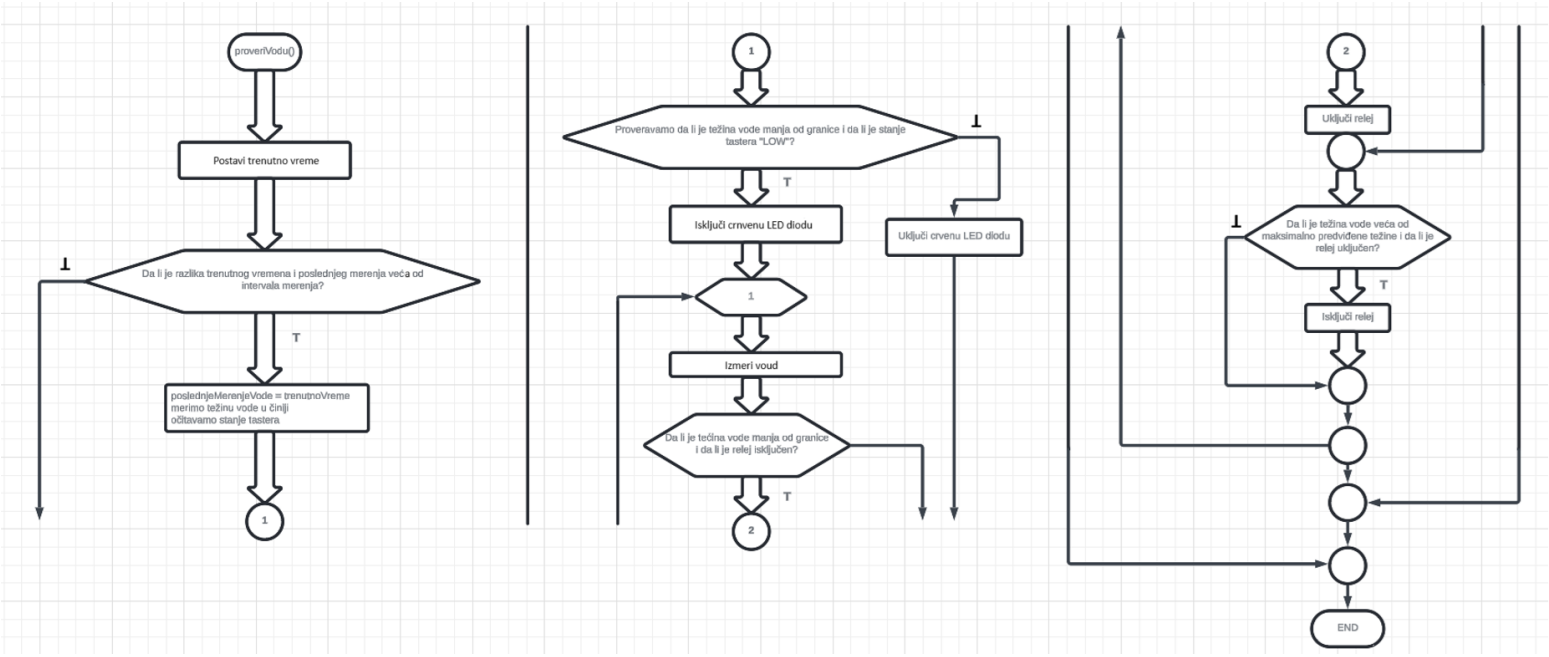


Figure 7.3.1.3. Algorithm outline of the proveriVodu() (checkWater) function.

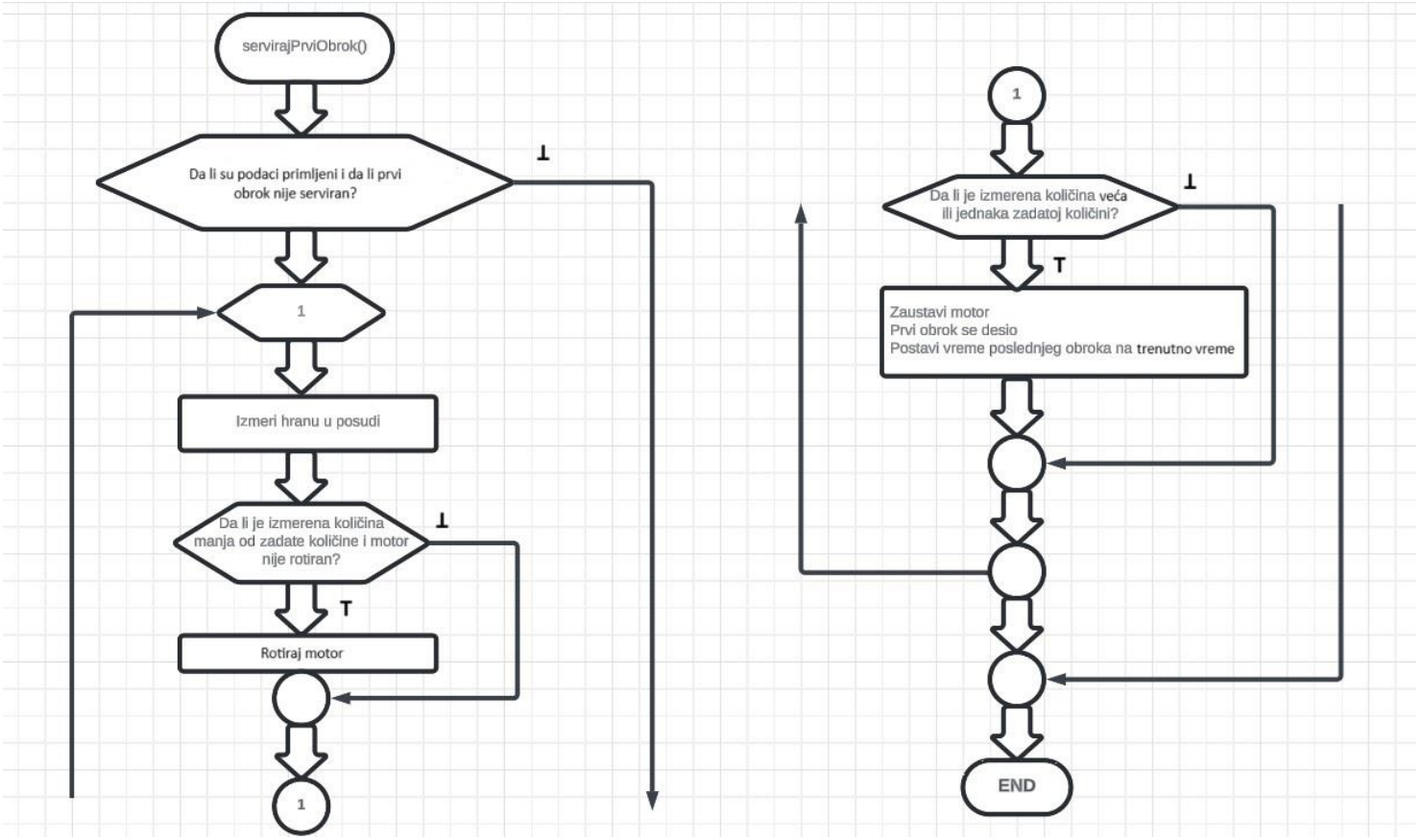


Figure 7.3.1.4. Algorithm outline of the servirajPrviObrok() (serveFirstMeal) function.

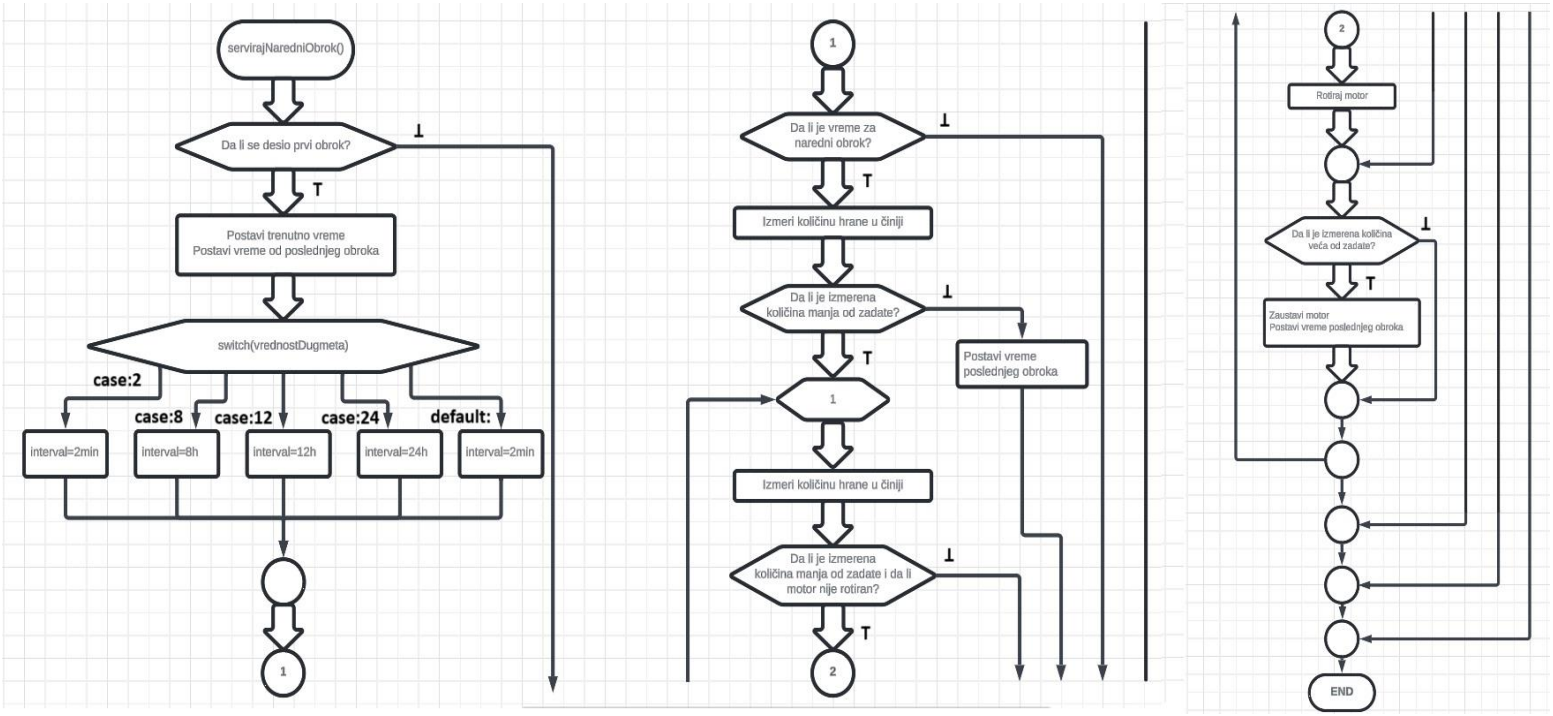


Figure 7.3.1.5. Algorithm outline of the servirajNaredniObrok() (serveNextMeal) function.

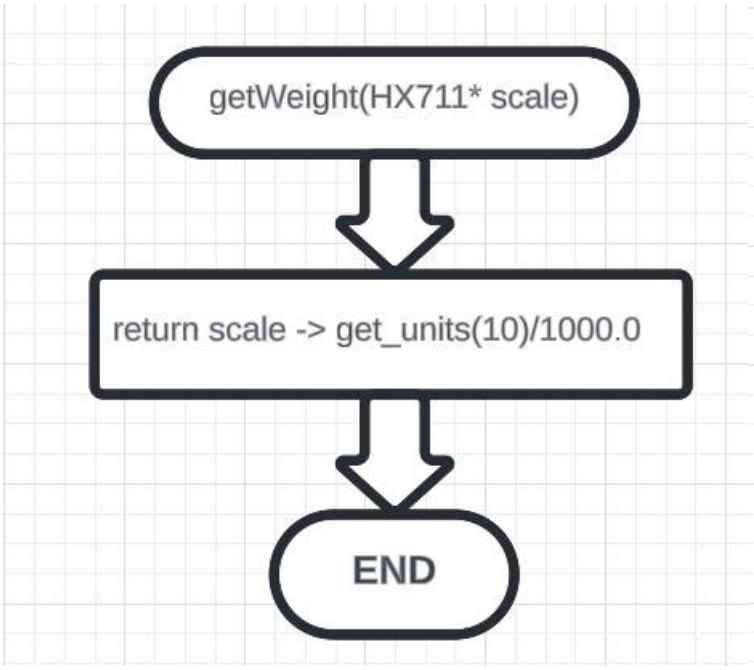


Figure 7.3.1.6. Algorithm outline of the getWeight() function.

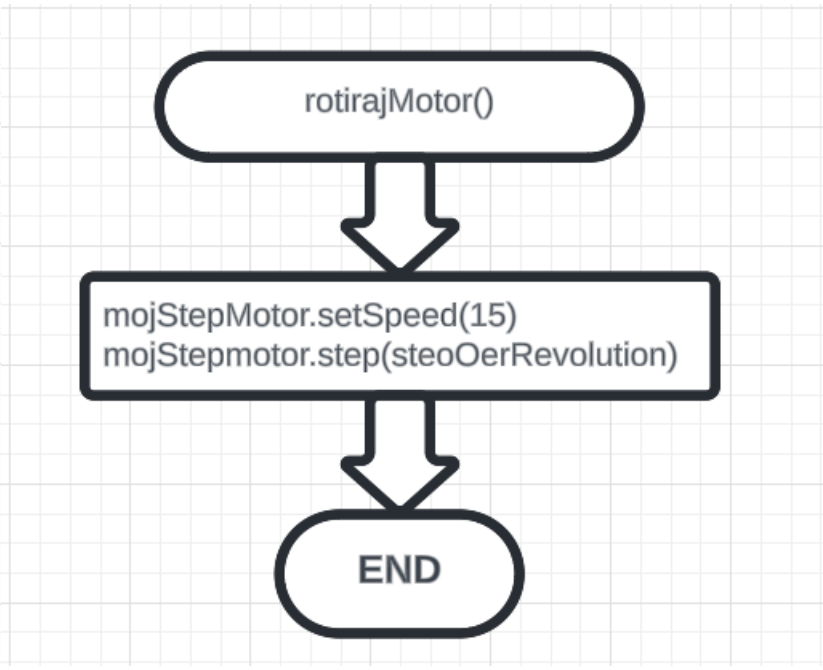


Figure 7.3.1.7. Algorithm outline of the rotirajMotor() (rotateMotor) function.

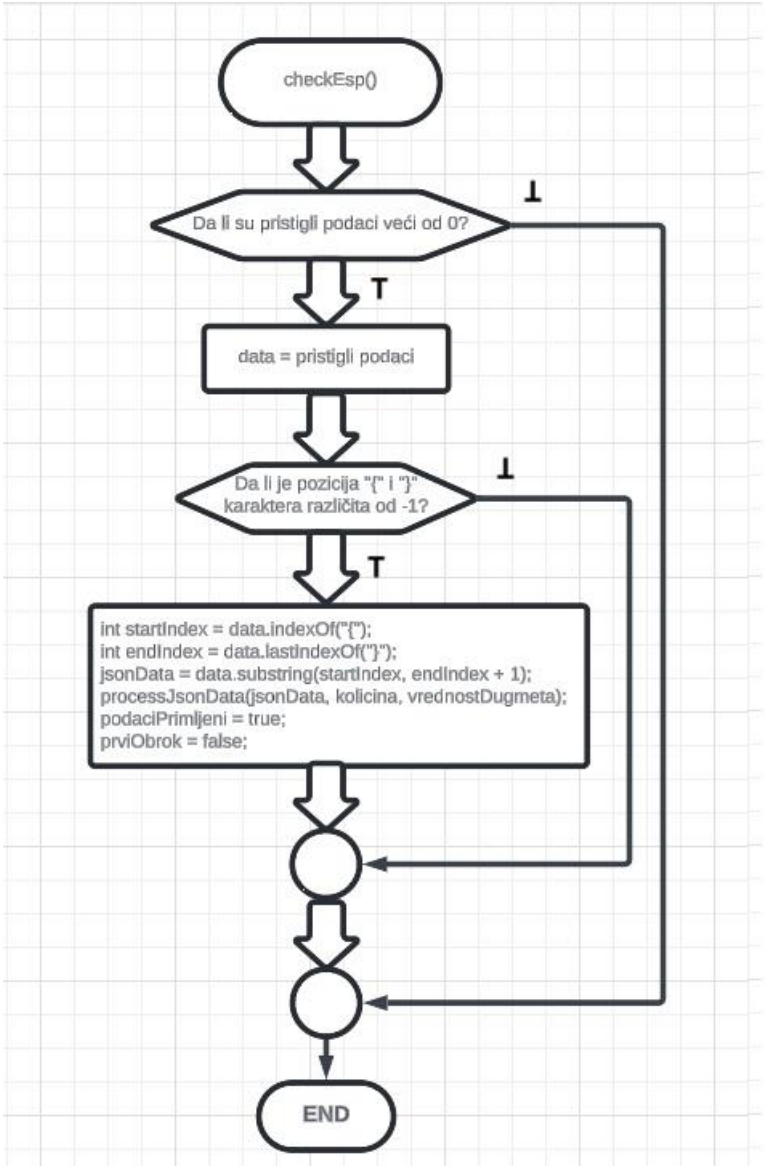


Figure 7.3.1.8. Algorithm outline of the checkEsp() function.

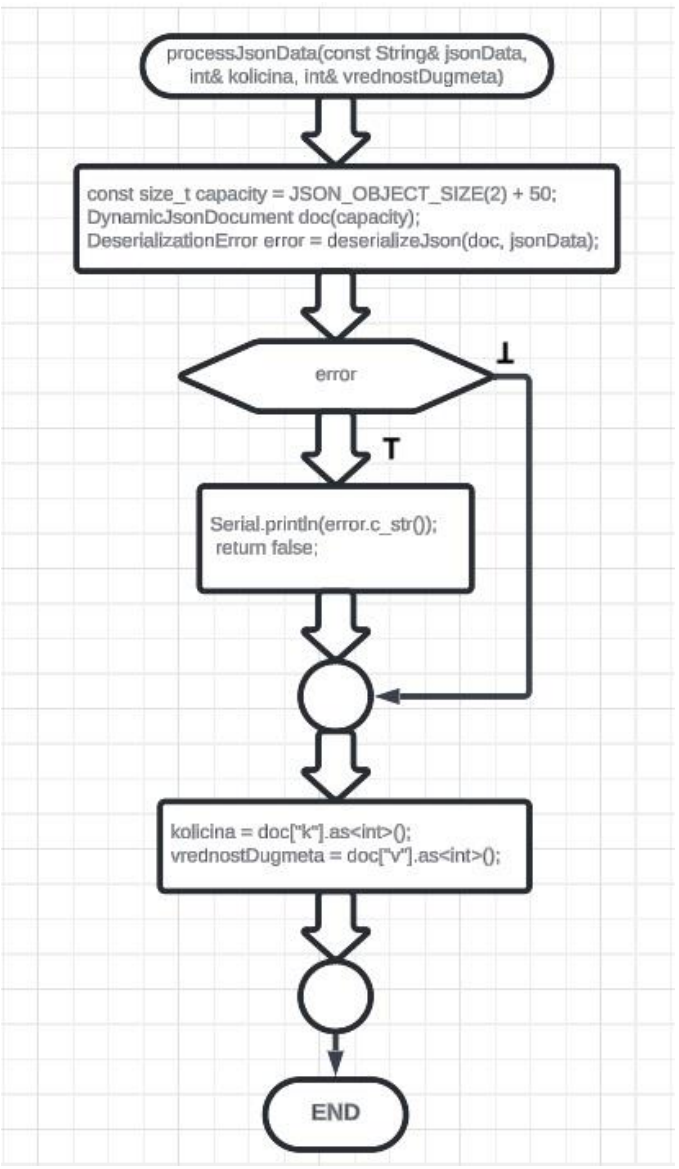


Figure 7.3.1.9. Algorithm outline of the procesJsonData() function.

The Arduino software configuration is implemented with the following code:

```
<SoftwareSerial.h>
<ArduinoJson.h>
"HX711.h"
<Stepper.h>
<TimeLib.h>

STEPPER_PIN_1 8
STEPPER_PIN_2 9
STEPPER_PIN_3 10
STEPPER_PIN_4 11
const int stepsPerRevolution = 2048;
Stepper mojStepMotor(stepsPerRevolution, 8, 10, 9, 11);
int step_number = 0;

SoftwareSerial espSerial(7, 6);
String jsonData;

HX711 vagaHrana;
HX711 vagaVoda;
const int DT_VHrana = A0;
const int SCK_VHrana = A1;
const int DT_VVoda = A2;
const int SCK_VVoda = A3;

const int LEDB = 5;
int ledBState = LOW;
const int LEDR = 4;
const int taster = 13;
int tasterState = 0;

unsigned long poslednjeMerenjeVode = 0;
const unsigned long intervalMerenjaVode = 30 * 1000; // 2 minuta u milisekundama
const float maxTezinaVode = 200.0; // Maksimalna težina vode u gramima, prilagodi
prema stvarnim vrednostima
const float granicaVode = maxTezinaVode / 3.0;
const int relejPin = 2;
bool relejUkljucen = false;

//Funkcija za pracenje odgovora sa ESP8622
boolean echoFind(String keyword) {
  byte current_char = 0;
  byte keyword_length = keyword.length();
  long deadline = millis() + 5000;
```

```
  while (millis() < deadline) {
    if (espSerial.available()) {
      char ch = espSerial.read();
      Serial.write(ch);

      if (ch == keyword[current_char]) {
        if (++current_char == keyword_length) {
          Serial.println();
          return true;
        }
      } else {
        current_char = 0;
      }
    }
  }
  return false;
}

//Funkcija za slanje AT komande na ESP8622
boolean SendCommand(String cmd, String ack) {
  espSerial.println(cmd);
  return echoFind(ack);
}

void setup() {
  Serial.begin(9600);
  espSerial.begin(9600);

  setTime(0,0,0,1,1,2024);
  if (SendCommand("AT+RST", "ready")) {
    delay(5000);
    Serial.println("ESP8266 reset successfully");
    blinkBlue(3,500);
  } else {
    Serial.println("Failed to reset ESP8266");
  }
  SendCommand(" =1", "OK");
  delay(500);

  pinMode(relejPin, OUTPUT);
  digitalWrite(relejPin, HIGH);

  pinMode(STEPPER_PIN_1, OUTPUT);
  pinMode(STEPPER_PIN_2, OUTPUT);
  pinMode(STEPPER_PIN_3, OUTPUT);
  pinMode(STEPPER_PIN_4, OUTPUT);
```

```
pinMode(LEDB, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(taster, INPUT);

  if (SendCommand("AT+CWJAP=\"Redmi Note 9\",\"ilija1999\"", "OK")) {
    delay(500);
    Serial.println("Connected to WiFi");
    blinkBlue(5,200);
    SendCommand("AT+CIFSR", "OK");
    delay(500);
    SendCommand("AT+CIPMUX=1", "OK");
    delay(500);
    SendCommand("AT+CIPSERVER=1,80", "OK");
    delay(500);

    vagaHrana.begin(DT_VHrana, SCK_VHrana);
    delay(500);
    vagaVoda.begin(DT_VVoda, SCK_VVoda);
    delay(500);
    if((vagaHrana.is_ready())&&(vagaVoda.is_ready())){
      delay(1000);
      vagaHrana.tare();
      delay(3000);
      vagaVoda.tare();
      delay(3000);
      Serial.println("Vage su spremne!");
      ledBState = HIGH;
      digitalWrite(LEDB,ledBState);
    }
    else{
      digitalWrite(LEDR, HIGH);
      Serial.println("Vage nisu spremne!");
    }

  } else {
    Serial.println("Failed to connect to WiFi");
    blinkRed(5,500);
  }
}
bool motorRotiran = false;
unsigned long motorRotationStartTime = 0;
bool podaciPrimljeni = false;
bool prviObrok;
int kolicina;
int vrednostDugmeta;
unsigned long vremePoslednjegObroka = 0;
```

```
void checkEsp() {
  if (espSerial.available() > 0) {
    String data = espSerial.readString();
    Serial.println("Received data:");
    Serial.println(data);

    if (data.indexOf("{") != -1 && data.lastIndexOf("}") != -1) {
      int startIndex = data.indexOf("{");
      int endIndex = data.lastIndexOf("}");
      jsonData = data.substring(startIndex, endIndex + 1);

      Serial.println("Linija sa JSON podacima: ");
      Serial.println(jsonData);

      processJsonData(jsonData, kolicina, vrednostDugmeta);

      Serial.print("Količina: ");
      Serial.println(String(kolicina));

      Serial.print("Vrednost dugmeta: ");
      Serial.println(vrednostDugmeta);

      podaciPrimljeni = true;
      prviObrok = false;
    }
  }
}

void proveriVodu() {
  unsigned long trenutnoVreme = millis();
  if (trenutnoVreme - poslednjeMerenjeVode >= intervalMerenjaVode) {
    poslednjeMerenjeVode = trenutnoVreme;
    float tezinaVode = getWeight(&vagaVoda);
    tasterState = digitalRead(taster);
    delay(500);
    Serial.println("Težina vode: " + String(tezinaVode) + "g");
    if(tezinaVode < granicaVode && tasterState == LOW){
      digitalWrite(LEDR,LOW);
      while (1) {
        tezinaVode = getWeight(&vagaVoda);
        delay(500);
        Serial.println(String(tezinaVode) + "g");
        if (tezinaVode < granicaVode && !relejUkljucen) {
          digitalWrite(relejPin, LOW); // Uključi relej
          relejUkljucen = true;
        }
```

```
        if (tezinaVode >= maxTezinaVode && relejUkljucen) {
          digitalWrite(relejPin, HIGH); // Isključi relej
          relejUkljucen = false;
          Serial.println("Relej isključen, pumpa staje...");
          break;
        }
      }
    }
    else{
      digitalWrite(LEDR, HIGH);
    }
  }
}
void servirajPrviObrok(){
  // Merenje težine sa senzora i rotacija motora samo ako su podaci primljeni
  if (podaciPrimljeni && !prviObrok) {
     while (1) {
        float tezina_VH = getWeight(&vagaHrana);
        Serial.println(String(tezina_VH) + "g");
        if (tezina_VH < kolicina && !motorRotiran) {
          rotirajMotor();
        }
        if (tezina_VH >= kolicina) {
          motorRotiran = false;
          prviObrok = true;
          vremePoslednjegObroka = now();
          delay(1000);
          break;
        }
     }
  }
}
int interval;
void servirajNaredniObrok(){
  if (prviObrok) {
    unsigned long trenutnoVreme = now();
    unsigned long vremeOdPoslednjegObroka = trenutnoVreme - vremePoslednjegObroka;

    // Postavljanje intervala na osnovu vrednostiDugmeta
    switch (vrednostDugmeta) {
      case 2:
        interval = 2 * 60; // 2 minuta u sekundama
        break;
      case 8:
interval = 8 * 3600; // 8 sati u sekundama
        break;
```

```
case 12:
        interval = 12 * 3600; // 12 sati u sekundama
        break;
      case 24:
        interval = 24 * 3600; // 24 sata u sekundama
        break;
      default:
        interval = 8 * 3600;
        break;
    }
    if (vremeOdPoslednjegObroka >= interval) {
      Serial.println("Vreme za sledeći obrok!");
      float tezina_VH = getWeight(&vagaHrana);
      if(tezina_VH <= kolicina)
      {
        while (1) {
          tezina_VH = getWeight(&vagaHrana);
          Serial.println(String(tezina_VH) + "g");
          if (tezina_VH < kolicina && !motorRotiran) {
            rotirajMotor();
          }
          if (tezina_VH >= kolicina) {
            motorRotiran = false;
            vremePoslednjegObroka = now();
            delay(1000);
            break;
          }
        }
      }else{
        vremePoslednjegObroka = now();
      }
    }
  }
}
void loop() {
  tasterState = digitalRead(taster);
  if (tasterState == HIGH) {
    digitalWrite(LEDR, HIGH);
  } else {
    digitalWrite(LEDR, LOW);
  }
  proveriVodu(); //Provera i dopuna vode
  servirajPrviObrok(); //Serviranje prvog obroka nakon prinjama informacija
  servirajNaredniObrok(); //Izracunavanje vremena za sledeci obrok i serviranje
  checkEsp(); //Osluskivanje da li su podaci sa aplikacije stigli
}
```

```
// Funkcija za tumacenje JSON fajla koji stigne sa beka na ESP8622 i izvlacenje
vrednosti svojstva kolicina, vrednostDugmeta i selectedTime iz JSON-a
void processJsonData(const String& jsonData, int& kolicina, int& vrednostDugmeta) {
  // Dinamičko alociraje memorije za JSON dokument.
  const size_t capacity = JSON_OBJECT_SIZE(2) + 50;
  DynamicJsonDocument doc(capacity);

  // Konvertovanje u JSON
  DeserializationError error = deserializeJson(doc, jsonData);

  // Provera moguce greske
  if (error) {
    Serial.print("Greška prilikom tumačenja JSON podataka: ");
    Serial.println(error.c_str());
    return false;
  }
  // JSON podaci su uspešno tumačeni
  Serial.println("JSON podaci uspešno tumačeni.");

  // Izvlacenje svojstva
  kolicina = doc["k"].as<int>();
  vrednostDugmeta = doc["v"].as<int>();
}
// Funkcija iscitavanje sa vage
float getWeight(HX711* scale) {
  // Čitanje i konvertovanje podataka iz vagi koristeći kalibraciju
  return scale->get_units(10) / 1000.0;
//Funkcija za rotaciju motora
void rotirajMotor() {
  mojStepMotor.setSpeed(15);
  mojStepMotor.step(stepsPerRevolution);
}
void blinkBlue(int count, int duration) {
  for (int i = 0; i < count; i++) {
    digitalWrite(LEDB, HIGH);
    delay(duration);
    digitalWrite(LEDB, LOW);
    delay(duration);
  }
}
void blinkRed(int count, int duration) {
  for (int i = 0; i < count; i++) {
    digitalWrite(LEDR, HIGH);
    delay(duration);
    digitalWrite(LEDR, LOW);
    delay(duration);
  }
}
```

7.3.2 Software implementation of the web application

7.3.2.1 Server-side application

In the following images, some of the most important algorithms used in the software implementation of the server-side of the web application will be shown.

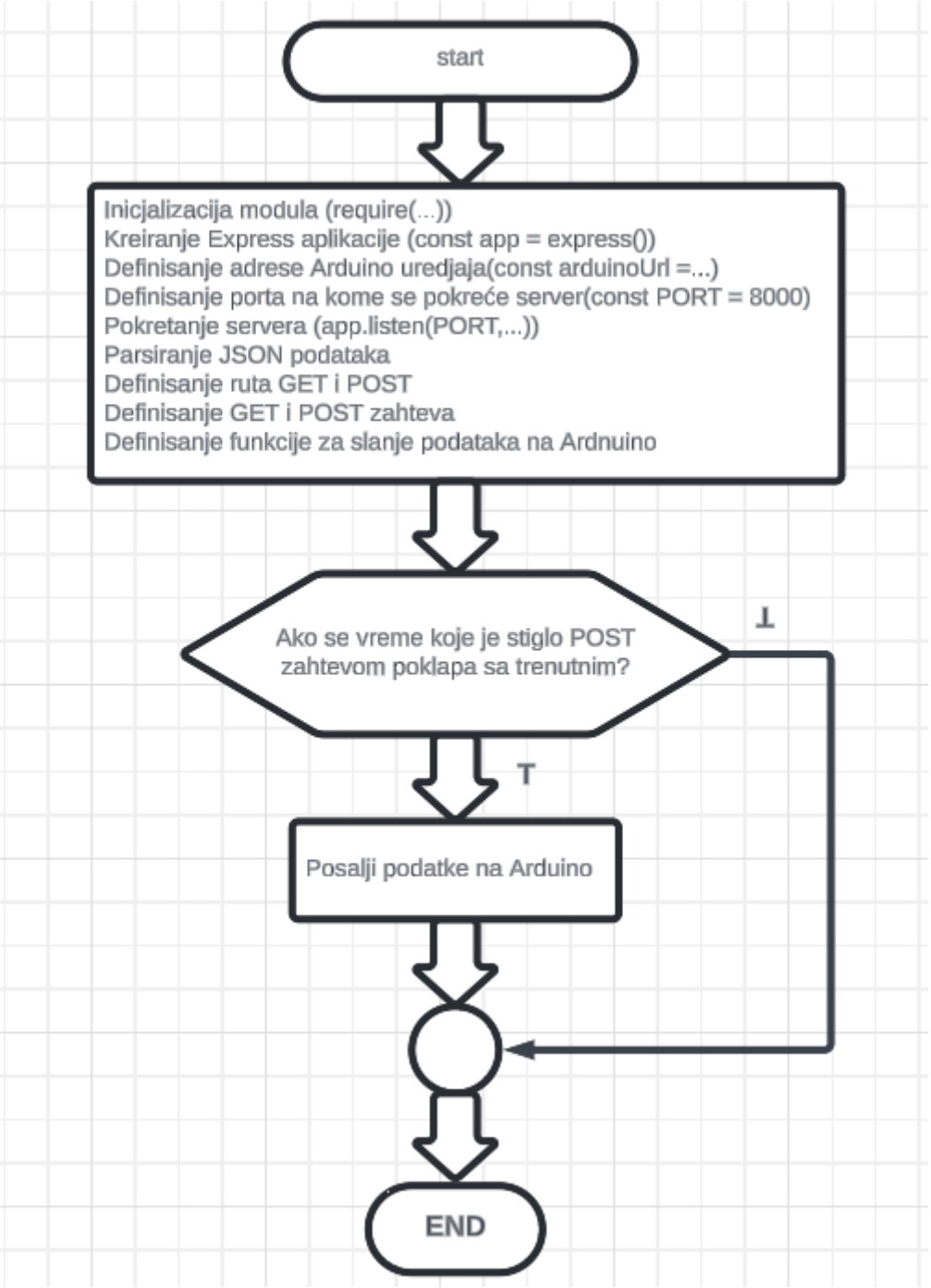


Figure 7.3.2.1.1. Algorithm outline of the main program.

The server software configuration is implemented with the following code:

```
import { useState } from 'react'
import Kolicina from './components/Kolicina'
import Dugmici from './components/Dugmici'
import Tajmer from './components/Tajmer'
import PotvrdniPopUp from './components/PotvrdniPopUp'
import './App.css'

function App() {
  const [kolicina, setKolicina] = useState(0);
  const [vrednostDugmeta, setVrednostDugmeta] = useState(null);
  const [selectedTime, setSelectedTime] = useState('00:00');
  const [isModalOpen, setIsModalOpen] = useState(false);

  const handleAmountChange = (amount) => {
    setKolicina(amount);
  }
  const handleTimeChange = (time) => {
    setSelectedTime(time);
  };

  const handleButtonClick = (value) => {
    setVrednostDugmeta(value);
  };

  const handleConfirmButtonClick = () => {
    setIsModalOpen(true);
  };

  const handleConfirmModalClose = () => {
    setIsModalOpen(false);
  };

  const handleConfirmModalConfirm = () => {
    posaljiNaBackend();
    setIsModalOpen(false);
  };

  const posaljiNaBackend = async () => {
    try {
      const response = await fetch('http://localhost:8000/api/saveData', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
```

```
          kolicina,
          vrednostDugmeta,
          selectedTime,
        }),
      });

      if (!response.ok) {
        throw new Error('Neuspešan zahtev na backend.');
      }

      // Ovde možete obraditi odgovor od servera ako je potrebno
      const data = await response.json();
      console.log('Odgovor sa servera:', data);
    } catch (error) {
      console.error('Greška prilikom slanja podataka na back:', error.message);
    }
  };


  return (
    <>
      <h1>Pametna hranilica</h1>
      <div id='container'>
        <Kolicina onAmountChange={handleAmountChange} />
        <Dugmici onButtonClick={handleButtonClick} />
        <p style={{ color: vrednostDugmeta !== null ? 'white' : 'red'}} >
          {vrednostDugmeta !== null ? `Odabrali ste: ${vrednostDugmeta}h` : 'Niste
odabrali razmak izmedju obroka!!!'}
        </p>
        <Tajmer onTimeChange={handleTimeChange} />
        <p>{selectedTime !== null ? `Odabrano vreme prvog obroka je:
${selectedTime}h` : 'Vreme prvog obroka nije odabrano!!!'}</p>
        <button onClick={handleConfirmButtonClick}>Potvrdi</button>
        <PotvrdniPopUp
          isOpen={isModalOpen}
          onRequestClose={handleConfirmModalClose}
          onConfirm={handleConfirmModalConfirm}
          kolicina={kolicina}
          ponavljanje={vrednostDugmeta}
          vreme={selectedTime}
        />
      </div>
    </>
}

export default App
```

7.3.2.2 Client-side application

The client software configuration is implemented with the following code:

```
import { useState } from 'react'
import Kolicina from './components/Kolicina'
import Dugmici from './components/Dugmici'
import Tajmer from './components/Tajmer'
import PotvrdniPopUp from './components/PotvrdniPopUp'
import './App.css'

function App() {
  const [kolicina, setKolicina] = useState(0);
  const [vrednostDugmeta, setVrednostDugmeta] = useState(null);
  const [selectedTime, setSelectedTime] = useState('00:00');
  const [isModalOpen, setIsModalOpen] = useState(false);

  const handleAmountChange = (amount) => {
    setKolicina(amount);
  }
  const handleTimeChange = (time) => {
    setSelectedTime(time);
  };

  const handleButtonClick = (value) => {
    setVrednostDugmeta(value);
  };

  const handleConfirmButtonClick = () => {
    setIsModalOpen(true);
  };

  const handleConfirmModalClose = () => {
    setIsModalOpen(false);
  };

  const handleConfirmModalConfirm = () => {
    posaljiNaBackend();
    setIsModalOpen(false);
  };

  const posaljiNaBackend = async () => {
    try {
      const response = await fetch('http://localhost:8000/api/saveData', {
        method: 'POST',
        headers: {
```

```
'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          kolicina,
          vrednostDugmeta,
          selectedTime,
        }),
      });

      if (!response.ok) {
        throw new Error('Neuspešan zahtev na backend.');
      }

      // Ovde možete obraditi odgovor od servera ako je potrebno
      const data = await response.json();
      console.log('Odgovor sa servera:', data);
    } catch (error) {
      console.error('Greška prilikom slanja podataka na back:', error.message);
    }
  };
  return (
    <>
      <h1>Pametna hranilica</h1>
      <div id='container'>
        <Kolicina onAmountChange={handleAmountChange} />
        <Dugmici onButtonClick={handleButtonClick} />
        <p style={{ color: vrednostDugmeta !== null ? 'white' : 'red'}} >
          {vrednostDugmeta !== null ? `Odabrali ste: ${vrednostDugmeta}h` : 'Niste
odabrali razmak izmedju obroka!!!'}
        </p>
        <Tajmer onTimeChange={handleTimeChange} />
        <p>{selectedTime !== null ? `Odabrano vreme prvog obroka je:
${selectedTime}h` : 'Vreme prvog obroka nije odabrano!!!'}</p>
        <button onClick={handleConfirmButtonClick}>Potvrdi</button>
        <PotvrdniPopUp
          isOpen={isModalOpen}
          onRequestClose={handleConfirmModalClose}
          onConfirm={handleConfirmModalConfirm}
          kolicina={kolicina}
          ponavljanje={vrednostDugmeta}
          vreme={selectedTime}
        />
      </div>
    </>
}
export default App
```

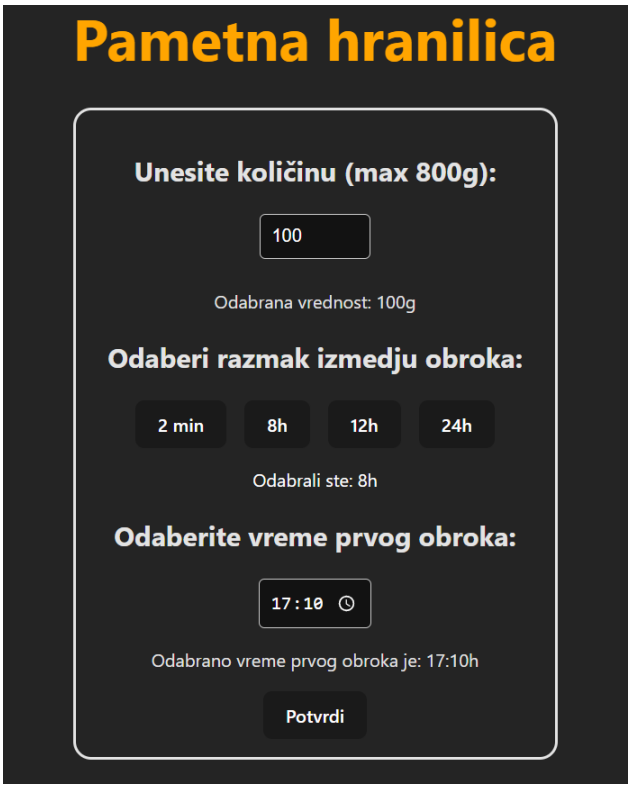


Figure 7.3.2.1. Graphical representation of the web application.

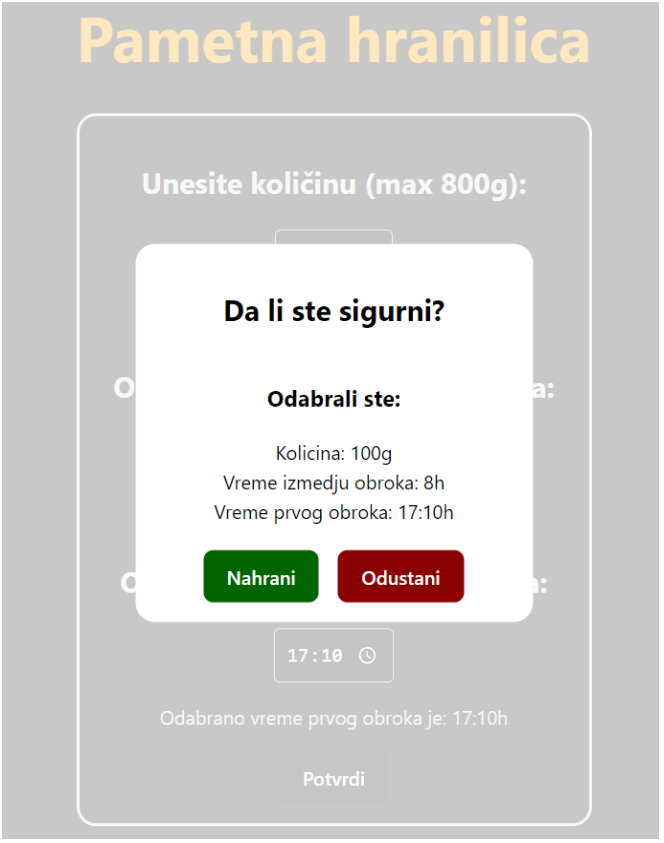


Figure 7.3.2.2. Graphic representation of the confirmation form.

8. Conclusion

This paper has detailed the process of implementing a smart pet feeder, enabling automatic feeding at specified time intervals and automatic water replenishment. The key component of this system is the Arduino UNO R3, which controls all hardware aspects of the device, including weight sensors, stepper motor, relay module, and LED indicators.

The web application, developed using React for the frontend and Node.js for the backend, allows users to intuitively manage the device, including entering food quantities, the time of the first meal, and the meal repetition interval.

This paper thoroughly analyzes the technical challenges and solutions encountered during the project implementation, emphasizing how the combination of Arduino technology and a web application can enhance daily pet care.

REFERENCES

[1] Günther Gridling, Bettina Weiss, Introduction to Microcontrollers, Viena University of Technology, february 2007, pp. 1-8

[2] Opis strukture mikrokontrolera 8051, https://www.automatika.rs/baza- znanja/mikrokontroleri/opis-strukture-mikrokontrolera-8051.html

[3] What Is A Microprocessor: Complete Guide With Examples, https://www.softwaretestinghelp.com/what-is-a-microprocessor/ , Jun 2023

[4] Difference between Microprocessor and microcontroller, https://www.sathyabama.ac.in/sites/default/files/course-material/2020-10/UNIT1 _9.pdf

[5] ATMega328P Microcontroller, https://components101.com/microcontrollers/atmega328p April 2018

[6] ATmega328P, https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810- Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[7] Uroš Pešović, “MIKROKONTROLERSKI SISTEMI”, pp 8-17

[8] AVR Memory, https://microchipdeveloper.com/8avr:memory

[9] Arduino Uno, https://www.farnell.com/datasheets/1682209.pdf

SAŽETAK / ABSTRACT

Automatska hranilica kontrolisana web aplikacijom

Ilija Milojković Rer 21/20 / dr Zoran Milivojević

Sažetak – Ovaj rad istražuje primenu Arduino UNO R3, senzora težine, step motora, relej modula i LED dioda u svrhu realizacije pametne hranilice za kućne ljubimce. Web aplikacija razvijena korišćenjem React-a i Node.js-a omogućava korisnicima intuitivno upravljanje uređajem, uključujući unos količine hrane, vreme prvog obroka i interval ponavljanja obroka. Sistem automatski proverava i dopunjuje vodu, te omogućava precizno doziranje hrane u zadatim vremenskim intervalima.

Automated Pet Feeder Controlled by a Web Application

Ilija Milojković Rer 21/20 / Dr. Zoran Milivojević

Abstract – This paper explores the application of Arduino UNO R3, weight sensors, stepper motor, relay module, and LED indicators for the realization of a smart pet feeder. A web application developed using React and Node.js allows users to intuitively control the device, including setting the amount of food, the time of the first meal, and the feeding interval. The system automatically checks and refills water and ensures precise food dispensing at scheduled intervals.

Biography

Ilija Milojković was born on April 10, 1999, in Niš, Republic of Serbia. He completed elementary and high school in Niš as well. He enrolled at the Academy of Technical Educational Studies, specializing in Modern Computer Technologies in Niš, in the 2020/21 school year and successfully passed all exams within the prescribed period. During his studies, he developed a passion for computer technologies, expanded his knowledge, and improved his skills through hard work and dedication.