Факултет инжењерских наука Универзитета у Крагујевцу

Лазар И. Спасенић

Мобилне апликације у контроли Arduino управљаних уређаја

Хранилица за кућне љубимце

завршни рад

Крагујевац, 2021.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Рачунарска техника и софтверско инжењерство

Ниво студија: Основне академске студије

Предмет: Програмирање мобилних апликација

Број индекса: 560/2015

Лазар И. Спасенић

Мобилне апликације у контроли Arduino управљаних уређаја

Хранилица за кућне љубимце

завршни рад

Комисија за преглед и одбрану:

1. **Ред. проф. др Ненад Грујовић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог завршног рада кандидат треба да

осмисли, формира и реализује хранилицу за кућне љубимце. Циљ пројекта јесте да се уз помоћ smart уређаја управља arduino деловима који чине главни део хранилице. Потребно је одрадити детаљан опис arduino делова као и развојних окружења која су коришћена у изради овог пројекта. Ради лакшег одржавања апликативног дела пројекта потребно је извршити и детаљан опис целокупног кода који је исписан. Корисничка апликација треба да буде једноставна и разумљива за свакога. Коришћење апликације и све њене функције потребно је описати.

Препоручена литература:

[1] https://bastiaanvanhengel.files.wordpress.com/2016/06/arduino_projects_book.pdf

[2] https://developer.android.com/studio

[3] .....

Крагујевац, датум
_____ ,

ментор:
др Ненад Грујовић, редовни професор

Резиме рада:

Циљ овог пројекта јесте креирање аутоматске хранилице за кућне љубимце којом ће се управљати преко smart уређаја. Комуникација између уређаја и аутоматске хранилице се врши путем интернета.
Главни део хранилице чине arduino stepper мотор и Wemos D1 mini плоча који су испрограмирани у Arduino IDE софтверу, и који комуницирају и примају команде из мобилне апликације која је развијена у Android Studio програму.

Кључне речи:

Arduino, Stepper мотор, Wemos D1 mini, Android Studio, Аутоматска хранилица

Summary of work:

The goal of this project is to create an automatic pet feeder who will be managed by smart device. Communication between the device and the automatic feeder is done with the localhost address.
The main part of the feeder consists of arduino stepper motor and Wemos D1 mini board which are programmed in Arduino IDE software, and which communicate and receive commands from the mobile application which was developed in the Android Studio program.

Key words:

Arduino, Stepper motor, Wemos D1 mini, Android Studio, Automatic feeder

Садржај

1. Увод 6
2. Компоненте коришћене за реализацију пројекта 7
2.1. Wemos d1 mini/ESP8266-12f 7
2.2. Stepper мотор 10
2.3. Драјвер A4988 за stepper мотор 13
3. Развојна окружења коришћена за реализацију пројекта 16
3.1. Развојно окружење Arduino 16
3.2. Развојно окружење Андроид 20
3.2.1. Структура Андроида 20
3.2.2. ANDROID STUDIO 20
4. Реализација 21
4.1. Arduino 21
4.2. Мобилна апликација 26
5. Коришћење апликације 48
6. Изглед хранилице 50
7. Принцип рада хранилице 55
Закључак 58
Литература 59

1. Увод

Тема овог завршног рада јесте „Хранилица за кућне љубимце“. Хранилица непрекидно пружа храну за вашег љубимца током дана, осигуравајући вам да ће остати сит и срећан када будете ишли на кратак одмор или ако останете прековремено на послу. Реализација овог пројекта вам омогућава да у сваком тренутку можете послужити изабрани број порција свом љубимцу без обзира где се налазите. Хранилица је прилагодљива и одлична за псе, мачке и мале животиње различитих величина. Употребом мобилне апликације, корисник се повезује са хранилицом преко localhost адресе, и управља бројем порција које жели да послужи. Оваквим начином храњења није потребно прилазити хранилици осим када се сипа храна у њу. Аутоматска хранилица чува храну, а на врху се налази поклопац који чини да храна остане хрскава, ароматична и укусна. Поред тога што корисник може да изабере број порција које ће се тренутно послужити кућном љубимцу, апликација садржи и тајмер који може да се подеси да у тачно одређено време послужи одговарајући број порција. Свако послуживање порција било тренутно или у одговарајуће изабрано време се бележи у базу, тако да корисник у сваком тренутку може да провери када и са колико порција је кућни љубимац нахрањен. Ти подаци се могу брисати ако је то потребно. Хранилица може испоручити 7 порција и након тога мора поново да се напуни. Аутоматска хранилица има два напајања, 12V које покреће stepper мотор и 5V које покреће Wemos D1 mini плочу и веома је једноставна за коришћење и одржавање.

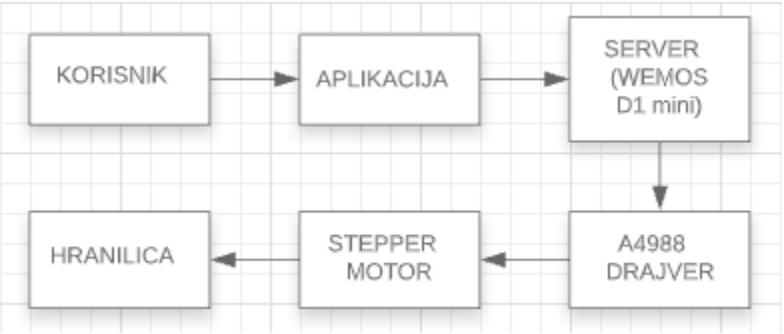
Пројекат је рађен у два програма.

- Arduino[1] – у овом делу пројекта су програмирани Wemos D1 mini плоча и stepper мотор, који се налазе директно у хранилици. Stepper мотор ротирањем вратила послужује број порција који је изабран у апликацији
- Android studio[2] – у овом делу пројекта је програмирана мобилна апликација која шаље одговарајуће информације localhost адреси, која управља stepper мотором

2. Компоненте коришћене за реализацију пројекта

Пројекат је реализован уз помоћ следећих компоненти:

- **Wemos d1 mini/ESP8266-12f** плоча служи да направи сервер на који ће се повезати телефон и управљати хранилицом. Ову плочу сам већ имао и радио са њом тако да сам се одлучио да је искористим у овом пројекту.
- **Stepper мотор** служи да контролише број порција које се полсужују љубимцу. Коришћен је stepper мотор јер је довољно јак да окрене силос у који се сипа храна.
- **Драјвер A4988** служи да контролише рад мотора. Мотор једноставно неће радити без овог драјвера. Он се програмира и задаје функцију коју мотор треба да уради.



Слика 1: Блок дијаграм узајамног деловања

2.1. Wemos d1 mini/ESP8266-12f

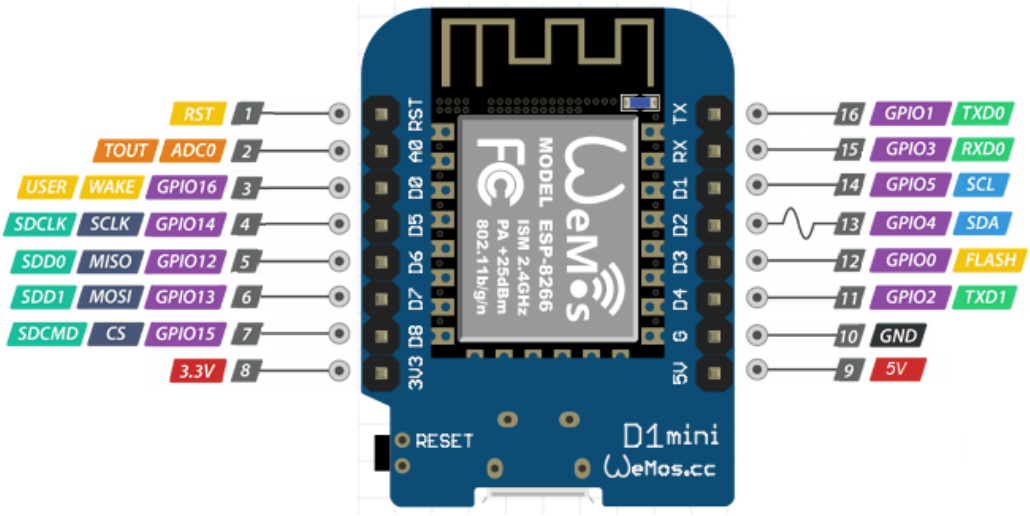
ESP8266 је веома популаран Wifi микроконтролер. Захваљујући атрактивној цени и великим могућностима, брзо је постао распрострањен и омиљен међу корисницима посебно онима који се баве кућном аутоматиком. Сам ESP8266 модул није тако једноставан за почетнике, јер је потребна конекција са Ардуином, повезивање, програмирање. Зато је направљен NodeMCU модул који знатно поједностављује рад са ESP8266. У раду је коришћен Wemos D1 mini (слика 2), веома сличан NodeMCU модулу.[3]

Највећа разлика је у величини. Wemos D1 је мањи, и има новију верзију Wi-Fi модула ESP8266-12f који је много стабилнији и има бољи домет.

На плочи је интегрисан USB-UART и за почетак програмирања је потребан само USB кабал.

Развојна плоча се састоји од ESP-12f модула који садржи чип ESP8266. У њему се налази Tensilica Xtensa® 32-битни LX106 RISC микропроцесор који ради на 80 до 160 MHz подесивој тактној фреквенцији и подржава RTOS. ESP8266 интегрише 802.11b/g/n HT40 Wi-Fi примопредајник, тако да осим што може да се повеже на Wi-Fi мрежу и комуницира са интернетом, такође може и да постави сопствену мрежу, омогућавајући тако другим уређајима да се директно повежу на то. Опсег радног напона ESP8266 од 3V до 3,6V, плоча долази са LDO регулатором напона који одржава напон стабилним на 3,3V. Може поуздано да напаја до 600мА, што би

требало да буде више него довољно када ESP8266 повуче чак 80мА током RF преноса. Напајање ESP8266 Wemos D1 mini плоче се испоручује путем уграђеног MicroB USB конектора.
ESP8266 Wemos D1 mini плоча има један тастер означен као RST који се налази у горњем левом углу и служи за ресетовање ESP8266 чипа. Плоча такође има LED индикатор који се може програмирати од стране корисника и повезан је са D0 пином плоче. Садржи 11 GPIO пинова (који се могу користити као улаз-излаз, осим пина D0) као и један аналогни улазни пин A0 за који је максимални улазни напон 3.2V.



Слика 2: Wemos D1 mini са интегрисаним ESP8266-12f

Power су пинови за напајање. Постоје два пина за напајање, један од 3.3V и један од 5V. Пин од 5V може да се користити за директно напајање ESP8266 и његових периферних уређаја. Пин 3,3V је излаз уграђеног регулатора напона. Овај пин се може користити за напајање спољних компоненти.

GND је уземљени пин развојне плоче ESP8266 Wemos D1 mini.

I2C (Inter-Integrated Circuit) пинови се користе за повезивање свих врста I2C сензора и периферних уређаја. Подржани су и I2C Master и I2C Slave. Функционалност интерфејса I2C може се програмски реализовати, а радна фреквенција је максимално 100kHz. Треба напоменути да тактна фреквенција I2C треба да буде већа од најспорије тактне фреквенције Slave уређаја.

GPIO (General-purpose input/output) пинова има 11 на ESP8266 Wemos D1 mini плочи и могу се програмски доделити разним функцијама као што су I2C, I2S, UART, PWM, IR даљински управљач, LED светло и дугме.

ADC (Analog to Digital Converter) пин Wemos D1 mini плоче је уграђен са 10-битним прецизним SAR ADC. Две функције се могу имплементирати помоћу ADC-а. Испитивање напона напајања пина VDD3P3 и испитивање улазног напона пина TOUT, али оне се не могу применити истовремено.

UART (Universal asynchronous receiver-transmitter) пинови ESP8266 Wemos D1 mini плоче имају 2 UART интерфејса, односно UART0 и UART1, који пружају асинхрону комуникацију (RS232 и RS485) и могу да комуницирају брзином до 4.5Mbps. UART0 (TXD0, RXD0, RST0 и CTS0) пинови могу се користити за комуникацију. Подржава контролу течности. Међутим, UART1 (TXD1) пин садржи само сигнал за пренос података, па се обично користи за штампање дневника.

SPI (Serial Peripheral Interface) пинови ESP8266 садрже два SPI-а (SPI и HSPI) у Slave и Master режиму. Ови SPI такође подржавају следеће SPI функције опште намене:

- 4 временска режима преноса SPI формата
- До 80MHz и подељени клокови од 80MHz
- До 64-битног FIFO-а

SDIO (Source digital input/output) пинови ESP8266 садрже сигурни дигитални улазно/излазни интерфејс који се користи за директно повезивање SD картица. Подржани су 4-битни 25MHz SDIO v1.1 и 4-битни 50MHz SDIO v2.0.

PWM (Pulse-width modulation) пинова има 4 на ESP8266 Wemos D1 mini плочи. PWM излаз се може програмски имплементирати и користити за погон дигиталних мотора и LED диода. Опсег фреквенције PWM може се подесити од 1000μs до 10000μs тј. између 100Hz и 1kHz.

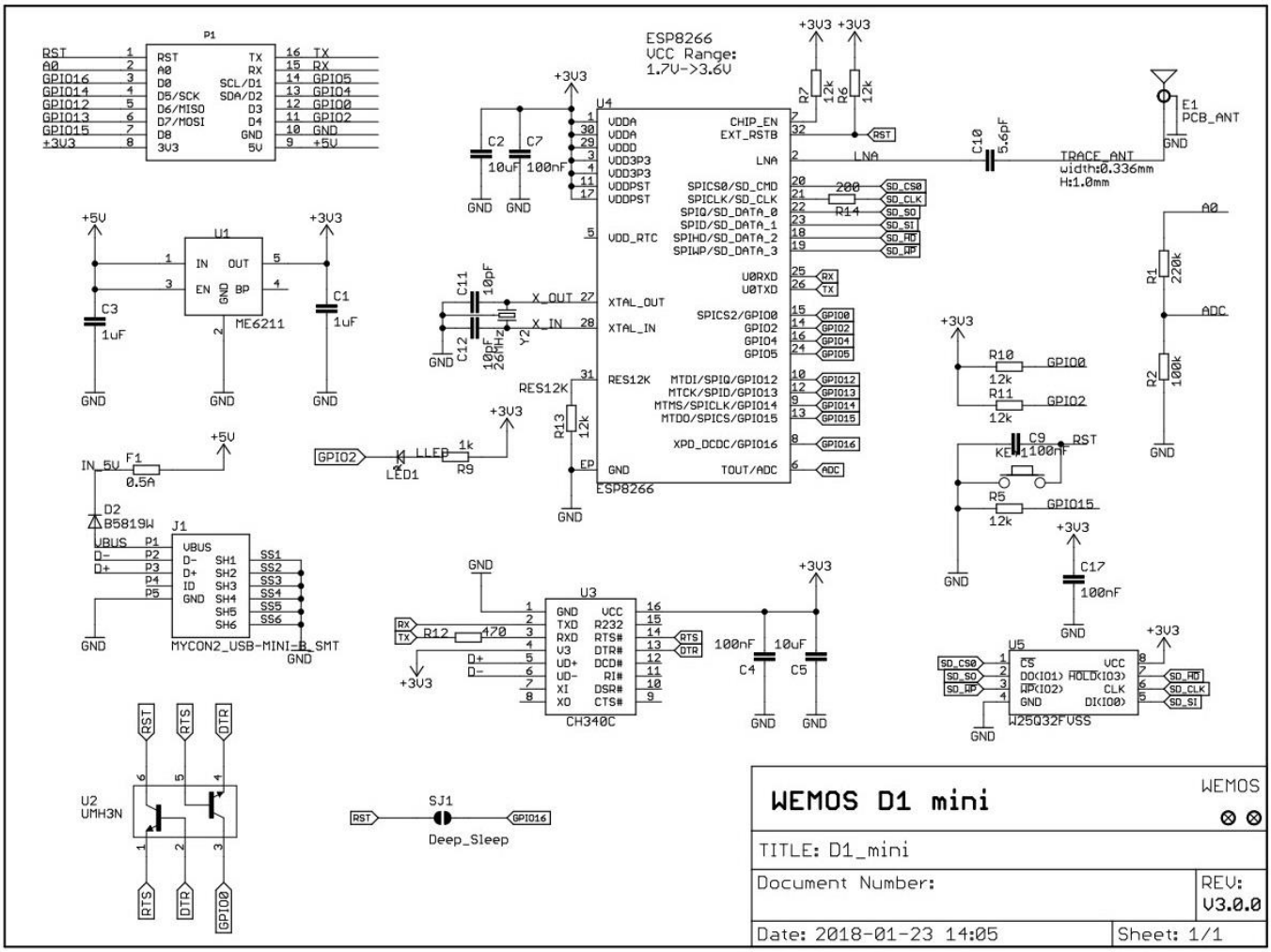
Control пинови се користе за контролу ESP8266-а. Ови пинови укључују пин за омогућавање чипа (EN), пин за ресетовање (RST) и WAKE пин.

- EN пин - ESP8266 чип је омогућен када се на EN пин позове HIGH (логичка 1). Када се на овај пин позове LOW (логичка 0), чип ради на минималној снази.
- RST пин - RST пин се користи за ресетовање чипа ESP8266.
- WAKE пин - WAKE пин се користи за буђење чипа из дубоког сна.

Спецификација:

- Омогућава рад ESP8266-12e са PCB антеном
- Меморија: 4MB
- Wi-Fi је стандард 802.11 b/g/n
- WIFI: AP (Access Point), STA (Standalone), AP+STA
- Po TKIP, WEP, CRC, CCMP, WPA/WPA2, WPS
- Напајање: 3.3V (или 5V преко USB-а)
- CPU: RISC 80MHz (подржава до 160MHz)
- 9 GPIO - PWM / I2C / SPI / 1-повезивање
- Максимални напон на У/И пиновима: 12mA
- Препоручени напон на У/И пиновима: 6mA
- USB-UART претварач - CH340
- ADC - 10-bit
- 16 пинова 2,54mm -

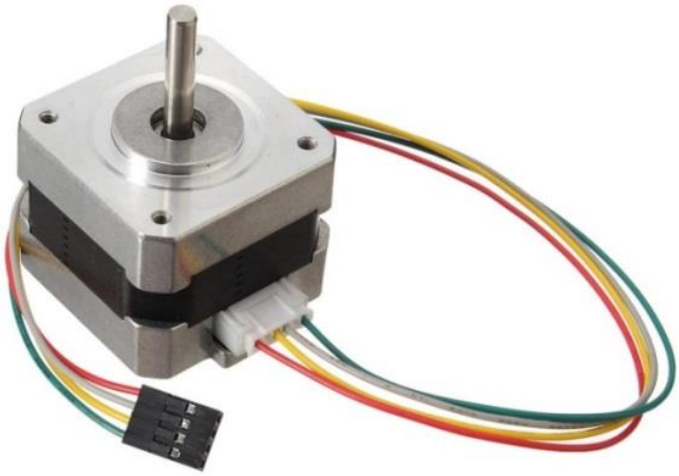
- micro USB B
- Димензије: 34 x 25mm
- LED конектован на GPIO2 (D4)



Слика 3: Електрична шема Wemos D1 mini плоче

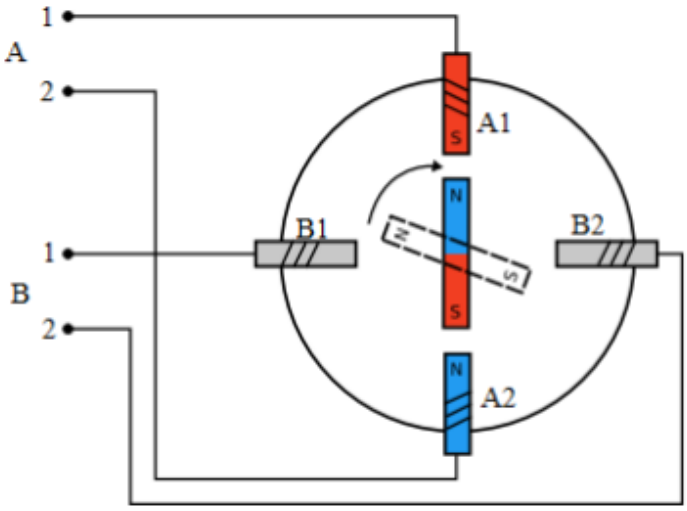
2.2. Stepper мотор

Stepper мотор је синхрони електрични мотор који прима и претвара дигиталне импулсе у механичко окретање вратила. Њиме се управља из Arduino окружења. Свака ротација stepper мотора је подељена на одвојен број корака, у многим случајевима на 200 корака, и мотор мора послати засебан импулс за сваки корак. Stepper мотор може да узме само један корак у сваком тренутку и сваки корак је исте величине. Будући да сваки импулс узрокује да мотор ротира тачан угао, обично 1,8°, положај вратила мотора се може контролисати без икаквог механизма повратне спреге. Како се дигитални импулси повећавају у фреквенцији, кретање корака се мења у континуирану ротацију, при чему је брзина ротације директно пропорционална фреквенцији импулса. Угао ротације вратила мотора је пропорционалан улазном дигиталном импулсу (слика 4).



Слика 4: Stepper мотор

Stepper мотор има прецизно позиционирање вратила, и грешку од 3% до 5% приликом ротирања. Веома је поуздан, јер у мотору нема контактних четкица, па животни век stepper мотора само зависи од издржљивости лежаја у њему.
Stepper мотори се свакодневно користе и у индустријској и у комерцијалној примени због своје ниске цене, високе поузданости, великог обртног момента при малим брзинама и једноставне, робусне конструкције која ради у скоро сваком окружењу.
Принцип рада степ мотора се заснива на чињеници да се различити магнетни полови привлаче. На слици 5 је приказан ротор у облику плочице који је намагнетисан и на коме су означени магнетни полови и четири статорска намотаја који су повезани у парове по 2 (А1-А2 и Б1-Б2).



Слика 5: Принцип рада Stepper мотора

Претпоставимо да се у почетном тренутку ротор налазио на позицији означеној испрекиданом линијом, тј да је његов северни пол био окренут намотају Б1 док је јужни био окренут ка намотају Б2. Оног тренутка када струја крене да тече кроз проводник А у смеру од 1 ка 2, због различитости смера намотаја А1 и А2,

стварају се различити магнетни полови означени као на слици. Тада северни магнетни пол ротора тежи да се приближи јужном магнетном полу статора, а јужни магнетни пол ротора тежи да се приближи северном магнетном полу статора. На овај начин се ротор заокреће у десно за одређен угао који је одређен бројем полова ротора и бројем парова намотаја статора. У случају да је потребо окренути ротор у супротном смеру, потребна је струја која се креће од краја 2 ка крају 1 намотаја А. На овај начин би намотаји А1 и А2 имали супротне магнетне полове него што је приказано на слици и ротор би се из почетног положаја заокренуо на лево за исти угао. Даље кретање се наставља тако што се сукцесивно укључују намотаји А и Б.

Stepper мотори се по типу деле на:

- Stepper моторе са перманентним (сталним) магнетом
- Stepper моторе са променљивом релуктансом
- Хибридне stepper моторе

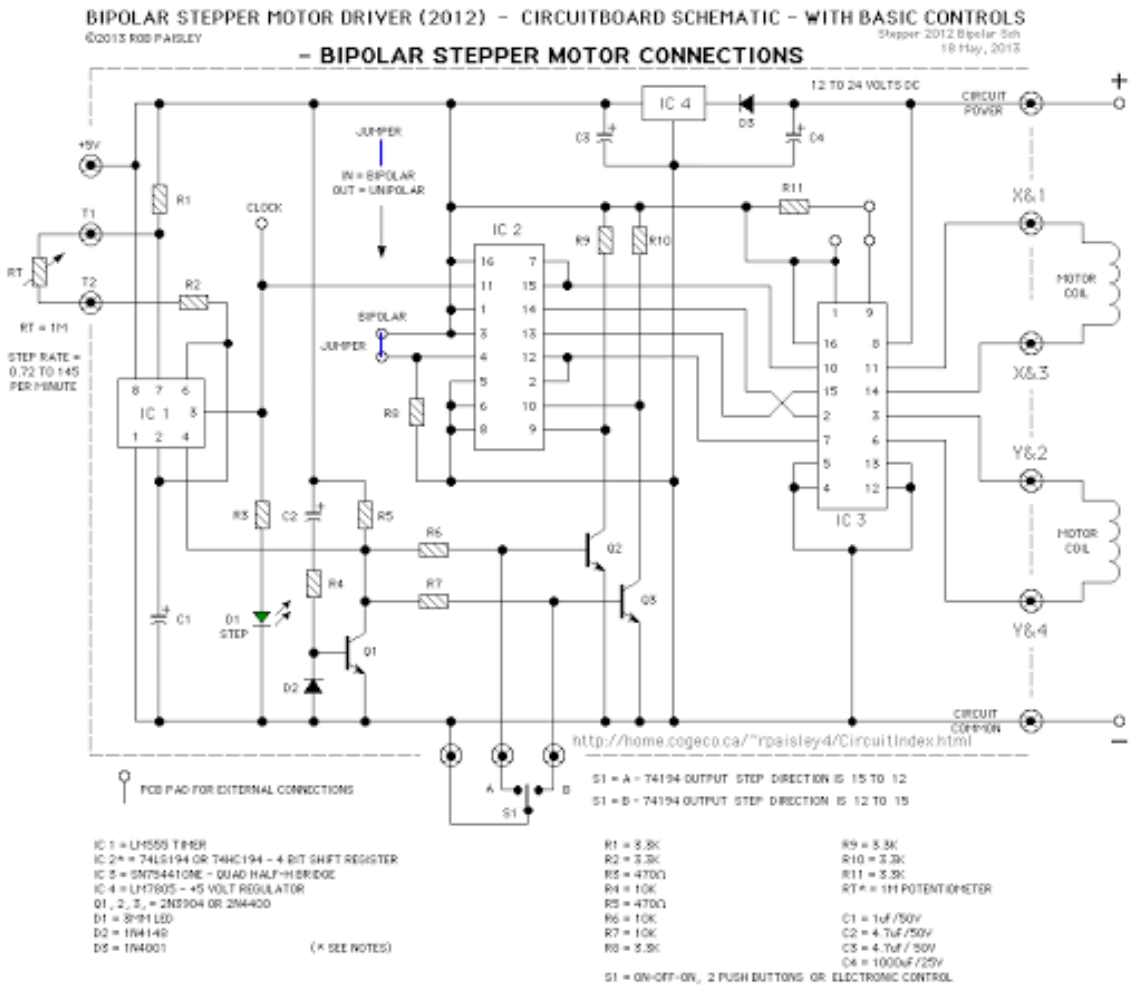
Према раду статорског намотаја разликујемо две врсте stepper мотора:

- униполарни stepper мотори (струја у фазном намотају може тећи само у једном смеру)
- биполарни stepper 6мотори (струја у фазном намотају може тећи у оба смера)

Stepper мотор који се користи у овом пројекту је биполарни хибридни.
Хибридни stepper мотори се заснивају на комбинацији начела на којима се заснива рад stepper мотора са перманентним магнетом и променљивом релуктансом. Они комбинују најбоље особине и једних и других stepper мотора. Ротор оваквих мотора је од сталног магнета, али је назубљен, или је у неким случајевима цилиндричан али пресвучен назубљеним феромагнетским материјалом. Такође, статор је назубљен и на њему се налазе равномерно распоређени намотаји. На овај начин се комбинују добра својства промењљиве релуктансе и перманентног магнетног поља.[4]

Спецификација:

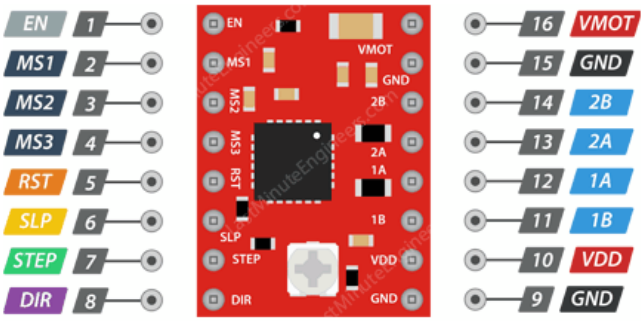
- Толеранција корака је ±5%
- Толеранција отпора је ±10%
- Толеранција индуктивности је ±20%
- Пораст радне температуре је до 80°C max
- Температура простора у коме ради: -20°C~ +50°C
- Температура простора у коме се складишти: -20°C~ +50°C
- Напајање: 12V
- Отпор: 100M Ω MIN. 500V DC
- Диелектрична чврстоћа: 500V AC 1мин
- Radial Play 0.02mm max. (450g Load)
- End Play 0.08mm max. (450g Load)
- Максимална радијална сила је 75N
- Максимална осовинска сила је 15N



Слика 6: Електрична шема stepper мотора

2.3. Драјвер A4988 за stepper мотор

A4988 је драјвер за контролисање stepper мотора са уграђеним програмом за лако руковање. Ово смањује број контролних пинова на само 2, један за контролу корака и други за контролу смера окретања. Драјвер има максимални излазни капацитет од 35V и ± 2A. Може да управља биполарним и униполарним stepper моторима (слика 7).



Слика 7: Пинови драјвера A4988 за stepper мотор

Драјвер A4988 има једноставан интерфејс за контролу ротирања вратила и смера ротирања. A4988 има укупно 16 пинова. Има подесиву контролу струје која омогућава подешавање максималне излазне струје помоћу потенциометра. То допушта да се користе напони изнад номиналног напона stepper мотора, и тиме се постиже већа брзина ротације вратила. Чип управљачког програма A4988 има неколико уграђених сигурносних функција попут прекомерне струје, кратког споја, блокаде под напоном и заштите од превисоке температуре.[5]

Драјвер A4988 захтева два прикључка за напајање:
VDD и GND служи за погон интерног логичког кола које може бити од 3V до 5,5V.
VMOT и GND служи за напајање мотора које може бити од 8V до 35V.

Драјвер A4988 има улазе за одабир величине корака (резолуције) MS1, MS2 и MS3. Постављањем одговарајућих логичких нивоа на ове пинове можемо подесити мотор на једну од пет резолуција (слика 8).

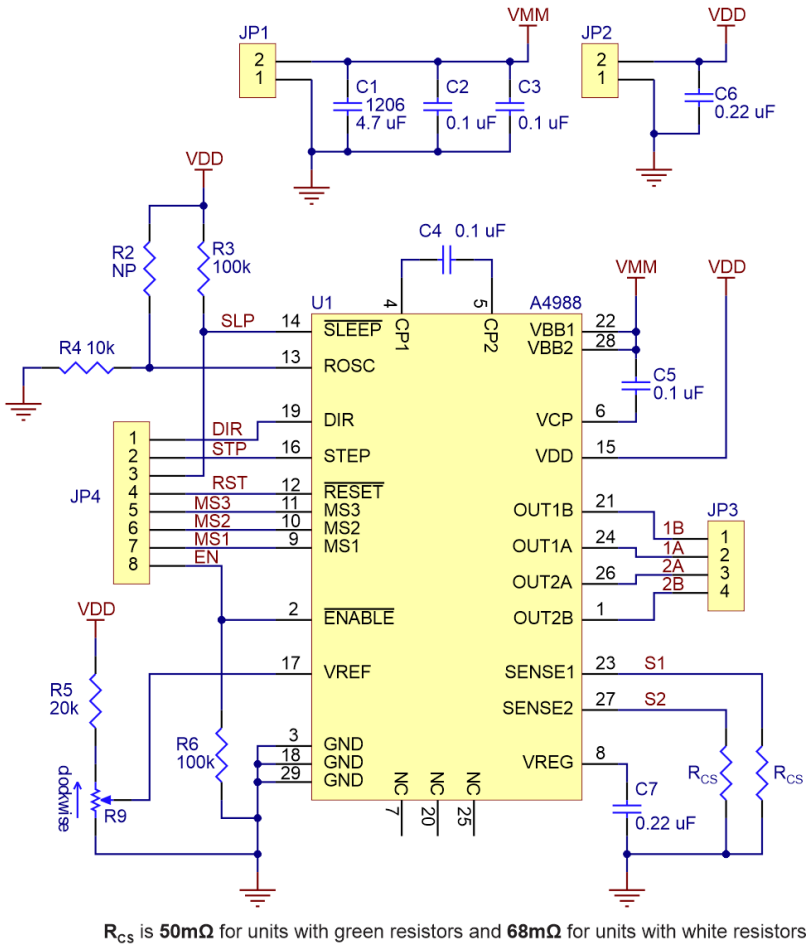
MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Слика 8: Резолуција драјвера A4988

Драјвер A4988 има два управљачка улаза:
STEP улаз контролише микрокораке мотора. Сваки високи импулс који се пошаље на овај пин ротира вратило мотора за одређен угао. Што су импулси бржи, мотор ће се брже окретати.
DIR улаз контролише смер окретања мотора. Када се позове HIGH (логичка 1) покреће се мотор у смеру кретања казаљке на сату, а када се позове LOW (логичка 0) покреће се мотор у смеру супротном од казаљке на сату.

Драјвер A4988 има три различита улаза за контролу својих стања снаге:
EN пин је активан низак улаз. Када се на овај пин позове LOW (логичка 0) управљачки програм A4988 је омогућен. На овом пину се подразумевано позива LOW (логичка 0), тако да је управљачки програм увек омогућен, осим када се позива HIGH (логичка 1).
SLP пин је активан низак улаз. Када се на овај пин позове LOW (логичка 0) поставља драјвер у режим спавања, минимизирајући потрошњу енергије. Ово се може употребљавати нарочито када се мотор не користи за уштеду енергије.

RST пин је активан низак улаз. Када се на овај пин позове LOW (логичка 0), сви STEP улази се занемарују, све док се не позове HIGH (логичка 1). Такође ресетује управљачки програм постављањем интерног преводиоца у унапред дефинисано почетно стање.
Драјвер A4988 има четри излаза 1B, 1A, 2A и 2B. На ове пинове се може повезати било који биполарни stepper мотор који има напон између 8V и 35V. Сваки излазни пин на драјверу може испоручити до 2A мотору.

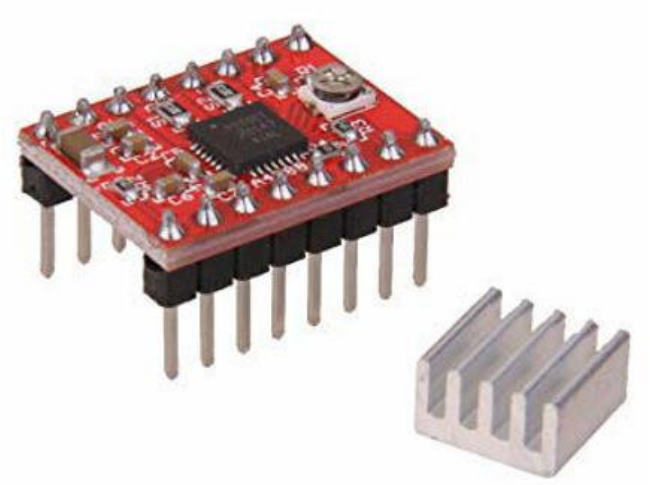


Слика 9: Електрична шема драјвера A4988

Спецификација:

- Минимални радни напон: 8V
- Максимални радни напон: 35V
- Непрекидна струја по фази: 1A
- Максимална струја по фази: 2A
- Минимални логички напон: 3V
- Максимални логички напон: 5,5V
- Микро кораци: 1, 1/2, 1/4, 1/8, 1/16
- Величина: 1,524cm × 2,032cm
- Тежина: 1,3g

Прекомерно расипање снаге драјвера A4988 резултира порастом температуре која може да превазиђе капацитет драјвера, и тако га оштети. Чак и ако драјвер A4988 има максималан напон од 2A по калему, чип може да испоручи само приближно 1A по калему а да се не прегреје. Ако се постиже више од 1A по калему потребно је наместити хладњак (слика 10).



Слика 10: Драјвер A4988 за stepper мотор

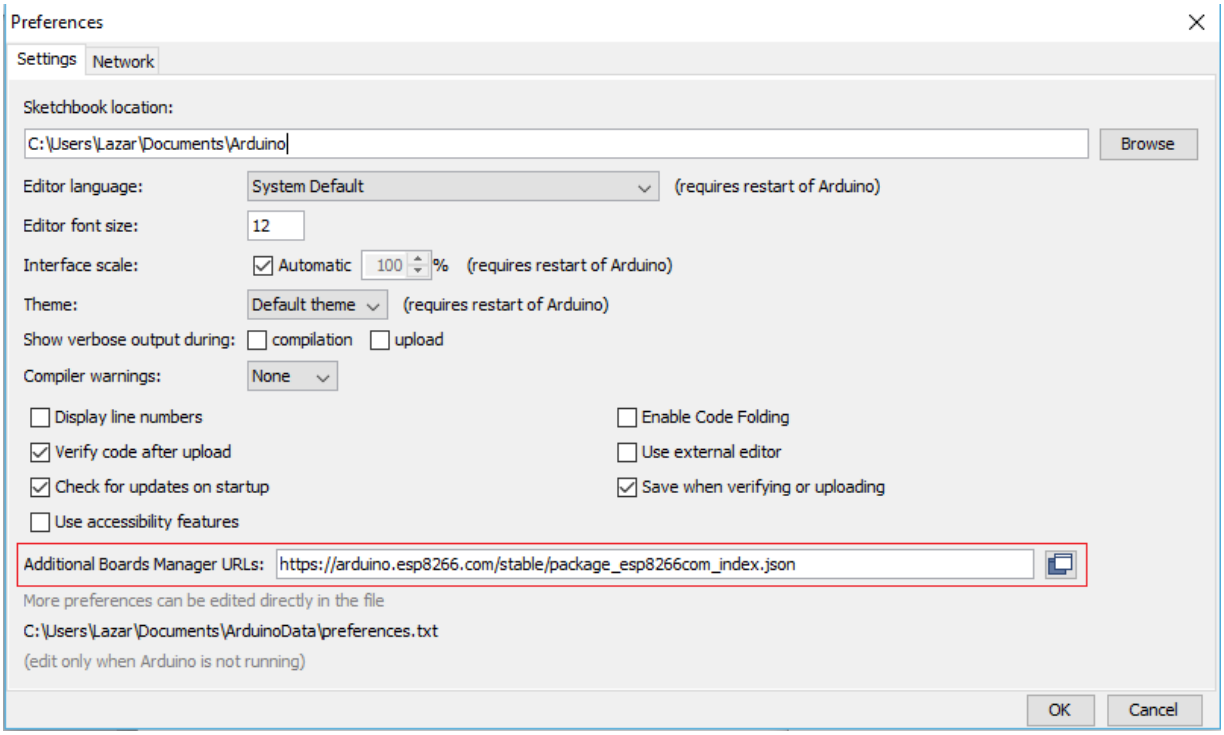
Пре употребе мотора, потребно је извршити мало подешавање. Мора се ограничити максимална количина струје која тече кроз калеме и спречити је да премаши номиналну струју мотора. На драјверу A4988 налази се мали потенциометар тримера који се може користити за подешавање тренутног ограничења. Требало би подесити ограничење струје на или испод тренутне вредности мотора.

3. Развојна окружења коришћена за реализацију пројекта

3.1. Развојно окружење Arduino

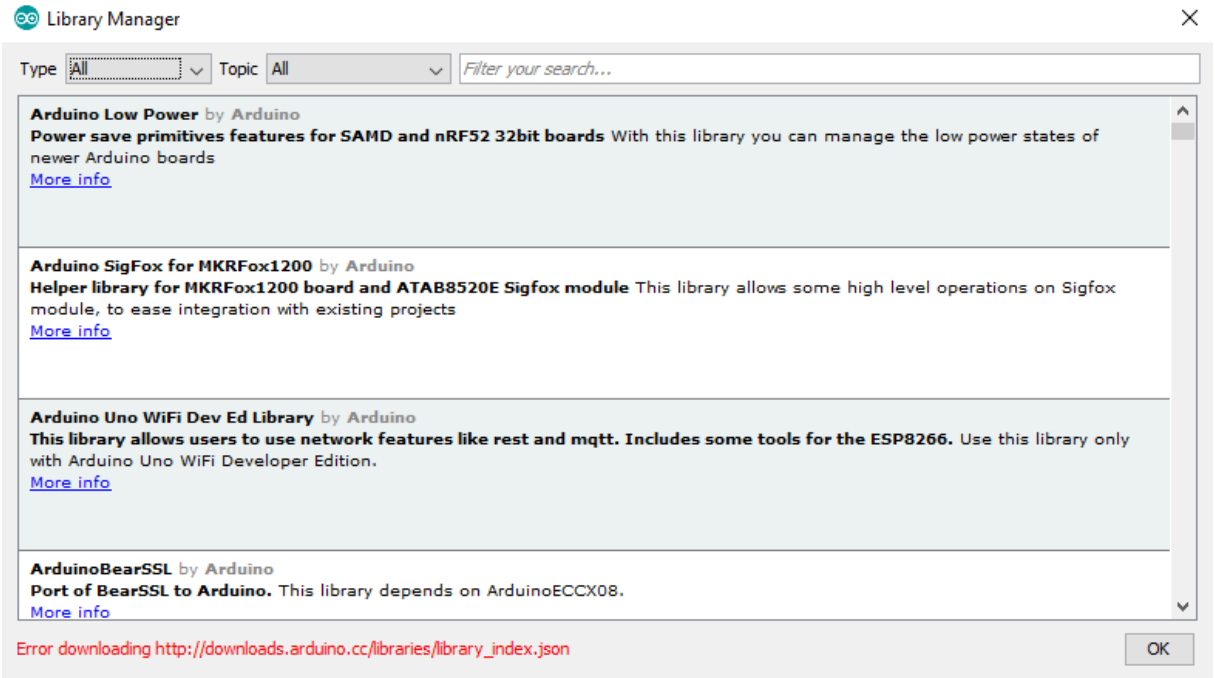
Open source Arduino software (IDE) олакшава писање и отпремање кода на плочу. Ради на свим платформама - Windows, Linux и Mac оперативним системима. Окружење је написано у JAVA програмском језику. Овај софтвер се може користити на свим Arduino плочама. Преко ARDUINO софтвера, код се може писати у било ком програмском језику. Такође, постоје и многобројне бесплатне библиотеке које се инсталирају како би писање софтвера било што једноставније.
Arduino софтвер се инсталира тако што се покрене инсталација за одговарајући оперативни систем која се налази на оригиналном сајту Arduina у одељку SOFTWARE. Након тога се врши оптимизација окружења у зависности од потребе пројекта који се преко овог окружења реализује.[1]
Да би се омогућило правилно функционисање програма, потребно је прилагодити одређене компоненте софтвера са потребама пројекта који се извршава. На

почетку рада треба креирати нов пројекат преко опције **File-New** и снимање фајла преко опције **File-Save as**.
Затим се преко опција **File-Preferences-Settings** долази до прозора где се у поље **Aditional Boards Managers URL** (слика 11) где се убацује одговарајући линк: http://arduino.esp8266.com/stable/package_esp8266com_index.json



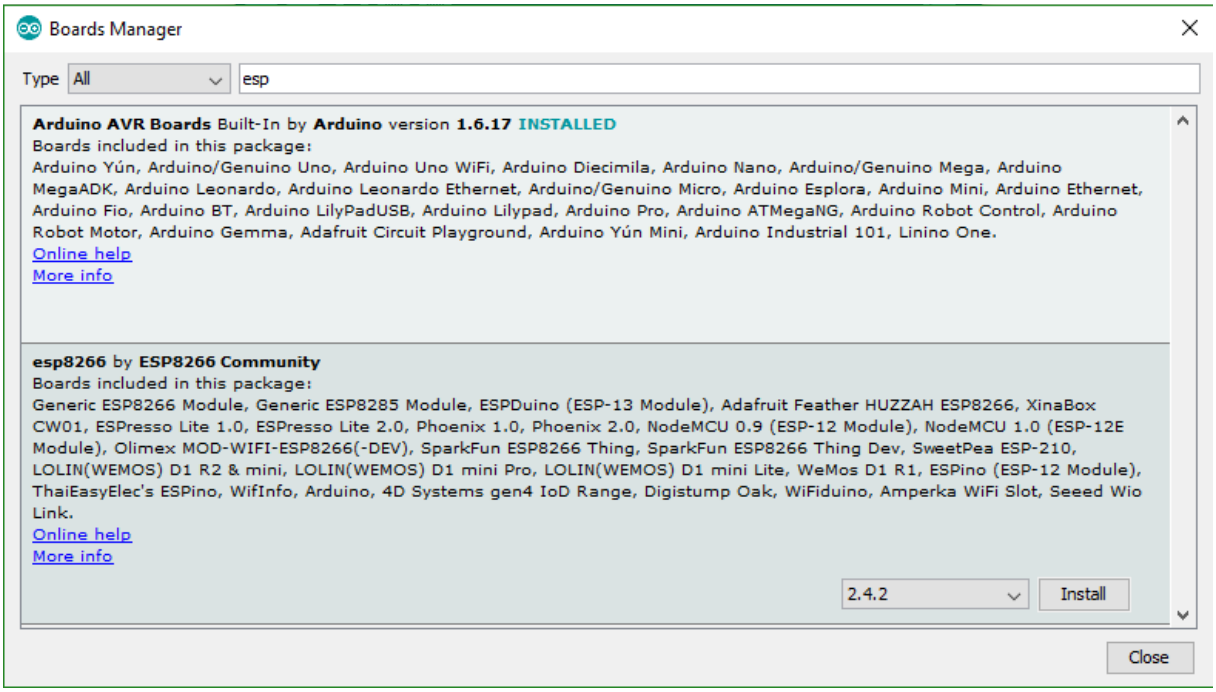
Слика 11: Прозор за додавање URL линка

Након тога се врши повезивање одговарајуће платформе. Физичко повезивање преко USB кабла и програмско преко опција: **Tools-Board** где се из листе платформи изабере одговарајућа. Затим, потребно је да се изабере одговарајући порт. (Провера се врши у **Device Manager-u**), који се налази у опцијама **Tools-Port.**
У зависности од платформе која се користи, некада је потребно инсталирати додатне драјвере, али то није чест случај. Углавном се инсталација истих врши приликом физичког повезивања платформе и рачунара.
Такође, у зависности од потреба пројекта, односно од сензора који се користе, потребно је инсталирати одређене библиотеке које су бесплатне и инсталирају се тако што се преко опција **Sketch-Include library-Manage libraries** дође до прозора (слика 12) и преко претраживача пронађе одговарајућа, а преко дугмета **Install** инсталира. Након тога потребно је ту исту библиотеку укључити у пројекат тако што се преко опција **Sketch-Include Library** из листе падајућег менија изабере одговарајућа библиотека.

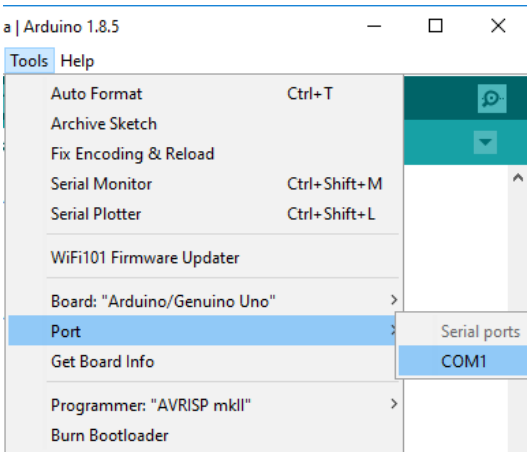


Слика 12: Прозор за инсталирање библиотека

Обзиром да користимо модул ESP8266-01 потребно је преко опције **Tools**-**Board**-**Boards Menager** пре свега инсталирати драјвере за њега, тако што се у прозору **Boards Menager**-а (слика 13), у простору за search укуца његов назив ESP8266. У случају да се користе други модели Wifi модула ESP8266, није потребно додатно инсталирање јер ови драјвери покривају све постојеће моделе.



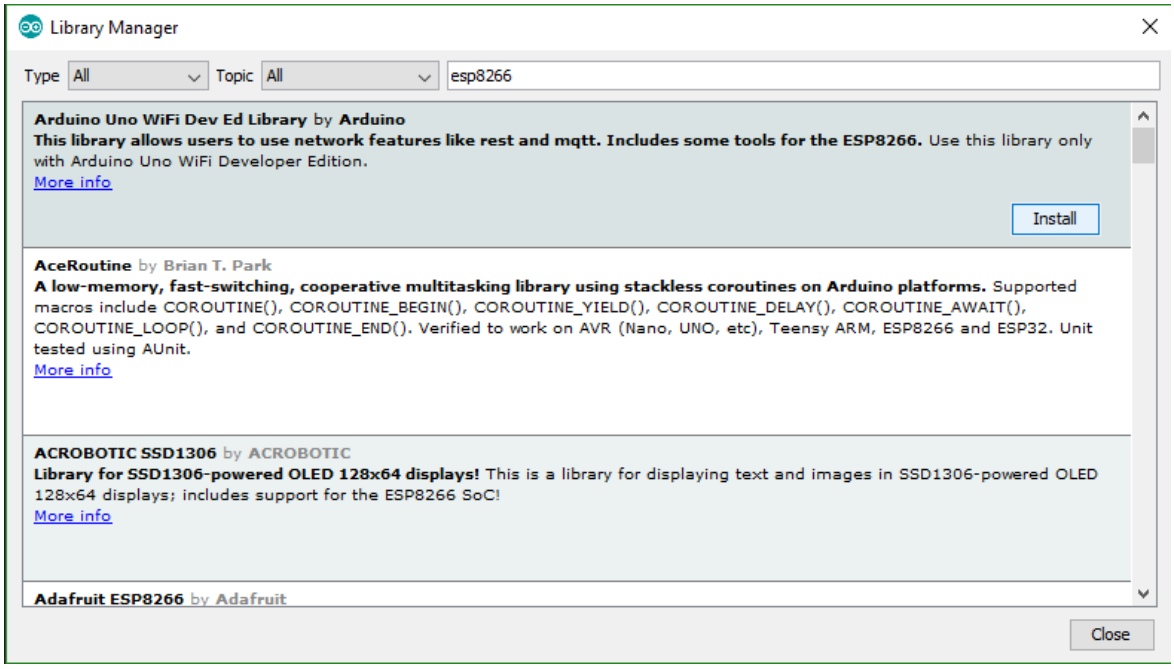
Слика 13: Инсталирање платформе ESP8266



Слика 14: Подешавање порта

Кораци:
1. Покретање новог пројекта
2. Подешавање порта, платформе, модула (слика 13)(слика 14)
3. Инсталација и укључивање библиотеке за модул ESP8266 (слика 12)
4. Имплементација кода

Инсталација библиотеке се врши преко опција **Sketch-Include library-Manage librarie** (слика 15), преко претраживача пронађе библиотека за ESP8266. Након тога потребно је ту исту библиотеку укључити у пројекат тако што се преко опција **Sketch-Include library** из листе падајућег менија изабере инсталирана библиотека модула ESP8266.[1]



Слика 15: Инсталирање библиотека

3.2. Развојно окружење Андроид

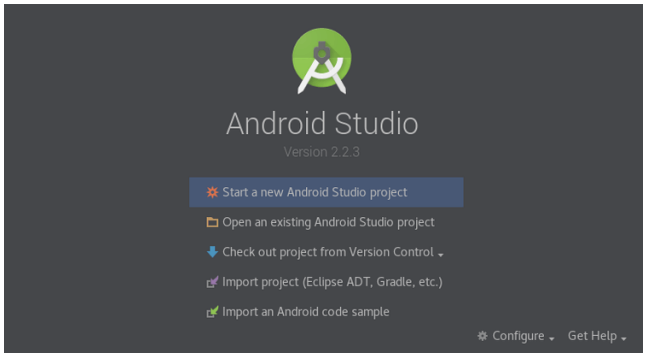
Андроид оперативни систем је свеобухватна платформа отвореног кода, осмишљена за израду апликација за мобилне уређаје. Андроид врло брзо постаје најпопуларнији мобилни оперативни систем на планети, и он нуди програмерима непревзиђене начине да уновче своје мобилне апликације на више тржишта. 2013. године, Андроид оперативни систем заузимао је више од 79% целокупне продаје паметних телефона широм САД-а, Европе и Азије. Свакодневно се активира преко 1.5 милиона Андроид уређаја.
Андроид представља комплетан развојни оперативни систем. Изворни код Андроид-а програмерима је доступан и једноставан је за разумевање.[6]

3.2.1. Структура Андроида

Андроид је заснован на Linux Kernel-u (језгру) 2.6 и Linux Kernel-u 3.x, библиотекама и АПИ (Апликационо програмски интерфејс) написани у програмском језику C/C++. С обзиром на отвореност изворног програмског кода, апликације имају могућност комуницирања и покретања других апликација. Иако је код писан у C/C++ већина ствари писана је у Java програмском језику користећи Android Software Development Kit (SDK). Андроид је и направио свој језик за програмирање апликација, који је извршавао апликације и до десет пута брже и боље располагање ресурса, али због сложене грађе прављења апликација се користи SDK. Архитектура Андроида се може представити кроз шест компоненти, где сваки компонент чини своју функцију што представља целину Андроида.[6]

3.2.2. ANDROID STUDIO

Android Studio је званично интегрисано развојно окружење (IDE) за развој Андроид апликација, базираног на IntelliJ IDEA. На врху IntelliJ-овог уређивача кода и алата за развијање, Android Studio нуди још више функција које побољшавају изградњу Андроид апликација, као што су флексибилност у систему изградње заснован на GRADLE-у, бржи и јачи емулатор, јединствено окружење у којем се може програмирати за све Андроид уређаје. Претходно коришћено радно окружење јесте Eclipse.[7]
Почетни прозор Андроид апликације приказан је на слици на слици 16.

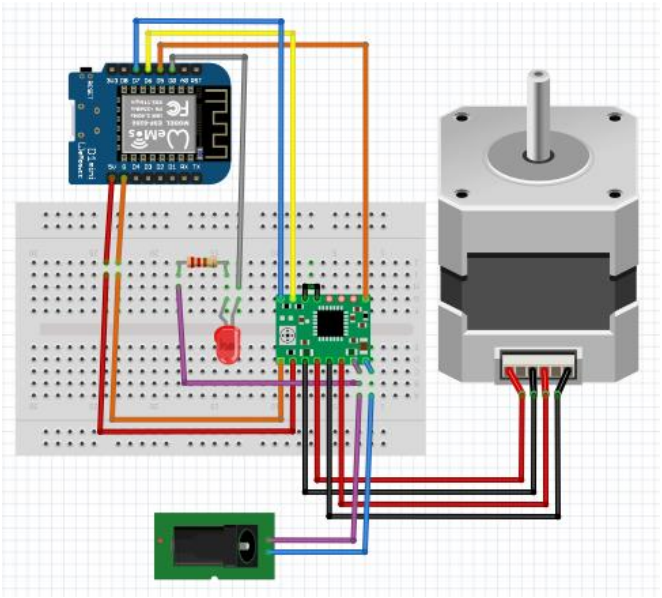


Слика 16: Android studio почетни екран

4. Реализација

4.1. Arduino

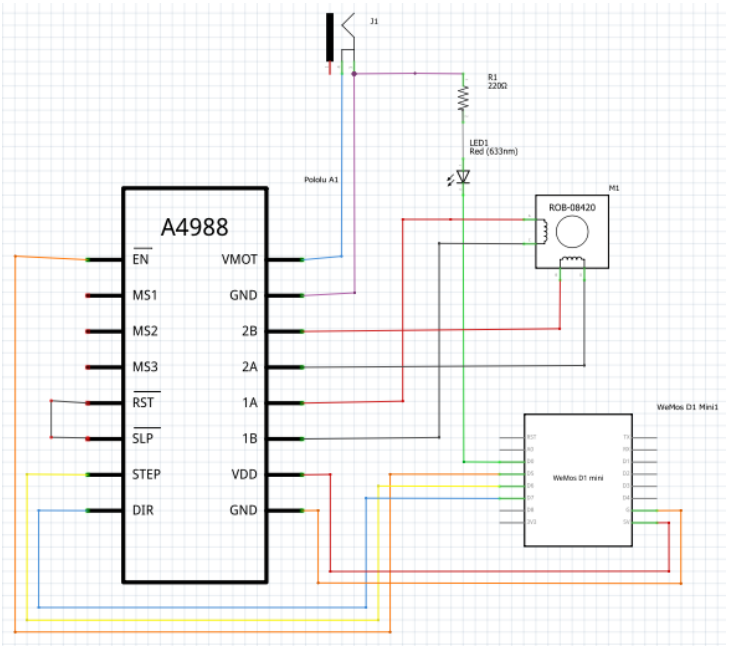
Прво је потребно повезати све компоненте на прави начин. Stepper мотор се повезује са драјвером A4988 који њиме управља. Драјвер A4988 се повезује са Wemos d1 mini/ESP8266-12f плочом, јер је она конектована на интернет и она прима потребне податке које ће да преноси драјверу. Такође повезана је и лед диода која симулира укључивање/искључивање мотора (слика 17). Wemos d1 mini има сопствени улаз за напон од 5V, док за покретање мотора потребно је довести напон од 12V до A4988 драјвера.



Слика 17: Повезивање Arduino компоненти

VMOT – 8-35V
GND – motor ground
2B, 2A, 1A, 1B – stepper motor
VDD – 5V
GND – logic ground
DIR – D7
STP – D6
SLP – RESET
ENABLE – D5

Електрична шема повезаних Arduino компоненти служи да би се тачно и прецизно повезале све компоненте (слика 18).



Слика 18: Електрична шема повезаних Arduino компоненти

После физичког повезивања компоненти потребно их је испрограмирати у Arduino развојном окружењу.

Прво је потребно да се укључе одређене библиотеке. Након тога се дефинишу подаци којима Wemos d1 mini повезује на интернет, ssid и password информације, и задају се почетне вредности параметрима који се користе за добијање сигнала са localhost адресе. Онда се дефинишу параметри који су потребни за програмирање драјвера A4988 и то су stepPin, dirPin, enPin који представљају драјвер пинове који су повезани на Wemos d1 mini плочу. currentAngle, angle, stepPerAngle и numstep су вредноси потребних углова за програмирање stepper мотора (слика 19).

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Stepper.h>
```

библиотеке које се користе у Arduino развојном окружењу

```
//Wi-Fi adresa i sifra
char* ssid = "BIA_kombi_za_nadzor";
char* password = "kompozitori";
String red_light = "0";
String one_portion = "0";
String start_position = "0";
```

дефинисање Wifi информација којима се Wemos d1 mini повезује на интернет, и задају се почетне вредности параметрима који се користе за рад

```
//Definisanje pinova
const int stepPin = D6;
const int dirPin = D7;
const int enPin = D5;
```

пинови на A4988 драјверу који се повезују са пиновима на Wemos d1 mini плочи

```
int currentAngle = 0;
int angle = 0;
float stepPerAngle = 1.8;
int numstep;
```

вредности углова који се користе у програмирању stepper мотора

```
ESP8266WebServer server;
```

Слика 19: Дефинисање потребних података

У другом делу кода или setup() функцији се одређују output пинови који су повезани на Wemos d1 mini плочу и напони на којима раде (LOW је 0V, HIGH је 3.3V или 12V), и у њој се Wemos d1 mini повезује на интернет (слика 20). Позива се функција serverSection() која конфигурише сервер.

```
void setup(){
  //Making Connection with network
  WiFi.begin(ssid, password);

  Serial.begin(115200);

  pinMode(D0, OUTPUT);
  digitalWrite(D0, LOW);

  pinMode(stepPin, OUTPUT);
  pinMode(dirPin, OUTPUT);
  pinMode(enPin, OUTPUT);

  digitalWrite(enPin, LOW);
  digitalWrite(dirPin, HIGH);

  Serial.print("Searching Connection");
  while (WiFi.status() != WL_CONNECTED){
    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.print("IP Address: ");
  Serial.print(WiFi.localIP());

  serverSection();
}
```

конфигурисање пинова да представљају излазе, и напоне који су им потребни

повезивање Wemos d1 mini плоче на интернет

Слика 20: setup() функција

Трећи део кода или loop() функција служи да дефинише сервер (слика 21).

```
void loop(){
  server.handleClient();
}
```

Слика 21: Дефинисање сервера

Четврти део кода или serverSection() функција служи да конфигурише потребне податке на сервер. На сервер се постављају подаци којима се пали лед диода (“/red_light”), покреће мотор (“/one_portion”) и бележи се тренутни стаус података (“/status”)(0 или 1, у зависности од тренутног стања мотора и диоде) слика 22).

```
void serverSection(){
  server.on("/", [](){
    server.send(200, "text/html", "<!DOCTYPE html><html><meta charset='UTF-8'><head></head>
                                   <body><h3><a href='/red_light'>Light</a></h3>
                                   <br><h3><a href='/one_portion'>One portion</a></h3></body></html>");
    Serial.print(" Poslato na server");
  });

  server.on("/red_light", red_light_state);
  server.on("/one_portion", one_portion_state);

  server.on("/status", all_state);

  server.begin();
}
```

Слика 22: Имплементација података на сервер

server.on() – позива одговарајућу функцију када клијент затражи URI
server.send() - када се затражи URI шаље веб страницу са дугмићима “Light” и “One portion”
server.begin() – стартује сервер

Пети део кода или red_light_state() функција служи да дефинише вредности за укључивање и искључивање лед диоде. Када је на серверу вредност 0 значи да је напон који се доводи до диоде једнак 0V, и диода је искључена, а Arduino исписује текст на серверу “Hranilica je isključena“, а када је на серверу вредност 1 значи да је напон који се доводи до диоде једнак 3,3V, и диода је укључена, а Arduino исписује текст на серверу “Hranilica je uključena“ (слика 23).

```
//Start i stop motor
void red_light_state(){
  if(red_light == "0"){
    red_light = "1";
    digitalWrite(D0, HIGH);
    server.send(200, "text/html", "Hranilica je uključena");
    Serial.print(" Light: ");
    Serial.print(red_light);
  }else{
    red_light = "0";
    digitalWrite(D0, LOW);
    server.send(200, "text/html", "Hranilica je isključena");
    Serial.println(" Light: " + red_light);
    //Serial.print(red_light);
  }
}
```

Слика 23: Дефинисање укључивања и искључивања мотора

Шести део или Control_motor() функција служи да дефинише кретање мотора (померање палице за одговарајући угао). Мотор сам види који је тренутни угао палице (currentAngle), и потребно му је задати одговарајућу угао (angle) да би он ротирао палицу за тај угао (слика 24).[8]

```
void Control_motor(){
  int n;
  if(currentAngle != angle){
      if(currentAngle < angle){
        digitalWrite(dirPin, HIGH);
        n = angle - currentAngle;
        numstep = n / stepPerAngle;
        }
        else if(currentAngle > angle){
          digitalWrite(dirPin, LOW);
          n = currentAngle - angle;
          if(angle == 0){
            n = currentAngle;
            }
            numstep = n / stepPerAngle;
        }
        for(int x = 0; x < numstep; x++){
            digitalWrite(stepPin, HIGH);
            delayMicroseconds(1000);
            digitalWrite(stepPin, LOW);
            delayMicroseconds(1000);
        }
        currentAngle = angle;
    }
    delay(500);
}
```

када је изабрани угао већи од тренутног угла палице, рачуна се њихова разлика n, и онда је број корака који су потребни да би се дошло до изабраног угла једнак n / 1,8 (угао сваког корака)

када је изабрани угао мањи од тренутног угла палице, рачуна се њихова разлика n, и онда је број корака који су потребни да би се дошло до изабраног угла једнак n / 1,8 (угао сваког корака), а ако је изабрани угао једнак 0, онда је разлика n једнака тренутном углу

функција која омогућава стално ротирање stepper мотора

Слика 24: Дефинисање рада мотора

Седми део или one_portion_state() функција ради по истом принципу као и red_light_state() функција. Она служи да покрене мотор. Добија вредности са сервера и на основу већ дефинисаног угла (angle) покреће функцију Control_motor() и ротира палицу. Сервер може да има вредност 0 или вредност 1 и за обе вредности мотор ће ротирати за исти угао (подешено је да буде 45^{o}), такође Arduino ће исписивати исти текст на сервер “Послужена је једна порција“ (слика 25).

```
//Define one portion
void one_portion_state(){
  if(one_portion == "0"){
    one_portion = "1";
    if(currentAngle = 0){
      angle = 315;
      Control_motor();
    }else if(currentAngle = 360){
      angle = 315;
      Control_motor();
    }
    server.send(200, "text/html", "Послужена је једна порција");
  }else{
    if(currentAngle = 0){
      angle = 315;
      Control_motor();
    }else if(currentAngle = 360){
      angle = 315;
      Control_motor();
    }
    server.send(200, "text/html", "Послужена је једна порција");
  }
}
```

Слика 25: Покретање мотора

Осми део или all_state() функција бележи на серверу статус (вредности 0 или 1) паљења диоде и послуживања једне порције (слика 26).

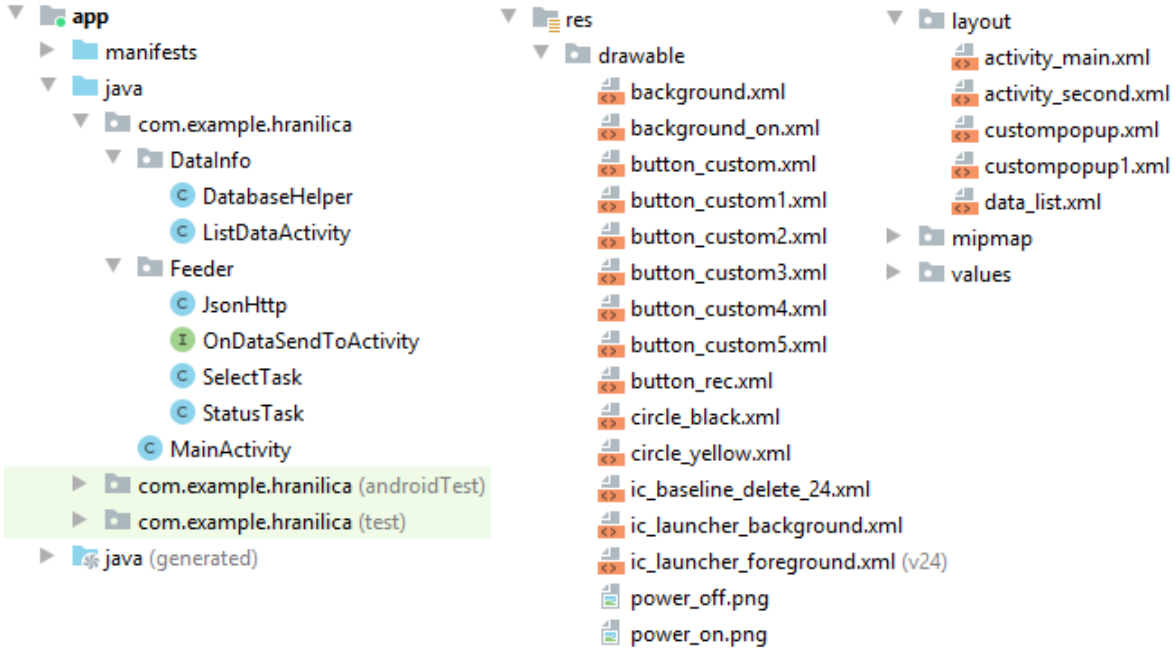
```
void all_state(){
  server.send(200, "text/html", "{'rl':'" + red_light + "','op':'" + one_portion + "'}");
}
```

Слика 26: Статус на серверу

Покретањем Arduino кода на Serial Monitoru се исписује localhost адреса на коју мора да се повеже мобилна апликација да би управљала мотором.

4.2. Мобилна апликација

Апликација је израђена у Android studio развојном окружењу, једноставно се инсталира на андроид уређај и сваки уређај може да је подржи и управља хранилицом. Апликација садржи SQLite базу и на сваком уређају ће бележити само време храњења које се задало са тог уређаја.
Структура мобилне апликације (слика 27):



Слика 27: Структура мобилне апликације

Апликација има два главна екрана:

- activity_main
- activity_second

Апликација има два подпрозора:

- custompopup
- custompopup1

- MainActivity class

Да би сви ови екрани имали своје функције потребно их је испрограмирати. Екран activity_main је повезан са класом MainActivity. У овој класи је испрограмиран највећи део апликације, јер у њој има највише активности везаних за храњење љубимца. Библиотеке које су коришћене у овој класи (слика 28).

```
package com.example.hranilica;

import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.app.TimePickerDialog;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.os.Handler;
import android.provider.AlarmClock;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;
import android.widget.ToggleButton;

import com.example.hranilica.DataInfo.DatabaseHelper;
import com.example.hranilica.DataInfo.ListDataActivity;
import com.example.hranilica.Feeder.OnDataSendToActivity;
import com.example.hranilica.Feeder.SelectTask;
import com.example.hranilica.Feeder.StatusTask;

import org.joda.time.Period;
import org.joda.time.PeriodType;
import org.json.JSONException;
import org.json.JSONObject;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;
```

Слика 28: Библиотеке у класи MainActivity

Прво је потребно дефинисати све податке који се користе у овој класи (слика 29). Осим података који су потребни за елементе почетног екрана (ImageView, Button, TextView, ProgressBar), потребно је дефинисати SQLite базу (DatabaseHelper), параметре који се користе за одбројавање тајмера (CountDownTimer, mTimeRunning, mStartTimeInMillis, mTimeLeftInMillis, mEndTime), и најбитнији део дефинисање localhost адресе помоћу које се телефон повезује са хранилицом.

```
public class MainActivity extends AppCompatActivity implements OnDataSendToActivity {

    private static final String TAG = "MainActivity";

    DatabaseHelper mDatabaseHelper;

    View bg_state;
    private Button btn_rl, first, mShowData, mChoose, mPortions;
    private TextView txt_notice, txt_network, mCountDown;
    private TextView mCurrentTime, mCurrentDate, mChooseTime,
                     mOneF, mTwoF, mThreeF, mFourF, mFiveF;
    private ProgressBar mBarCircle;
    private CountDownTimer mCountDownTimer;

    private boolean mTimerRunning;
    private long mStartTimeInMillis;
    private long mTimeLeftInMillis;
    private long mEndTime;

    String temp = "http://192.168.1.9/";

    int i = 0;
```

дефинисање базе

дефинисање параметара за изглед почетног екрана

подаци потребни за тајмер

дефинисање localhost адресе

Слика 29: Дефинисање података у MainActivity класи

Након дефинисања података потребно је повезати податке који чине изглед почетног екрана са њиховим одговарајућим id-ем. Након тога одређене елементе (TextView поља) почетног екрана се камуфлирају методом View.INVISIBLE тако да нису видљиве на почетном екрану, а користе се уписивање одговарајућих података, и на крају се дефинише календар (слика 30).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mDatabaseHelper = new DatabaseHelper( context: this);

    bg_state = findViewById(R.id.bg_status);
    txt_notice = findViewById(R.id.txt_notice);
    txt_network = findViewById(R.id.txt_network);
    btn_rl = findViewById(R.id.red);
    first = findViewById(R.id.btnFood);

    mCurrentTime = findViewById(R.id.tvCurrentTime);
    mCurrentDate = findViewById(R.id.tvCurrentDate);
    mChooseTime = findViewById(R.id.tvChooseTime);
    mCountDown = findViewById(R.id.tvCountdown);
    mChoose = findViewById(R.id.btnChooseTime);
    mShowData = findViewById(R.id.btnShowData);
    mPortions = findViewById(R.id.btnSetPorAlarm);

    mOneF = findViewById(R.id.tvOneF);
    mTwoF = findViewById(R.id.tvTwoF);
    mThreeF = findViewById(R.id.tvThreeF);
    mFourF = findViewById(R.id.tvFourF);
    mFiveF = findViewById(R.id.tvFiveF);

    mBarCircle = findViewById(R.id.BarCircle);

    mCurrentTime.setVisibility(View.INVISIBLE);
    mCurrentDate.setVisibility(View.INVISIBLE);
    mOneF.setVisibility(View.INVISIBLE);
    mTwoF.setVisibility(View.INVISIBLE);
    mThreeF.setVisibility(View.INVISIBLE);
    mFourF.setVisibility(View.INVISIBLE);
    mFiveF.setVisibility(View.INVISIBLE);

    Calendar calendar = Calendar.getInstance();
    int currentHour = calendar.get(Calendar.HOUR_OF_DAY);
    int currentMinute = calendar.get(Calendar.MINUTE);
```

Слика 30: Повезивање података са id-ем у MainActivity класи

Следећи део кода позива методу која проверава повезаност телефона са интернетом. Ако уређај није повезан са интернетом није могуће управљати хранилицом (слика 31).[8]

```
final Handler handler = new Handler();
handler.postDelayed(new Runnable() {
    @Override
    public void run() {
        if (isNetworkAvailable()) {
            bg_state.setBackground(ContextCompat.getDrawable(getApplicationContext(), background_on));
            txt_network.setText("Connect");
            btn_rl.setEnabled(true);
            mPortions.setEnabled(true);
        } else {
            bg_state.setBackground(ContextCompat.getDrawable(getApplicationContext(), background_off));
            txt_network.setText("Could't connect to the server");
            btn_rl.setEnabled(false);
            mPortions.setEnabled(false);
        }
        updateStatus();
        handler.postDelayed( r: this,  delayMillis: 2000);
    }
},  delayMillis: 3000);
```

ако је телефон повезан са интернетом поставиће се одговарајућа позадина да то прикаже, исписиваће се "Connect", и дугме Start и Portions ће бити активни

ако телефон није повезан са интернетом поставиће се одговарајућа позадина да то прикаже, исписиваће се "Could't connect to the server", и дугме Start и Portions неће бити активни

Слика 31: Проверавање конекције са интернетом у MainActivity класи

Након тога, ако је телефон повезан са интернетом могуће је укључити хранилицу и послужити одабран број порција. То је дефинисано следећим дугмићима (слика 32). Btn_rl је дугме (Старт/Стоп) за укључивање хранилице, а first је дугме (Feeding) које отвара подпрозор у којем се бира број порција за послуживање.

```
btn_rl.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) { runApp(); }
});

first.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { showPortion(); }
});
```

Слика 32: Start и Feeding дугмићи у MainActivity класи

Следећи део кода проверава тренутно време и датум, јер на основу тренутног времена се рачуна време за тајмер, то јест време преостало до изабраног времена за храњење љубимца (слика 33).

```
Thread thread1 = run() -> {
        try {
            while (!isInterrupted()) {
                Thread.sleep( millis: 1000);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        SimpleDateFormat simpleDateFormat1 = new SimpleDateFormat( pattern: "HH:mm:ss");
                        String time1 = simpleDateFormat1.format(Calendar.getInstance().getTime());
                        mCurrentTime.setText(time1);
                        SimpleDateFormat simpleDateFormat2 = new SimpleDateFormat( pattern: "dd.MM.yyyy");
                        String time2 = simpleDateFormat2.format(Calendar.getInstance().getTime());
                        mCurrentDate.setText(time2);
                    }
                });
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
};
thread1.start();
```

Слика 33: Тренутно време у MainActivity класи

У следећем делу се дефинише TimePickerDialog у коме се бира време за храњење (слика 34). Када се време изабере аутоматски се пали аларм на телефону и креће одбројавање тајмера у апликацији.

```
mChoose.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        TimePickerDialog timePickerDialog1 = new TimePickerDialog( context: MainActivity.this, android.R.style.Theme_Holo_Dialog,
                new TimePickerDialog.OnTimeSetListener() {
            @Override
            public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
                String time2 = (String.format("%02d:%02d", hourOfDay, minute));
                mChoose.setText(time2);
                Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM);
                intent.putExtra(AlarmClock.EXTRA_HOUR, hourOfDay);
                intent.putExtra(AlarmClock.EXTRA_MINUTES, minute);
                startActivity(intent);
                setAlarm();
            }
        }, currentHour, currentMinute,  is24HourView: true);
        timePickerDialog1.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
        timePickerDialog1.show();
    }
});
```

Слика 34: Бирање времена у MainActivity класи

Следећи део кода се односи на mPortions дугме (Portions) које отвара подпрозор у којем се бира број порција које ће се послужити у изабрано време (слика 35). Приликом избора броја порција које треба да се послуже, у одговарајуће TextView поље се уписује вредност “1”, па сваким притиском на дугме Portions све вредности из TextView поља се бришу и поново се бира број порција које ће се послужити, тако да није могуће да буде укључено више дугмића са изабраним порција у исто време.

```
mPortions.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showPortionAlarm();
        mOneF.setText("");
        mTwoF.setText("");
        mThreeF.setText("");
        mFourF.setText("");
        mFiveF.setText("");
    }
});
```

Слика 35: Portions дугме у MainActivity класи

Када крене одбројавање тајмера, двокликом на тајмер - време стаје, и могуће је поново изабрати време у које ће се послужити порција (слика 36). Није могуће рестартовати тајмер или наставити одбројавање од паузираног времена, већ мора да се изабере ново време.

```
mCountDown.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        i++;
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                if (i == 2) {
                    if (mTimerRunning) {
                        pauseTimer();
                    }
                }
                i = 0;
            }
        }, delayMillis: 500);
    }
});
```

Слика 36: Заустављање тајмера у MainActivity класи

Дефинисање дугмета за приказивање базе у којој се смешта свако храњење љубимца (слика 37).

```
    mShowData.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent( packageContext: MainActivity.this, ListDataActivity.class);
            startActivity(intent);
        }
    });
}
```

Слика 37: Дугме за приказивање базе у MainActivity класи

isNetworkAvailable() метода проверава повезаност телефона са интернетом (слика 38).

```
    private boolean isNetworkAvailable() {
        ConnectivityManager connectivityManager
                = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
        return activeNetworkInfo != null && activeNetworkInfo.isConnected();
    }
```

Слика 38: isNetworkAvailable() метода у MainActivity класи

updateStatus метода служи да ажурира стања у којима се тренутно налазе мотор и диода, вредности “0” или “1” (слика 39).

```
private void updateStatus() {
    String url_rl = temp + "status";
    StatusTask task = new StatusTask(url_rl,  activity: this);
    task.execute();
}
```

Слика 39: updateStatus() метода у MainActivity класи

updateButtonStatus() метода узима податке из json фајла, вредности “0” или “1”. Оне се могу прочитати из updateStatus() методе. Приликом сваког клика на дугме Start мењају се наизменично вредности “0” или “1” у json фајлу и на основу прочитане вредности се приказује одговарајућа позадина дугмета, и активира дугме Feeding (слика 40).

```
@Override
public void sendData(String str) {
    updateButtonStatus(str);
}

private void updateButtonStatus(String jsonStrings) {
    try {
        JSONObject json = new JSONObject(jsonStrings);

        String red_light = json.getString( name: "rl");

        if (red_light.equals("1")) {
            btn_rl.setCompoundDrawablesWithIntrinsicBounds( left: 0,  top: 0,  right: 0, R.drawable.power_on);
            first.setEnabled(true);
            txt_notice.setText("Hranilica je uključena!");
        } else {
            btn_rl.setCompoundDrawablesWithIntrinsicBounds( left: 0,  top: 0,  right: 0, R.drawable.power_off);
            first.setEnabled(false);
            txt_notice.setText("Hranilica je isključena!");
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Слика 40: updateButtonStatus() метода у MainActivity класи

Након сваког клика на дугмиће Start и Feeding, учитава се одговарајући линк. Свако учитавање линка поставља вредности “0” или “1” у json фајл и они се користе за укључивање/искључивање хранилице и храњење (слика 41).

```
void runApp() {
    String url_rl = temp + "red_light";
    SelectTask task = new SelectTask(url_rl);
    task.execute();
    updateStatus();
}
```

```
void Portion() {
    String url_rl = temp + "one_portion";
    SelectTask task = new SelectTask(url_rl);
    task.execute();
    updateStatus();
}
```

Слика 41: runApp() и Portion() методе у MainActivity класи

Следећи део кода покреће Portion() методу и тако покреће хранилицу. За једну порцију, Portion() метода се само једном позива и храњење се уписује у базу (слика 42, наранџасти део), док код две порције Portion() метода се позива два пута у интервалу од три секунде. Након истека треће секунде и послуживања другог дела порције, апликација је подешена да сачека још једну секунду и након тога даје сигнал да је могуће поново изабрати нову порцију за послуживање. Такође приликом послуживања две порције, храњење се бележи у базу (слика 42, зелени део). Одабир већег броја порција функционише на исти начин као и одабир две порције, тако да између сваког послуживања једне појединачне порције постоји интервал од три секунде. Тако на пример за одабир четри порције, њихово послуживање ће трајати девет секунди и након истека девете секунде и послуживања четвртог дела порције апликација је подешена да сачека још једну секунду и након тога даје сигнал да је могуће поново изабрати нову порцију за послуживање.

```
void onePortion() {
    Portion();
    Toast.makeText(getApplicationContext(), text: "Posluzena je jedna porcija", Toast.LENGTH_SHORT).show();
    String foodTime = mCurrentTime.getText().toString();
    String foodDate = mCurrentDate.getText().toString();
    SaveData( newEntry: "Posluzena je jedna porcija " + foodDate + " u " + foodTime);
}
```

приликом одабира једне порције, чита се тренутно време и датум и уз коментар "Poslužena je jedna porcija " се уписује у базу.

```
void twoPortions() {
    Handler handler = new Handler();
    Portion();
    handler.postDelayed(mRunnablePortion, delayMillis: 3000);
    Toast.makeText(getApplicationContext(), text: "Posluzene su dve porcije", Toast.LENGTH_SHORT).show();
    String foodTime = mCurrentTime.getText().toString();
    String foodDate = mCurrentDate.getText().toString();
    SaveData( newEntry: "Posluzene su dve porcije " + foodDate + " u " + foodTime);
    btn_rl.setBackgroundResource(R.drawable.button_custom3);
    first.setBackgroundResource(R.drawable.button_custom3);
    first.setText("Wait 4 seconds");
    handler.postDelayed(mRunnableOnOff, delayMillis: 4000);
}
```

приликом одабира две порције, једна се послужује истог тренутка, а друга преко функције mRunnablePortion се послужује после 3 секунде. Такође се чита тренутно време и датум и уз коментар "Poslužene su dve porcije " се уписује у базу. Приликом одабира две порције за послуживање на екрану се поставља сигнал да не треба користити дугмиће Start и Feeding 4 секунде

Слика 42: Позивање на храњење у MainActivity класи

Следећи део кода служи да дефинише функцију која ће се позивати након постављеног delay-a (слика 43). У зависности од тога шта је потребно да се уради позива се одговарајућа функција и поставља се време за које ће се она извршити.

Слика 43: Подешавање фукција за дилеј у MainActivity класи

showPortion() метода служи да дефинише рад првог подпрозора (слика 44). Овај део кода управља custompopup подпрозором, и ту се дефинишу подаци који се налазе на том подпрозору (повезују се са својим id-ем). У њему се налази пет дугмића којима се бира број порција за послуживање. Приликом одабира одређеног броја порција покреће се одговарајућа метода у којој је дефинисан број порција (слика 45).

```
public void showPortion() {
    final AlertDialog.Builder alert = new AlertDialog.Builder( context: MainActivity.this);
    View mView = getLayoutInflater().inflate(R.layout.custompopup,  root: null);

    Button mOneP = mView.findViewById(R.id.oneP);
    Button mTwoP = mView.findViewById(R.id.twoP);
    Button mThreeP = mView.findViewById(R.id.threeP);
    Button mFourP = mView.findViewById(R.id.fourP);
    Button mFiveP = mView.findViewById(R.id.fiveP);

    alert.setView(mView);
    final AlertDialog alertDialog = alert.create();
    alertDialog.setCanceledOnTouchOutside(false);
```

Слика 44: Подешавање првог подпрозора у MainActivity класи

```
mOneP.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        onePortion();
        alertDialog.dismiss();
    }
});
```

```
mTwoP.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        twoPortions();
        alertDialog.dismiss();
    }
});
```

Слика 45: Одабир порције у MainActivity класи

У следећем делу кода се дефинише функција за рад другог подпрозора (слика 46). Овај део кода управља custompopup1 подпрозором, и ту се дефинишу подаци који се налазе на том подпрозору (повезују се са својим id-ем). У њему се налази пет дугмића којима се бира број порција за послуживање у изабрано време.

```
public void showPortionAlarm() {
    final AlertDialog.Builder alert1 = new AlertDialog.Builder( context: MainActivity.this);
    View mView = getLayoutInflater().inflate(R.layout.custompopup1,  root: null);

    ToggleButton nOneP = mView.findViewById(R.id.onePo);
    ToggleButton nTwoP = mView.findViewById(R.id.twoPo);
    ToggleButton nThreeP = mView.findViewById(R.id.threePo);
    ToggleButton nFourP = mView.findViewById(R.id.fourPo);
    ToggleButton nFiveP = mView.findViewById(R.id.fivePo);

    alert1.setView(mView);
    final AlertDialog alertDialog1 = alert1.create();
    alertDialog1.setCanceledOnTouchOutside(false);
```

Слика 46: Подешавање другог подпрозора у MainActivity класи

Приликом одабира одређеног броја порција уписује се вредност 1 у одговарајуће TextView поље (постоји 5 дугмића за различите порције и за свако дугме постоји по једно TextView поље)(слика 47). Приликом притиска на дугме Portions којим се улази у подпрозор за одабир броја порција, из сваког TextView поља се бришу вредности и приликом избора броја порција, увек ће само у једном пољу бити уписана вредност 1, тако да није могуће да у исто време буду укључена два дугмета са изабраним порцијама.

```
nOneP.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            mOneF.setText("1");
            mChoose.setEnabled(true);
            Toast.makeText(getApplicationContext(),  text: "Izabrana je jedna porcija.", Toast.LENGTH_SHORT).show();
            alertDialog1.dismiss();
        } else {
            mOneF.setText("");
            mChoose.setEnabled(false);
        }
    }
});
```

ако се изабере једна порција за послуживање, у прво TextView поље ће се уписати вредност "1", док ће се у свим осталим TextView пољима уписати вредност 0. Након тога активираће се дугме за одабир времена и приказаће се коментар "Izabrana je jedna porcija". Ако се изабере две порције за послуживање у друго TextView поље ће се уписати вредност "1", док ће се у свим осталим TextView пољима уписати вредност "0". На тај начин добија се знак колико порција треба бити послужено

Слика 47: Одабир порције која се послужује у изабрано време у MainActivity класи

Следећи део кода служи да израчуна време за тајмер. Он узима вредност тренутног времена и вредност времена које се изабере, рачуна колика је разлика између њих (колико је времена преостало од тренутног до изабраног времена) и

уписује вредност у одговарајуће TextView поље које га памти (слика 48). Приликом уписа вредности у TextView поље, аутоматски се време сређује за тајмер и креће да одбројава (слика 49).

```
public void setAlarm() {
    String sTime = mChoose.getText().toString();
    String eTime = mCurrentTime.getText().toString();
    SimpleDateFormat simpleDateFormat1 = new SimpleDateFormat( pattern: "HH:mm");
    try {
        Date date1 = simpleDateFormat1.parse(sTime);
        Date date2 = simpleDateFormat1.parse(eTime);

        long startTime = date1.getTime();
        long endTime = date2.getTime();

        if (startTime <= endTime) {
            Period period = new Period(startTime, endTime, PeriodType.seconds());
            String secound1 = String.valueOf(86400 - Math.abs(period.getSeconds()));
            mChooseTime.setText(secound1);
            Toast.makeText(getApplicationContext(), text: "Vreme je podeseno.", Toast.LENGTH_SHORT).show();
        } else {
            Period period = new Period(startTime, endTime, PeriodType.seconds());
            String secound1 = String.valueOf(Math.abs(period.getSeconds()));
            mChooseTime.setText(secound1);
            Toast.makeText(getApplicationContext(), text: "Vreme je podeseno.", Toast.LENGTH_SHORT).show();
        }
    } catch (ParseException e) {
        e.printStackTrace();
    }
```

узима тренутно и изабрано време која су уписана у одговарајућа TextView поља

поставља почетно и крајње време које је узео из TextView поља

део у којем се рачуна разлика између изабраног и тренутног времена и добијени резултат се уписује у одговарајуће TextView поље

Слика 48: Рачунање времена у MainActivity класи

```
    }
    String input = mChooseTime.getText().toString();
    if (input.length() == 0) {
        return;
    }
    long millisInput = Long.parseLong(input) * 1000;
    if (millisInput == 0) {
        return;
    }
    setTime(millisInput);
    mChooseTime.setText("");
    if (mTimerRunning) {
        pauseTimer();
    } else {
        startTimer();
    }
}
```

део који додељује тајмеру вредност, и аутоматски почиње да одбројава

Слика 49: Стартовање тајмера у MainActivity класи

У следећем делу кода дефинишу се милисекунде да би се израчунало и одбројавало време. Метода startTimer() покреће тајмер када се изабере време и извршава одговарајућу наредбу када време истекне (слика 50). Када истекне време проверава се у којем TextView пољу је уписана вредност 1, на основу које се узима жељени број порција за послуживање, затим се после пет секунди пали хранилица, а после осам секунди испоручује жељени број порција (слика 51). У зависности од броја порција, између сваке појединачне порције постоји интервал од три секунде и после последње послужене порције одбројава се три секунде након које се хранилица гаси.

```
private void setTime(long milliseconds) {
    mStartTimeInMillis = milliseconds;
    resetTimer();
}
```

постављено време у тајмеру се гледа кроз милисекунде

```
private void startTimer() {
    mEndTime = System.currentTimeMillis() + mTimeLeftInMillis;
    mCountDownTimer = new CountDownTimer(mTimeLeftInMillis,  countDownInterval: 1000) {
        @Override
        public void onTick(long millisUntilFinished) {
            mTimeLeftInMillis = millisUntilFinished;
            updateCountDownText();
            mBarCircle.setProgress((int) millisUntilFinished / 1000);
        }
```

метода стартује тајмер, ажурира време у тајмеру и ажурира крог око тајмера који прати његово одбројавање

Слика 50: Покретање тајмера у MainActivity класи

```
        Handler mHandler = new Handler();

        @Override
        public void onFinish() {
            mTimerRunning = false;
            updateWatchInterface();
            setProgressBar();
            mChoose.setEnabled(false);
            if (isNetworkAvailable()) {
                String input1 = mOneF.getText().toString();
                String input2 = mTwoF.getText().toString();
                String input3 = mThreeF.getText().toString();
                String input4 = mFourF.getText().toString();
                String input5 = mFiveF.getText().toString();

                if (input1.length() == Integer.parseInt( s: "1")) {
                    mHandler.postDelayed(mRunnableStart,  delayMillis: 5000);
                    mHandler.postDelayed(mRunnableOne,  delayMillis: 8000);
                    mHandler.postDelayed(mRunnableStart,  delayMillis: 11000);
                    mOneF.setText("");
                }
            }
        }
    }.start();
    mTimerRunning = true;
    updateWatchInterface();
}
```

након истека тајмера проверава се у којем TextView пољу је уписана вредност "1" и у зависности од тога се послужује одговарајући број порција

Слика 51: Послуживање порција након истека тајмера у MainActivity класи

Следећи део кода служи да дефинише функције које ће се позивати након неког времена (поставља се временски delay) после истека тајмера (слика 52). Увек се прво позива mRunnableStart функција да би се укључила хранилица, а након тога у зависности од тога колико порција треба да се послужи позива се одговарајућа функција са бројем порција и на крају се опет позива mRunnableStart функција да би се искључила хранилица. За сваку фунцкију се поставља време (delay) за које ће се оне извршити.

```
private Runnable mRunnableStart = new Runnable() {
    @Override
    public void run() {
        runApp();
    }
};

private Runnable mRunnableTwo = new Runnable() {
    @Override
    public void run() {
        twoPortions();
    }
};

private Runnable mRunnableOne = new Runnable() {
    @Override
    public void run() {
        onePortion();
    }
};

private Runnable mRunnableThree = new Runnable() {
    @Override
    public void run() {
        threePortions();
    }
};
```

Слика 52: Подешавање функција за delay које се извршавају након истека тајмера у MainActivity класи

Следећи део служи да се манипулише тајмером. Он паузира и заустави тајмер када се двокликом притисне на њега (слика 53).

```
private void pauseTimer() {
    mCountDownTimer.cancel();
    mTimerRunning = false;
    updateWatchInterface();
    setProgressBar();
    mCountDown.setText("00:00");
    mChoose.setText("Choose time");
}

private void resetTimer() {
    mTimeLeftInMillis = mStartTimeInMillis;
    updateCountDownText();
    updateWatchInterface();
    mChoose.setEnabled(false);
    mCountDown.setText("00:00");
}
```

Слика 53: Методе за паузирање и заустављање тајмера у MainActivity класи

updateCountDownText() метода претвара разлику између тренутног и изабраног времена у милисекунде које одбројавају у тајмеру. Време у тајмеру је приказано у нормалном формату hh:mm:ss (слика 54).

```
private void updateCountDownText() {
    int hours = (int) (mTimeLeftInMillis / 1000) / 3600;
    int minutes = (int) ((mTimeLeftInMillis / 1000) % 3600) / 60;
    int seconds = (int) (mTimeLeftInMillis / 1000) % 60;
    String timeLeftFormatted;
    if (hours > 0) {
        timeLeftFormatted = String.format(Locale.getDefault(),
                format: "%d:%02d:%02d", hours, minutes, seconds);
    } else {
        timeLeftFormatted = String.format(Locale.getDefault(),
                format: "%02d:%02d", minutes, seconds);
    }
    mCountDown.setText(timeLeftFormatted);
}
```

време у тајмеру се претвара у милисекунде

ако је разлика између изабраног и тренутног времена већа од једног сата, у тајмеру ће се приказивати формат времена са сатима, а ако је разлика мања од једног сата приказиваће се формат времена без сати

Слика 54: Подешавање тајмер одбројавања у MainActivity класи

updateWatchInterface() метода дефинише изглед интерфејса приликом рада тајмера. Када се тајмер покрене нестају дугмићи за одабир порција и поновно подешавање времена све док се тајмер не заустави или истекне (слика 55).

```
private void updateWatchInterface() {
    if (mTimerRunning) {
        mChoose.setVisibility(View.INVISIBLE);
        mPortions.setVisibility(View.INVISIBLE);
    } else {
        mChoose.setVisibility(View.VISIBLE);
        mPortions.setVisibility(View.VISIBLE);
    }
}
```

приликом одбројавања тајмера дугмићи Portions и Choose time ће се искључити са екрана тако да није могуће поново изабрати број порција и време послуживања док се аларм не заустави

Слика 55: Изглед интерфејса приликом рада тајмера у MainActivity класи

Следеће две функције контролишу рад тајмера (слика 56)(слика 57).

```
@Override
protected void onStop() {
    super.onStop();

    SharedPreferences prefs = getSharedPreferences( name: "prefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putLong("startTimeInMillis", mStartTimeInMillis);
    editor.putLong("millisLeft", mTimeLeftInMillis);
    editor.putBoolean("timerRunning", mTimerRunning);
    editor.putLong("endTime", mEndTime);
    editor.apply();
    if (mCountDownTimer != null) {
        mCountDownTimer.cancel();
    }
}
```

Слика 56: onStop функција у MainActivity класи

```
@Override
protected void onStart() {
    super.onStart();

    SharedPreferences prefs = getSharedPreferences( name: "prefs", MODE_PRIVATE);
    mStartTimeInMillis = prefs.getLong( key: "startTimeInMillis",  defValue: 600000);
    mTimeLeftInMillis = prefs.getLong( key: "millisLeft", mStartTimeInMillis);
    mTimerRunning = prefs.getBoolean( key: "timerRunning",  defValue: false);
    updateCountDownText();
    updateWatchInterface();
    setProgressBar();
    if (mTimerRunning) {
        mEndTime = prefs.getLong( key: "endTime",  defValue: 0);
        mTimeLeftInMillis = mEndTime - System.currentTimeMillis();
        if (mTimeLeftInMillis < 0) {
            mTimeLeftInMillis = 0;
            mTimerRunning = false;
            updateCountDownText();
            updateWatchInterface();
        } else {
            startTimer();
        }
    }
}
```

Слика 57: onStart функција у MainActivity класи

setProgressBar() метода дефинише круг који је повезан са тајмером и како време одбројава тако се круг ажурира (смањује)(слика 58).[9]

```
private void setProgressBar() {
    mBarCircle.setMax((int) mStartTimeInMillis / 1000);
    mBarCircle.setProgress((int) mStartTimeInMillis / 1000);
}
```

Слика 58: Дефинисање круга за тајмер у MainActivity класи

saveData() метода служи за уписивање података у SQLite базу (слика 59). Бележи се свако храњење и време у које је порција послужена.

```
    private void SaveData(String newEntry) {
        boolean insertData = mDatabaseHelper.addData(newEntry);
    }
}
```

Слика 59: Чување података у SQLite бази у MainActivity класи

- DatabaseHelper class

Да би се чували подаци о сваком храњењу у SQLite бази потребно је конструисати DatabaseHelper, који ће управљати табелом са подацима. Креирање табеле се врши као на слици 60. Дефинише се назив табеле, број колона и подаци који ће се уписивати у те колоне. Након тога се креира табела која се сама ажурира када се подаци пакују у њу.

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final String TAG = "DatabaseHelper";

    private static final String TABLE_NAME = "data_information";
    private static final String COL1 = "ID";
    private static final String COL2 = "information";

    public DatabaseHelper(Context context) { super(context, TABLE_NAME, factory: null, version: 1); }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String createTable = "CREATE TABLE " + TABLE_NAME + " (ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                COL2 + " TEXT)";
        db.execSQL(createTable);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int i, int i1) {
        db.execSQL("DROP IF TABLE EXISTS "+ TABLE_NAME);
        onCreate(db);
    }
```

дефинисање података који ће се уносити у табелу

креирање табеле

апдејтовање табеле приликом додавања нових података

Слика 60: Креирање SQLite базе

Додавање података у SQLite базу се врши као на слици 61. Сваки податак који се додаје у табелу има свој јединствен id, а табела је сортирана у растућем поретку id-а.

```
public boolean addData(String item) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put(COL2, item);

    Log.d(TAG, msg: "addData: Adding " + item + "to " + TABLE_NAME);

    long result = db.insert(TABLE_NAME, nullColumnHack: null, contentValues);

    if (result == -1) {
        return false;
    } else {
        return true;
    }
}
```

Слика 61: Додавање података у SQLite базу

Следећи део кода служи да препозна сваки појединачни податак са својим јединственим id-ем када се кликне на њега (слика 62). То допушта да се манипулише сваким појединачним податком.

```
public Cursor getData() {
    SQLiteDatabase db = this.getWritableDatabase();
    String query = "SELECT * FROM " + TABLE_NAME;
    Cursor data = db.rawQuery(query,  selectionArgs: null);
    return data;
}
```

означава све податке из табеле

```
public Cursor getItemID(String information) {
    SQLiteDatabase db = this.getWritableDatabase();
    String query = "SELECT " + COL1 + " FROM " + TABLE_NAME +
            " WHERE " + COL2 + " = '" + information + "'";
    Cursor data = db.rawQuery(query,  selectionArgs: null);
    return data;
}
```

означава сваки појединачни податак из табеле

Слика 62: Препознавање података из SQLite базе

Сваким податком из табеле може се манипулисати на различите начине, али у овом пројекту је потребно само да могу да се обришу ако су вишак (слика 63). То је могуће уз помоћ методе deleteInformation().

```
    public void deleteInformation(int id, String information) {
        SQLiteDatabase db = this.getWritableDatabase();
        String query = "DELETE FROM " + TABLE_NAME + " WHERE "
                + COL1 + " = '" + id + "'" + " AND " + COL2 +
                " = '" + information + "'";
        Log.d(TAG,  msg: "deleteInformation: query: " + query);
        Log.d(TAG,  msg: "deleteInformation: Deleting " + information + " from database.");
        db.execSQL(query);
    }
}
```

Слика 63: Брисање података из SQLite базе

- ListDataActivity class

SQLite базу контолише ListDataActvity класа. Она управља екраном activity_second, извршава задатке које корисник задаје (позива методе које су дефинисане у DatabaseHelper-у), а који су везани за базу. Подаци који се чувају на SQLite бази, уписују се у listView.

Библиотеке које су се користиле у ListDataActivity класи (слика 64).

```
package com.example.hranilica.DataInfo;

import android.annotation.SuppressLint;
import android.database.Cursor;
import android.graphics.Color;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import androidx.coordinatorlayout.widget.CoordinatorLayout;

import com.example.hranilica.R;
import com.google.android.material.snackbar.Snackbar;

import java.util.ArrayList;
```

Слика 64: Библиотеке у класи ListDataActivity

Прво је потребно дефинисати све податке који се користе у овој класи. Треба дефинисати елементе који су потребни за изглед другог екрана екрана (activity_second)(ListView, TextView, CoordinatorLayout), потребно је дефинисати SQLite базу (DatabaseHelper), и параметар за креирање ListView (ArrayList<String>)(слика 65).

```
public class ListDataActivity extends AppCompatActivity {

    private static final String TAG = "ListDataActivity";

    DatabaseHelper mDatabaseHelper;

    private ListView mListview;
    private TextView tvInfo;
    private CoordinatorLayout coordinatorLayout;

    ArrayList<String> listData;

    int i = 0;
```

Слика 65: Дефинисање података у ListDataActivity класи

После дефинисања података потребно је повезати податке који чине изглед activity_second екрана са њиховим одговарајућим id-ем, и позива се метода infoListView()(слика 66).

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);

    mDatabaseHelper = new DatabaseHelper( context: this);

    mListview = (ListView) findViewById(R.id.listView);
    tvInfo = findViewById(R.id.rowLV);
    coordinatorLayout = findViewById(R.id.coordinator_layout);

    infoListView();
}
```

Слика 66: Повезивање података са id-ем у ListDataActivity класи

ListDataActivity метода дефинише додавање (сортирање) података у табелу, као и одговарајуће поље (adapter) у које се подаци додају (слика 67).

```
private void infoListView() {
    Log.d(TAG,  msg: "infoListView: Displaying data in the ListView.");

    Cursor data = mDatabaseHelper.getData();
    listData = new ArrayList<>();

    while (data.moveToNext()) {
        listData.add(data.getString( columnIndex: 1));
    }
    ArrayAdapter adapter = new ArrayAdapter<>( context: this, R.layout.data_list, R.id.rowLV, listData);
    mListview.setAdapter(adapter);
```

Слика 67: infoListView() метода у ListDataActivity класи

У следећем делу кода дефинише се одабир сваког појединачног податка из ListView табеле, и приликом двоклика на изабрани податак, препознаје се његов id и податак се брише из табеле (SQLite базе)(слика 68). Приликом брисања податка појављује се команда UNDO која служи да податак врати у табелу (SQLite базу) ако је грешком обрисан.


```
mListview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {
        String information = adapterView.getItemAtPosition(position).toString();
        Log.d(TAG, msg: "onItemClick: You Clicked on " + information);

        i++;
        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                if (i == 2) {
                    Cursor data = mDatabaseHelper.getItemID(information);
                    int itemID = -1;
                    while (data.moveToNext()) {
                        itemID = data.getInt( columnIndex: 0);
                    }
                    if (itemID > -1) {
                        Log.d(TAG, msg: "onItemClick: The ID is: " + itemID);
                        mDatabaseHelper.deleteInformation(itemID, information);
                        adapter.remove(itemID);
                        adapter.remove(information);
                        adapter.notifyDataSetChanged();
                        Snackbar snackbar = Snackbar.make(coordinatorLayout, text: "Deleted", Snackbar.LENGTH_LONG)
                                .setAction( text: "UNDO", new View.OnClickListener() {
                                    @RequiresApi(api = Build.VERSION_CODES.P)
                                    @SuppressLint("ResourceType")
                                    @Override
                                    public void onClick(View v) {
                                        mDatabaseHelper.getItemID(information);
                                        SaveData(information);
                                        adapter.add(information);
                                        adapter.notifyDataSetChanged();
                                    }
                                }).setActionTextColor(Color.RED);
                        snackbar.show();
                    }
                }
                i = 0;
            }
        }, delayMillis: 500);
    }
});
```

Слика 68: Дефинисање бирања сваког појединачног податка из ListView табеле у ListDataActivity класи

Следећи део кода препознаје податак који треба да се упише у SQLite базу и уписује га на одговарајућу позицију у табели (слика 69).

```
    private void SaveData(String newEntry) {
        boolean insertData = mDatabaseHelper.addData(newEntry);
    }
}
```

Слика 69: SaveData() метода у ListDataActivity класи

- JsonHttp class

JsonHttp класа служи да креира захтев за повезивање са localhost адресом, да би апликација могла да прочита статус активности лед диоде и stepper мотора (у json фајл се уписују наизменично вредности “0“ или “1“ на основу којих се контролише рад хранилице)(слика 70)(слика 71).

```
public class JsonHttp {

    public static String makeHttpRequest(String url) {
        String strResult = "";

        try {
            URL u = new URL(url);
            HttpURLConnection con = (HttpURLConnection) u.openConnection();
            strResult = readStream(con.getInputStream());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return strResult;
    }
```

Слика 70: Креирање захтева за узимање вредности из json фајла у JsonHttp класи

```
    private static String readStream(InputStream in) {
        BufferedReader reader = null;
        StringBuilder sb = new StringBuilder();

        try {
            reader = new BufferedReader(new InputStreamReader(in));
            String line;
            while ((line = reader.readLine()) != null) {
                sb.append(line + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return sb.toString();
    }

    public interface AsyncResponse {
        void processFinish(String output);
    }
}
```

Слика 71: Метода која учитава вредности из json фајла у JsonHttp класи

- SelectTask class

SelectTask класа служи да препозна који статус у json фајлу на localhost адреси треба да се промени приликом одабира одговарајућег дугмета на почетном екрану (вредности које контролишу укључивање/искључивање хранилице или вредности које контролишу рад stepper мотора)(слика 72). Апликација мења вредности у json фајлу, чита их, и у зависности да ли је “0” или “1” уписано у фајл извршава одговарајућу радњу.

```
public class SelectTask extends AsyncTask<Void, Void, String> {

    private String mUrl;

    public SelectTask(String url) {
        super();
        mUrl = url;
    }

    @Override
    protected String doInBackground(Void... voids) {
        String jsonString = JsonHttp.makeHttpRequest(mUrl);
        return jsonString;
    }

    @Override
    protected void onPostExecute(String result) { super.onPostExecute(result); }
}
```

Слика 72: Класа која препознаје који статус у json фајлу треба да се промени

- StatusTask class

StatusTask класа ажурира статус у json фајлу. Када се преко SelectTask класе препозна које је дугме притиснуто, препознаје се која је вредност уписана у json фајлу (вредности “0” или “1”) у тренутку притискања дугмета, и на основу тога се мења у супротну (слика 73). Након тога се учитава тренутна вредност из josn фајла у arduino окружење и на основу ње arduino препознаје која је функција задата за извршавање (укључивање/искључивање хранилице или рад stepper мотора). Такође та вредност се учитава и у апликацији и извршава изабране функције.

```
public class StatusTask extends AsyncTask<Void, Void, String> {

    private String mUrl;

    OnDataSendToActivity dataSendToActivity;

    public StatusTask(String url, Activity activity){
        dataSendToActivity = (OnDataSendToActivity)activity;
        mUrl = url;
    }

    @Override
    protected String doInBackground(Void... params) {
        String jsonString = JsonHttp.makeHttpRequest(mUrl);
        return jsonString;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);
        dataSendToActivity.sendData(result);
    }
}
```

Слика 73: Класа која мења вредности “0” или “1” у json фајлу

- OnDataSendToActivity class

Преко ове класе (слика 74) StatusTask класа шаље вредности које треба да се промене у json фајлу.

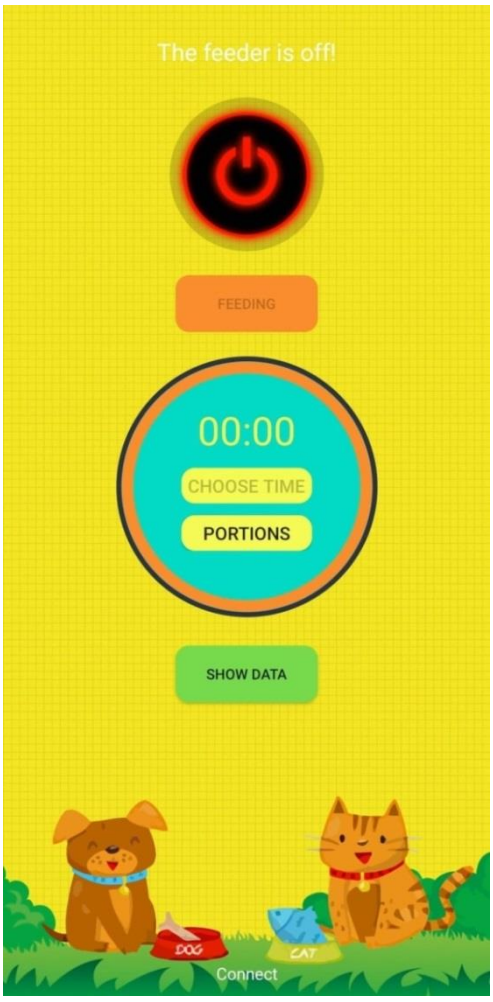
```
public interface OnDataSendToActivity {
    public void sendData(String str);
}
```

Слика 74: Класа преко које се шаљу подаци у json фајл

5. Коришћење апликације

Почетни екран апликације контролише хранилицу. Коришћење апликације је могуће након успостављања интернет конекције са уређајем. Ако уређај није повезан са интернетом на почетном екрану ће се приказивати слика 75 са љубичастом позадином која то сигнализира, и докле год је она приказана није могуће користити апликацију, могуће је само погледати време када је љубимац нахрањен. Када уређај успостави комуникацију са интернетом на почетном екрану ће се приказивати слика 76 са жутом позадином и онда се активирају команде којима се управља хранилицом.

Слика 75: Почетни екран апликације када телефон није повезан са интернетом



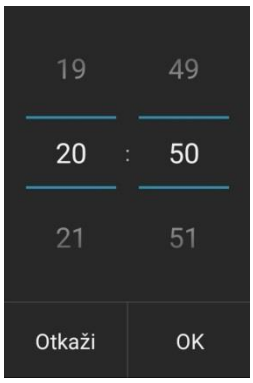
Слика 76: Почетни екран апликације када је телефон повезан са интернетом

На почетном екрану се налази дугме за укључивање и искључивање хранилице, и док се хранилица не укључи није могуће послужити порције љубимцу. Када се хранилица укључи активира се дугме Feeding, и притиском на то дугме отвара се подпрозор (слика 76) у којем се бира број порција које се послужују љубимцу.

Слика 77: Изглед подпрозора за храњење

Слика 78: Изглед подпрозора за храњење у изабрано време

Дугме Portions се активира када се уређај повеже са интернетом, у њему се бира број порција које ће се послужити у изабрано време (слика 78). Није потребно укључивати хранилицу да би изабрали број порција и време када ће се оне послужити, јер ће се хранилица сама укључити непосредно пре него што треба да се послуже порције љубимцу. Када се изабере број порција откључава се дугме Choose time да би се изабрало време у које ће се послужити изабране порције (слика 79).



Слика 79: Изглед прозора за бирање времена

Слика 80: Изглед табеле у коју се бележи време сваког храњења

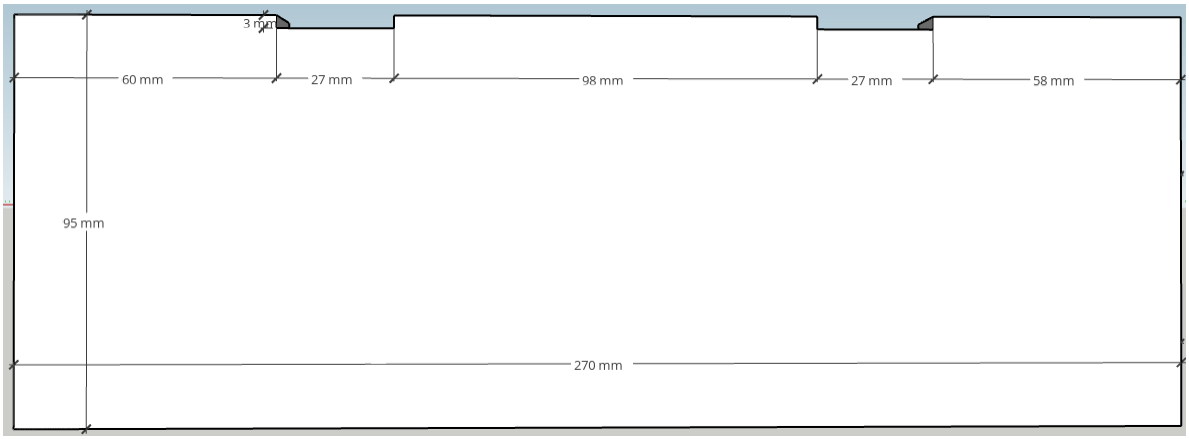
Дугме Show data служи да прикаже табелу у којој се бележи време сваког храњења животиње (слика 80). Сваки податак појединачно може да се обрише из базе двокликом на њега, а ако се грешком притисне на неки податак постоји и опција UNDO која враћа податак у базу.

6. Изглед хранилице

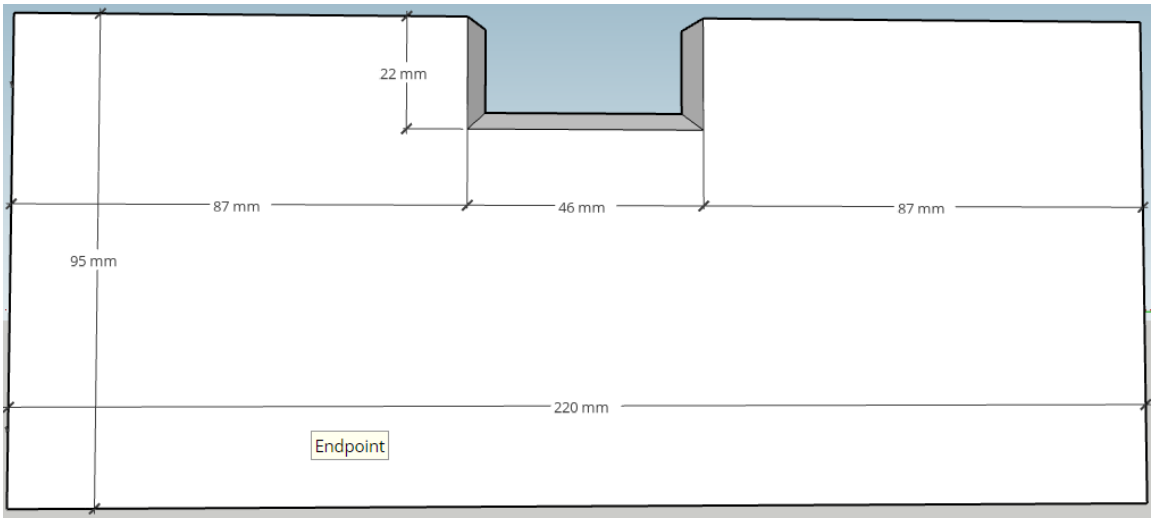
Цела хранилица је прецизно измоделирана у SketchUp програму. Делови од којих је хранилица направљена нису фиксирани и могу да се померају ако је потребно детаљно прегледање хранилице.

Кутија у којој се налазе arduino делови потребни за рад хранилице је направљена од иверице и она може да се отвара по потреби. Њена израда је трајала свега пар сати, јер су биле познате све димензије које је било потребно направити.

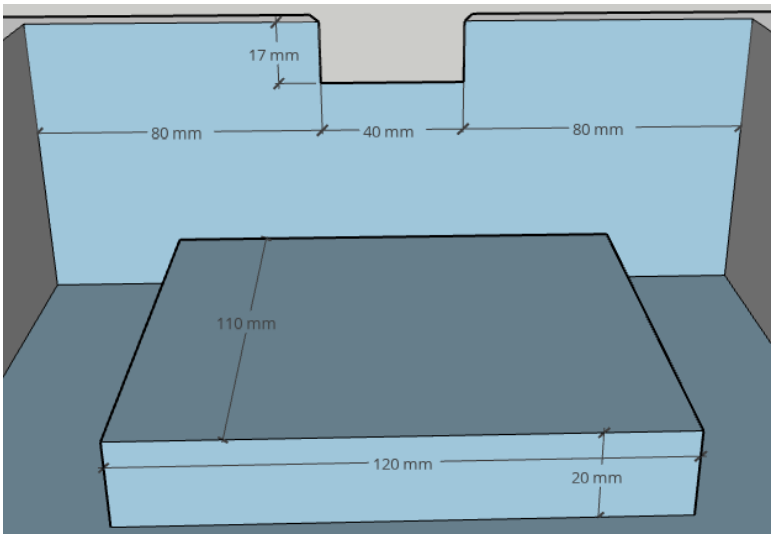
Димензије кутије су следеће:



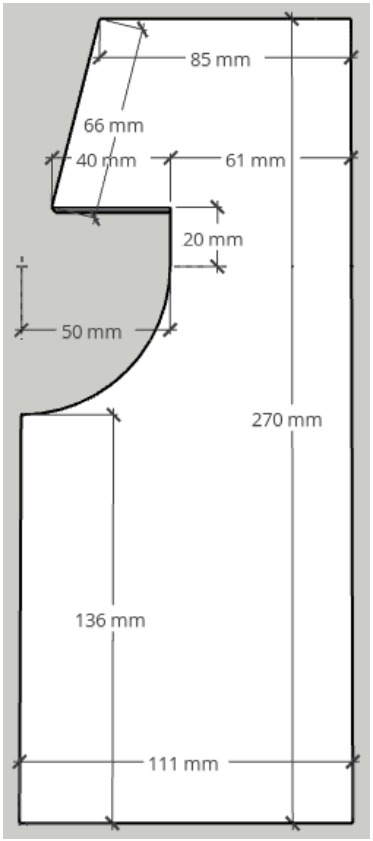
Слика 81: Димензије бочне стране кутије



Слика 82: Димензије предње стране кутије



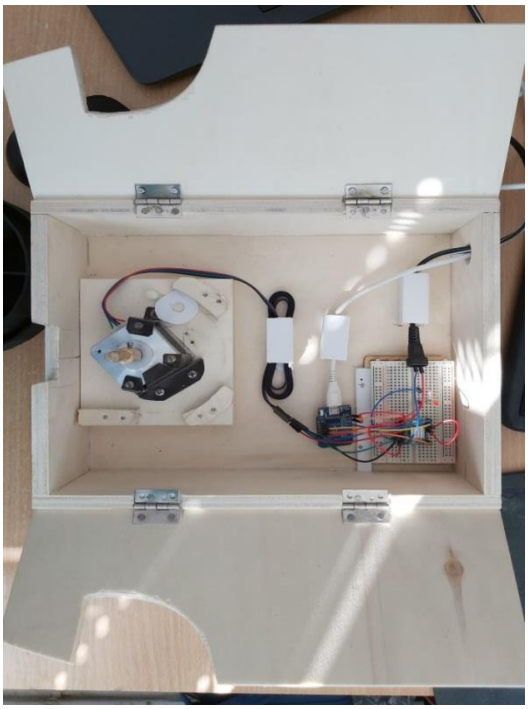
Слика 83: Димензије унутрашњости кутије



Слика 84: Димензије поклопца кутије

Изглед кутије у којој стоје ардуино делови:


Слика 85: Изглед затворене кутије

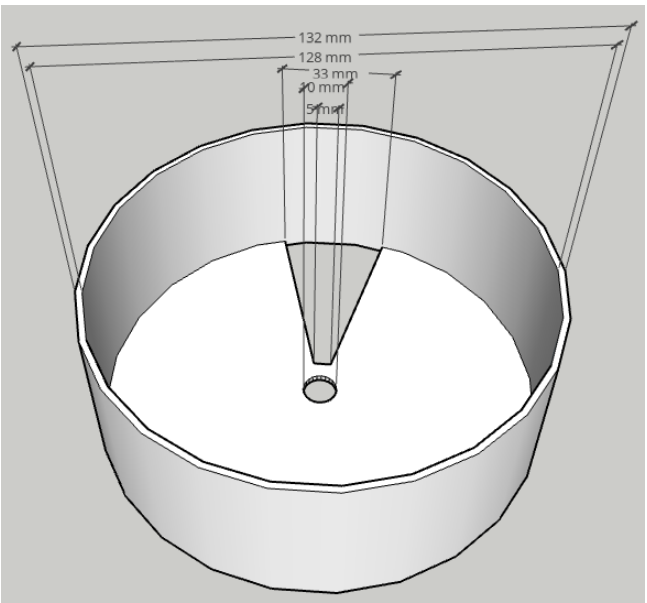

Слика 86: Изглед отворене кутије

Следећи делови су направљени уз помоћ 3D штампача. 3D штампа је моддрна технологија производње тродимензионалних објеката. У тродимензионалној штампи објекат се креира сукцесивним наношењем слојева материјала. 3D штампа представља генерално брже, јефтиније и лакше решење од других технологија производње 3D бјеката. Ова технологија производи моделе који верно опонашају изглед, утисак и функционалност производа прототипа. За израду ових делова је било потребно око 48 сати.
Постоји више техника штампања:

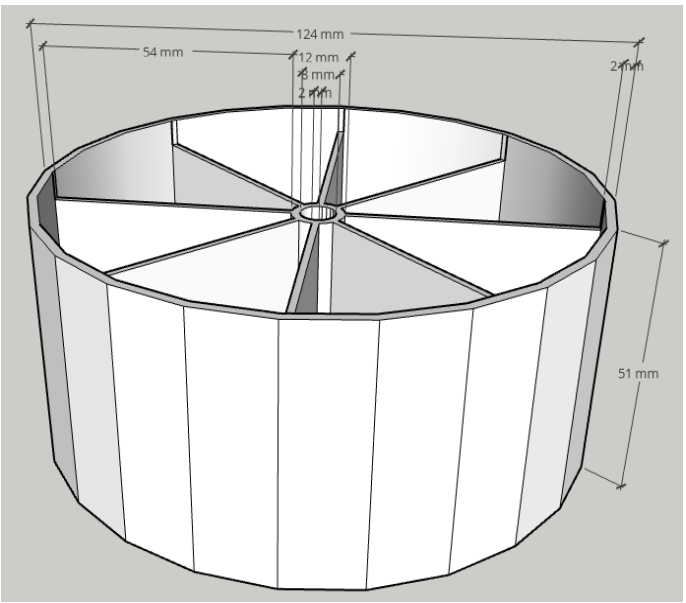
1. Inkjet
2. Fused Deposition Modeling (FDM)
3. Стереолитографија
4. Селективно ласерско синтеровање (SLS)
5. Производња објеката ламинацијом (LOM)

За израду делова за хранилицу је коришћена техника Fused Deposition Modeling. Овом методом, слојеви се добијају тако што млазница истискује танко влакно истопљене термопластике на површину за штампу. Слојеви се праве укрштено, тј. сваки слој се истискује под углом од 90° у односу на претходни. Тиме се постиже чврстина завршног модела.[10]
Ови делови се убацује у кутију. Посуда у којој стоји храна (слика 87), и у њој се налази силос који регулише број порција које треба да се послуже љубимцу (слика 88). Силос се ставља на вратило мотора које ротира за 45° и тако окреће једну по једну преграду. Делови су направљени од истопљене термопластике.

Димензије посуде за храну и силоса који регулише број порција које треба да се послуже су следеће:



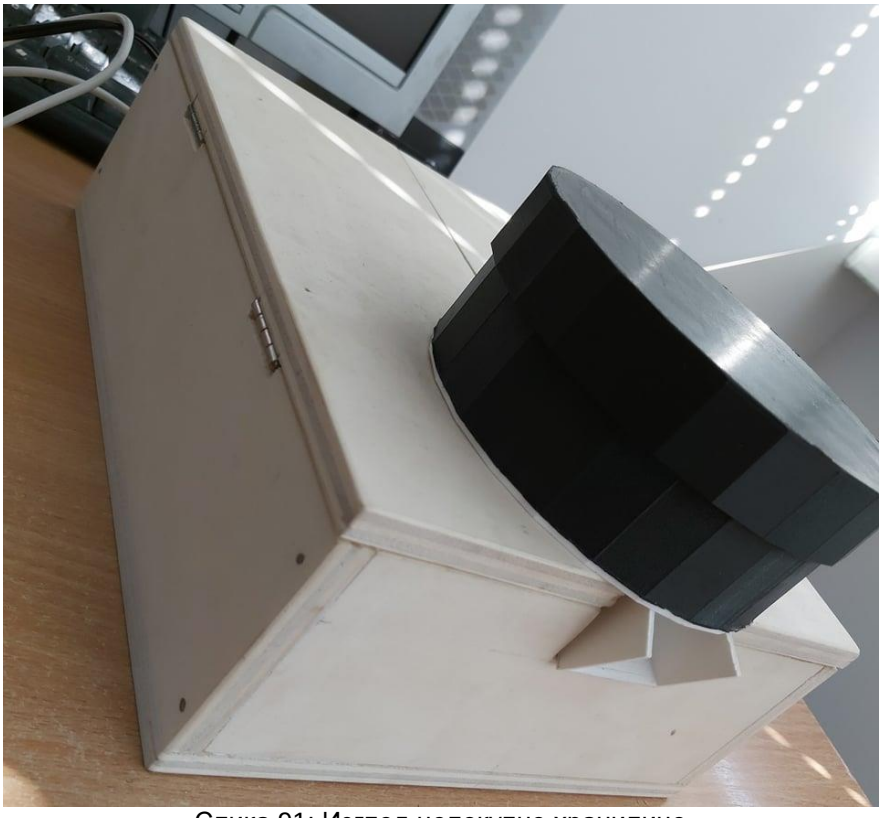
Слика 87: Димензије посуде за храну



Слика 88: Димензије силоса који регулише број порција које треба да се послуже

Слика 89: Изглед посуде у којој стоји храна

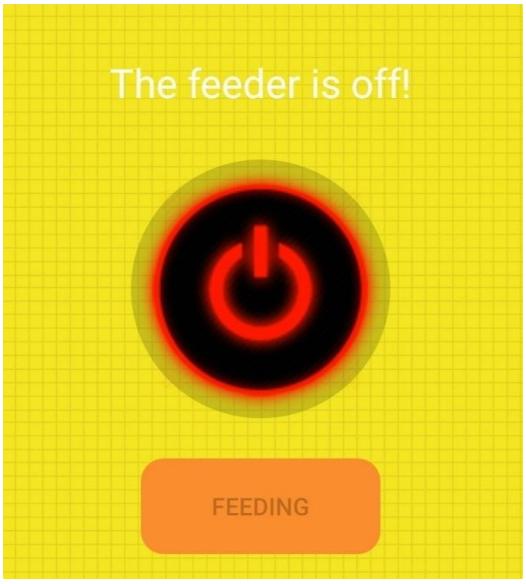
Слика 90: Изглед силоса који регулише број порција које треба да се послуже љубимцу



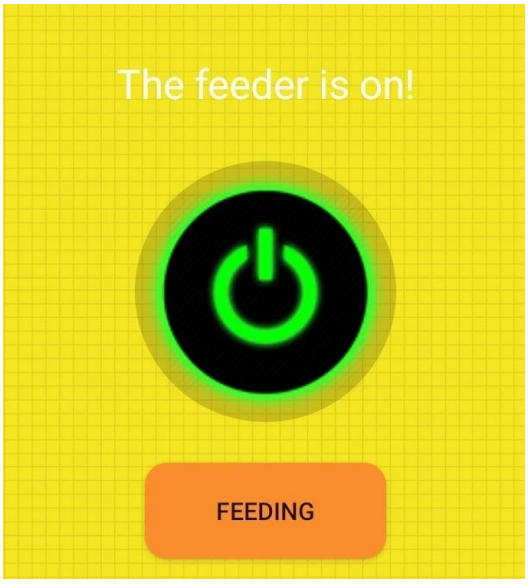
Слика 91: Изглед целокупне хранилице

7. Принцип рада хранилице

Коришћење апликације је могуће након успостављања интернет конекције са уређајем. Након тога је могуће укључити хранилицу. Када се хранилица укључи, откључава се дугме Feeding у којем се бира број порција које желите да послужите љубимцу.

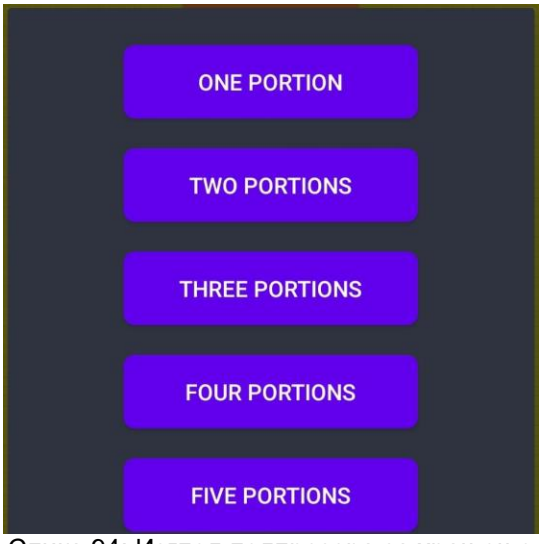


Слика 92: Хранилица је искључена



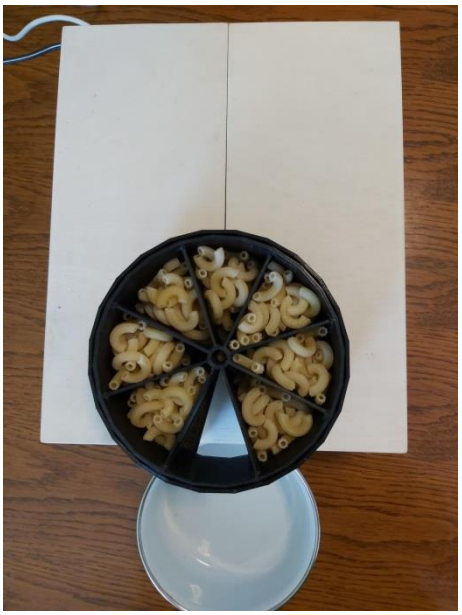
Слика 93: Хранилица је укључена

Притиском на дугме Feeding отвара се подпрозор у којем се бира број порција (слика 94).



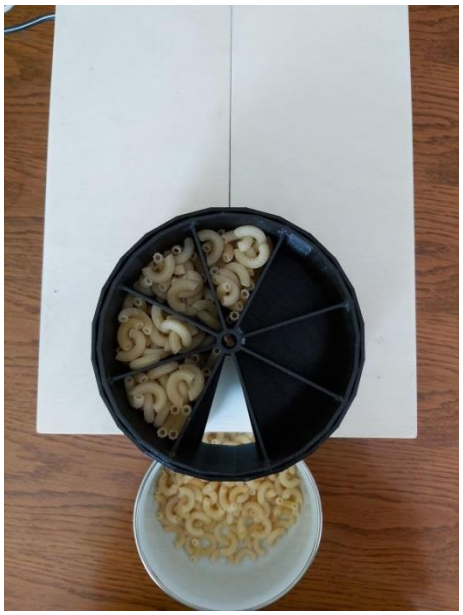
Слика 94: Изглед подпрозора за храњење

Да би се порције послужиле потребно је да се хранилица напуни одговарајућом храном (слика 94). Када је хранилица напуњена, притиском на дугме One portion мотор ротира вратило за 45° и тако допрема храну из једне преграде силоса у чинију (слика 95).



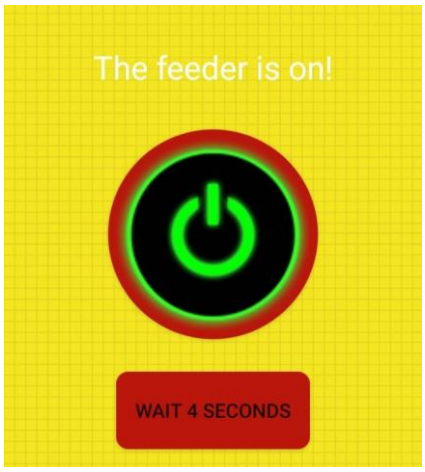
Слика 95: Изглед пуне хранилице

Слика 96: Изглед хранилице када се послужи једна порција



Слика 97: Изглед хранилице када се послуже још две порције

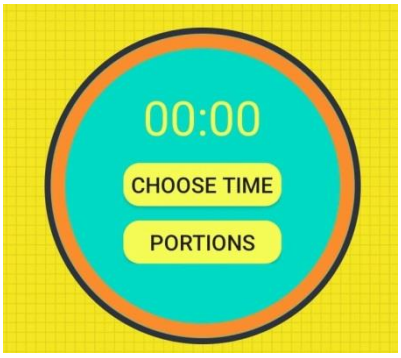
Након одабира две порције за послуживање мотор ротира вратило за 45° да би испустио храну из једне преграде силоса, након тога чека три секунде и онда ротира вратило за још 45° да би испустио храну из још једне преграде силоса (слика 96). Када се заврши процес храњења, апликација чека још једну секунду пре него што сигнализира кориснику да је могућ поновни одабир порција. У току трајања тог процеса храњења апликација сигнализира кориснику да се тренутно служи храна и да не треба у том тренутку поново бирати број порција док се не заврши предходни процес (слика 98).



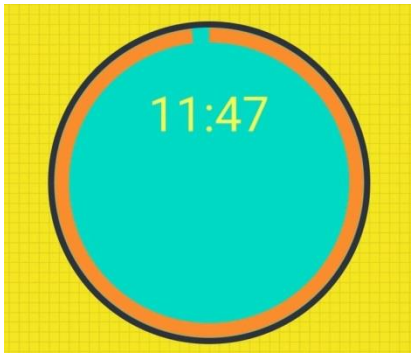
Слика 98: Апликација сигнализира да се тренутно послужују две порције љубимцу

Како апликација омогућава и заказивање термина храњења, корисник ако жели да закаже време у које ће се послужити храна треба прво изабрати број порција притиском на дугме Portions (слика 99). Након тога откључава се дугме Choose time где се бира време у које ће се послужити порције (слика 100). Када се подеси време, такође се подешава и аларм на телефону. Тајмер у апликацији креће да одбројава и тада није могуће поново изабрати број порција за послуживање или поново подесити време док тајмер не истекне, али је могуће двокликом на тајмер зауставити одбројавање и поново изабрати број порција и време у које ће се послужити (слика 101).

Слика 99: Одабир броја порција за послуживање



Слика 100: Одабир времена када ће се порције послужити



Слика 101: Одбројавање тајмера

Након истека времена апликација ће сама укључити хранилицу и послужити изабран број порција. Такође апликација сигнализира кориснику да се тренутно служи храна и да не треба у том тренутку поново бирати број порција док се не заврши предходни процес.

Закључак

Апликација се показала као прецизна, веома лака и једноставна за коришћење. Arduino компоненте су лепо упаковане тако да не кваре визуелни изглед целокупне хранилице, а у потпуности извршавају задата наређења. Апликацију може користити било који корисник андроид уређаја, који има основно знање употребе, потребно је само да је инсталира и да се повеже са интернетом. Корисник преко апликације може приступити подацима и контролисати хранилицу независно од места на ком се налази. У апликацији је све естетски срећено, и није потребно неко посебно упутство за рад са њом, већ после првог коришћења лако ће се запамтити све функције. Апликација је подложна надоградњи и ажурирању у зависности од тога шта је све потребно да се дода на хранилици.
На основу представљених технологија, поступака у реализацији и начином рада апликације може се видети да се употребом савремених технологија могу направити апликације које значајно поједностављују, убрзавају и олакшавају свакодневне животне процесе и активности.

Пројекат представља комплетан прототип који се може тестирати и ван развојног окружења.

Литература

[1] Званична Arduino веб страница, **https://www.arduino.cc/**, приступљено: фебруар 2021.

[2] Званична Android studio веб страница, **https://developer.android.com/studio/**, приступљено: фебруар 2021.

[3] **https://www.wemos.cc/en/latest/**, приступљено: март 2021.

[4] Висока школа електротехнике и рачунарства струковних студија Београд, step мотори, **file:///G:/faks/DIPL/Step%20motori.pdf**, приступљено: март 2021.

[5] Званична Last Minute Engineer веб страница, A4988 Stepper Driver, **https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/**, приступљено: март 2021.

[6] Званична Knjizara.com веб страница, Uvod u Android programiranje, **https://www.knjizara.com/pdf/142076.pdf**, приступљено: фебруар 2021

[7] Android Studio Wikipedia, **https://en.wikipedia.org/wiki/Android_Studio**, приступљено: фебруар 2021.

[8] **https://github.com/makertut/stepper-angle-ctrl/blob/master/code1.ino**, приступљено: новембар 2020.

[9] Званична Android Tutorials Hub веб страница, Android Count Down Timer, **http://www.androidtutorialshub.com/android-count-down-timer-tutorial/**, приступљено: децембар 2020.

[10] 3Д штампа Wikipedia, **https://sr.wikipedia.org/wiki/3Д_штампа**, приступљено: фебруар 2021.