Факултет инжењерских наука Универзитета у Крагујевцу

Милица, С. Петровић

ДИЗАЈН WIFI УРЕЂАЈА И ПРОГРАМИРАЊЕ МОБИЛНЕ АПЛИКАЦИЈЕ

-ОКСИМЕТАР НА ESP8266 АРДУИНО ПЛАТФОРМИ-

-АНДРОИД ПРОГРАМИРАЊЕ-

завршни рад

Крагујевац, 2018.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Програмски језици

Број индекса: 52/2015

Милица, С. Петровић

ДИЗАЈН WIFI УРЕЂАЈА И ПРОГРАМИРАЊЕ МОБИЛНЕ АПЛИКАЦИЈЕ

-ОКСИМЕТАР НА ESP8266 АРДУИНО ПЛАТФОРМИ-

-АНДРОИД ПРОГРАМИРАЊЕ-

завршни рад

Комисија за преглед и одбрану:

1. **Ред. проф. др Ненад Грујовић - ментор**
2. _____ Датум одбране:_____
3. _____ Оцена:_____

У оквиру овог завршног рада кандидат треба да изради бежични уређај који може да мери ниво кисеоника у крви и који ће комуницирати са Андроид апликацијом путем Wifi модула ESP8266.

Препоручена литература:

[1] James Steele, Nelson To: Андроид: Израда апликација помоћу пакета Андроид SDK; Микро књига, Београд, 2011.

[2] dr John Allwork: C# програмирање за Windows и Android; EHO, 2016.

[3] James Talbot, Justin McLean: Програмирање Андроид апликација; Cet, 2014.

[4] Bert van Dam: Ардуино уно: 45 пројеката за почетнике и стручњаке; EHO,2016.

[5] Simon Monk: Ардуино: увод у програмирање, превод 2. издања; Микро књига, Београд, 2017.

[6] Prof Dr Dogan Ibrahim i Ahmet Ibrahim: ESP8266 i MicroPython; EHO, 2017.

Крагујевац, 2018.

ментор:

др Ненад Грујовић, редовни професор

Резиме рада:

Циљ овог пројекта је израда бежичног уређај који може да мери ниво кисеоника у крви и који ће комуницирати са Андроид апликацијом путем Wifi модула ESP8266.

Уз помоћ софтвера Arduino IDE ћемо оспособити сензоре за комуникацију са Wifi модулом. Php програмски језик и NetBeans развојно окружење ће нам помоћи да направимо сервер који ће комуницирати са апликацијом која је развијена у Android Studio-у тако да прима податке са сервера, обрађује и приказује резултате кориснику.

Кључне речи:

Оксиметрија, модул ESP8266, Wemos D1, Arduino, Android апликација, температурни сензор, сензор MAX30100.

Summary of work:

The goal of this project is developing a wireless device that is able to measure blood oxygen level, and communicate with a mobile Android app using the ESP8266 Wifi module.

We will enable the oxygen sensor to communicate via Wifi module, using the Arduino IDE for our development. Php and NetBeans IDE will help us build a server, which will communicate with the application, which is developed using Android Studio, in such a way that it receives data from server, parses it, and displays the results to the end user.

Key words:

Oximetry, Wifi module ESP8266, Wemos D1 mini, Arduino, Android application, temperature and humidity sensor, sensor MAX30100, deep sleep.

Садржај

1. УВОД

Развој преносних технологија и бежичних уређаја је донео многобројне иновације у здравству. Уређаји који служе за праћење здравственог стања пацијента, као што су ЕКГ(електрокардиографија), оксиметри, монитори који прате откуцаје срца итд. се искључиво могу наћи у болницама, клиникама и осталим здравственим установама, пре свега јер су комплексни, скупи и непортабилни. Са појавом технологије која је базирана на бежичној конекцији, праћење одређених здравствених стања у људском телу постало је могуће и ван здравствених установа што је врло значајно за медицину и благовремено отклањање здравствених проблема.

Циљ овог пројекта јесте да се направи преносни уређај што мањих размера који ће служити човеку да уз једноставно коришћење може да прати своје или туђе здравствено стање које је повезано са нивоом кисеоника у крви.

Преко Андроид апликације, корисник ће моћи да прати ниво кисеоника у крви, а поред тога ће бити обавештен нотификацијом сваки пут када кисеоник падне испод границе нормале.

Идеја пројекта јесте да корисник може да изабере путем апликације више сензора са којих жели да прочита резултате. Он то може учинити са било ког места. У теретани, у току вожње, кући, на мору итд, све док има приступ интернету.

Преносни уређај описан у овом раду служи за мерење кисеоника у крви. Малих је димензија, једноставан за коришћење, пре свега јефтин и самим тим доступан свима. На овај начин можемо пратити стање особе која није са нама, довољно је само да имамо инсталирану апликацију у своме телефону. Уређај је посебно користан код праћења стања пацијената који нису при свести јер није потребно да се при сталном мерењу буде поред њега, већ уређај може стајати на прсту пацијента и у случају пада кисеоника, нотификацијом обавести надлежну особу која може благовремено да реагује у циљу побољшања његовог здравственог стања. У раду ће бити описана два уређаја која су повезана са Андроид апликацијом смарт телефона.

Примарни уређај јесте уређај за мерење кисеоника у крви, а секундарни је уређај за мерење температуре. Оба ће бити детаљно описана, од технологија, радних окружења која су коришћена, физичког повезивања, имплементације кода, па све до крајњих резултата где можемо видети ефикасност самих уређаја.

Уређај за мерење кисеоника у крви је базиран на модулу **Wemos D1 mini** са уграђеним **ESP8266-12f** wifi модулом и сензором **MAX30100** за мерење кисеоника у крви уз помоћ фотоплетисмографије.

За конструисање уређаја који мери температуру и влажност ваздуха, коришћена је **ARDUINO UNO** платформа, wifi модул **ESP8266-01** и сензор **DHT11** за мерење температуре и влажности ваздуха.

2. ОКСИМЕТРИЈА

Пулсна оксиметрија (pulse oximetry) је неинвазивна метода мерења оксигенације или процента хемоглобина који је засићен кисеоником. Мерење је изражено у процентима. Процедура је кратка и безболна и изводи се тако што се малим уређајем обухвати кажипрст руке при чему се приказују вредности пулса и проценат засићености артеријске крви кисеоником. Сензор се састоји од два извора светла, који се делимично апсорбују у хемоглобину у зависности од засићености или незасићености кисеоником.

Пулсна оксиметрија је индицирана у свим клиничким стањима у којима постоји вероватноћа од појаве хипоксемије. Инвазиван начин мерења подразумева узимање узорка крви из артерије ради анализе.

Важно је изабрати сензор адекватне величине за датог пацијента. Правилна величина омогућава исправно усмеравање светла ЛЕД према фотодетектору. Ако је кућиште сензора превелико може спасти са прста. Ако је кућиште сензора премало, може довести до појаве венских пулсација, које ће резултовати лажно ниским вредностима SpO2. Предност се даје самолепљивим сензорима за једнократну употребу који минимализују утицај спољашњих фактора на измерене вредности.

Нормалне вредности сатурације O2 се крећу између 95-100%. Ако постоје проблеми са плућима и срцем вредности сатурације ће бити смањене чак и у мировању. Сатурација кисеоником мања од 94% сматра се хипоксијом, а код вредности испод 90% хипоксија се сматра израженом и потребно ју је терапијски третирати.

Да би се могло констатовати стање пацијента, идеално би било да се рачуна процентуално колико хемоглобина постоји који је засићен кисеоником. Ако се деси да пацијент има велики проценат хемоглобина који није функционалан, односно није способан да веже кисеоник за себе, али има карбохемоглобин (HbCO) и метемоглобин (METHb) онда очитавање није поуздано. Функционални хемоглобин представља онај који је способан да веже кисеоник за себе и у њега спадају оксигенован хемоглобин (HbO2) и дезоксигенован хемоглобин(Hb). На основу ових података, идеално засићење крви кисеоником се може добити изразом [5]:

$$100 * \frac{HbO2}{Hb + HbO2 + HbCO + METHb + \text{други нефункционални хемоглобини}}$$

Пулсни оксиметри усвајају да не постоји нефункционалан хемоглобин у артеријској крви и зато користе мерење на основу следећег израза:

$$100 * \frac{HbO2}{Hb + HbO2}$$

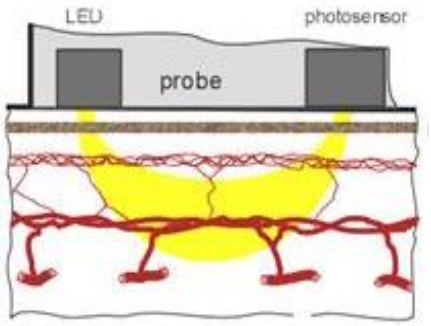
Идеално место за постављање сензора пулсног оксиметра је оно које је добро прокрвљено, релативно непокретно, лако доступно и удобно за пацијента. Најчешћа места су јагодица прстију шаке и ресица ушне шкољке. [1]

2.1. Фотоплетизмографија (Photoplethysmography-PPG)

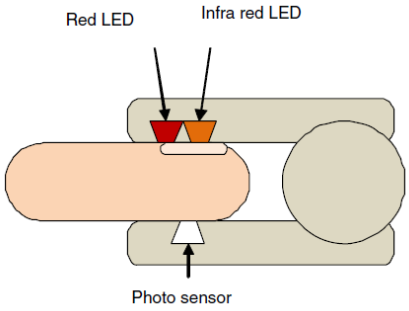
Плетизмографија је метода за мерење промена у запремини органа које могу настати од промена у току крви али и од промене количине кисеоника. Фотоплетхизмографија је облик плетизмографије која на основу оптике мери промене у волумену крви у микроваскуларном ткиву. То је неинвазивна, једноставна и приступачна техника на основу које се откривају недостаци и промене које могу лоше да утичу на здравствено стање човека.

У Фотоплетизмографији, кожа прста се осветљава преко LED-емитујуће диоде, инфра ред или црвене LED диоде. Светлост која се одбије од површине коже, сакупља се преко фотодиоде и на основу тога се одређују промене у крви. Фотодиоде се могу поставити или са супротне стране од емитовања светлости или на истој страни и тако детектује одбијену светлост.

На слици 1 је пример где се преко рефлексије детектују промене, а на слици 2 преко преноса светлости.



Слика 2: *Рефлексија*



Слика 1: *Детекција преко преноса светлости*

Фотоплетхизмографија се примењује у развоју различитих здравствених мониторинг и мерних уређаја као што су оксиметар, монитор за мерење крвног притиска итд. [2]

3. КОМПОНЕНТЕ И СОФТВЕР

3.1. ARDUINO UNO платформа

Arduino uno је микроконтролер базиран на ATmega328P и састоји се од софтверског и хардверског дела. Софтверски део развојног Arduino uno система састоји се од програма који се користе за превођење програмског језика C у асемблерски код. Хардверски део Arduino uno развојног система састоји се од развојне плоче која садржи микроконтролер ATMEL који разуме само асемблерски код. [19]

Слика 3: *Ардуино Уно*

Након пренесеног преведеног кода у меморију микроконтролера, он се извршава све док има напајања. Приликом прекида напајања или рестартовања, програмски код почиње да се извршава из почетка. Време које је потребно за обраду појединих података унутар микроконтролера краће је од милисекунде, што омогућава релативно брзу обраду података. Садржи 14 дигиталних пинова(излаза/улаза), 6 аналогних улаза, 16Mhz кварцног кристала, улаз за USB кабал, улаз за напајање и ресет дугме. Лако спајање нових модула и прикључака је кључна ствар код Arduina. Додаци – модули за Arduino се називају шилдовима (енгл.shield). Купују се готови или их сами правимо по потреби. Сврха шилдова је проширење функционалних могућности. Најпознатији шилдови су: GPS, GSM, bluetoothe, ethernet, LCD екран, RFID, итд.[18]

Готови Ардуино бордови долазе са сопственим бутлоадером који омогућава аплоадовање написаних програма у флеш меморију ардуина. Раније верзије Ардуино система су користиле серијски порт RS-232 за програмирање и комуникацију, а новији Ардуино системи данас раде преко USB- а. [7]

Радни напон	5 V
Напон напајања(препоручено)	7-12V
Максимални напон напајања(не препоручује се)	20 V
Дигитални У/И пинови	14
Аналогни У/И пинови	6
Струја по У/И пину (5 V)	40mA
Струја по У/И пину (3.3 V)	50mA
Флеш меморија	32 KB
SRAM	2 KB
EEPROM	1 KB
Радни такт	16 MHz

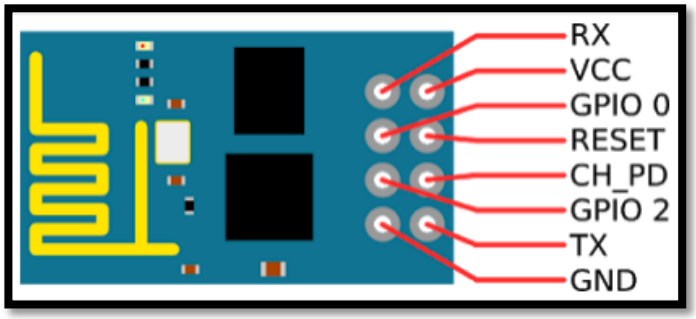
Табела 1: *Техничка спецификација микроконтролера ATmega328*

Напајање	Vin, 3.3V, 5V, GND	
Ресет	Reset	Ресетује микроконтролер
Аналогни пинови	A0 – A5	Омогућава аналогни улаз од 0-5V
Дигитални пинови	0 - 13	Као улазни/излазни пинови
Серијал пинови	0(Rx), 1(Tx)	Пријем и слање серијских података
Спољни прекиди	2, 3	За покретање прекида.
PWM	3, 5, 6, 9, 11	Омогућава 8-bit PWM излаз.
SPI	10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK)	Служи за SPI комуникацију.
TWI	A4 (SDA), A5 (SCA)	Служи за TWI комуникацију..
AREF	AREF	Обезбеђује референтни напон за улазни напон

Табела 2: *Функције пинова на Ардуину*

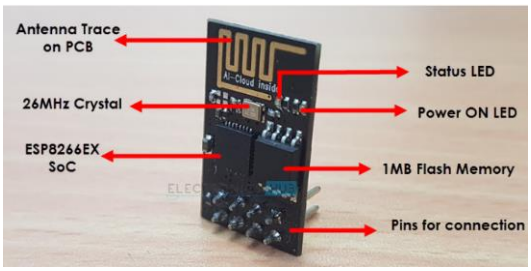
3.2. WiFi модул ESP8266-01

ESP8266-01 WIRELESS модул омогућава спајање Arduina на интернет бежичном везом. Пинови са модула се спајају на одговарајуће пинове Arduina. Потребно је пазити на напајање, код модула износи 3.3 V, док је напајање Arduina 5V.



Слика 4: *Wifi модул ESP8266-01*

ESP8266-01 модул долази са програмираним firmware-ом, што значи да је довољно да се повеже физички са Arduino уређајем и одмах се може користити. Wifi модул је врло економичан и и постао је јако популаран широм света, коришћен у разним технологијама.



Слика 5: *Компоненте на модулу*

ESP8266-01 има уграђену Flash меморију, RAM (енгл. Random Access Memory) и EEPROM (енгл. Electricaly Erasable Programmable Read-Only Memory), што омогућава директно програмирање из Arduina без потребе за додатним микроуправљачем. Подржани су WiFi протоколи 802.11b/g/n, има 1 MB Flash меморије, а радни напон је у распону од 0.8 до 3.6 V. Састоји се од 32-битног микроуправљача, Флеш меморије, антене и 8 пинова.[20]

Овај модул поседује довољно моћне способности за обраду и складиштење података на плочи, што омогућава да се интегрише са сензорма и другим специфичним уређајима преко GPIO пинова.

Његов висок степен интеграције на чипу омогућава минимално спољашње коло, укључујући и предњи модул, дизајниран да заузима минималну површину штампане плоче. Након успешног развоја модула ESP8266-01, Ai-Thinker је на основу њега развио неколико нових модула са главном разликом у броју GPIO чипова, брзини. То су на пример ESP-WROOM, NodeMCU, WeMOS, SparkFun итд. [8]

VCC	Напонски пин за добијање струје
GND	Уземљење
TX	Пин за пренос серијских података на друге уређаје
RX	Пин за пренос података са других уређаја
CH_PD	Пин за активацију чипа. Конектује се на напон.
GPIO0	Има 2 функције. Режим за програмирање када је на GND
GPIO2	Повезивање са другим уређајима(сензорима)
RST	Пин за ресетовање модула.

Табела 3: *Функције пинова модула ESP8*

3.2.1. Штедљиви режим рада

Обзиром да углавном не можемо да бирамо компоненте микроконтролера, нисмо у могућности да оптимизујемо хардвер. То се може надокнадити софтвером који ће да успава хардвер и тако уштеди мноштво енергије. Док су у фази сна, уређаји вуку знатно мање енргије него док раде.

Постоји четири врсте сна (sleep mode) за модул ESP8266, а то су:

- No-sleep;
- Modem-sleep;
- Light-sleep;
- Deep-sleep.

Item		Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi		OFF	OFF	OFF
System clock		ON	OFF	OFF
RTC		ON	ON	ON
CPU		ON	Pending	OFF
Substrate current		15 mA	0.4 mA	~ 20 μA
Average current	DTIM = 1	16.2 mA	1.8 mA	-
	DTIM = 3	15.4 mA	0.9 mA	
	DTIM = 10	15.2 mA	0.55 mA	

Слика 6: *Разлике у sleep mode-овима*

No-sleep

Уређај који нема у себи позив фунцкије за sleep mode ће константно трошити енергију док је на напајању и самим тим је и неефикасан.

Modem-sleep

Док је у фази спавања, ESP8266 може да остане конектован на интернет и да прима информације са телефона или сервера. Он искључује модем између интервала DTIM(Delivery Traffic Indication Message) Beacon-a, који поставља рутер. Modem sleep се користи код уређаја којима процесор мора да ради стално.

Light-sleep

Light-sleep има исту функцију као и modem-sleep, али поред тога, он искључује системски сат и зауставља процесор.

Deep-sleep

Код deep sleep-a све је искључено осим RTC-a (реалног времена), помоћу којег рачунар држи време. Обзиром да су све функције искључене, овај вид спавања представља најефикаснији вид чувања енергије.

Modem и Light sleep су погодни када је потребна мала уштеда енергије, а функционисање ESP-а мора да буде константно, док се deep sleep користи када је потребна озбиљна уштеда енергије.

SDK обезбеђује интерфејсе за омогућавање modem и light sleep-а, а доњи слој система одлучује када се креће у стање спавања. Режимом дубоког спавања управљају корисници. Корисници могу позвати функцију да одмах активирају deep-sleep.

Са овим модом апликациона структура прати кораке:

1. Читање са сензора (У нашем случају MAX30100 и DHT11)
2. Спавање на неколико милисекунди
3. Понављање акције

Потребно је нагласити да се дужина спавања изражава у милисекундама.

Да би се омогућио deep sleep, потребно је прво повезати пинове RST и D0. RST пин се држи на High сигналу док ESP8266 ради. Међутим, када RST пин прими слаб сигнал, микроконтролер се рестартује. Када се уређај налази у дубоком спавању, послаће слаб сигнал D0 када је тајмер за спавање укључен. Потребно је да буде повезан D0 са RST-ом да би се уређај пробудио када је спавање завршено.[4]

На следећој слици се може видети функција преко које се уређај ставља у deep-sleep стање на 20 секунди.

```
void setup() {
                Serial.begin(115200);
                Serial.setTimeout(2000);
                while(!Serial) { }
                Serial.println("I'm awake.");
                Serial.println("Going into deep sleep for 20 seconds");
                ESP.deepSleep(20e6);
              }
              void loop() {
              }
```

Код 1: *Функција за покретање deep sleep-а*

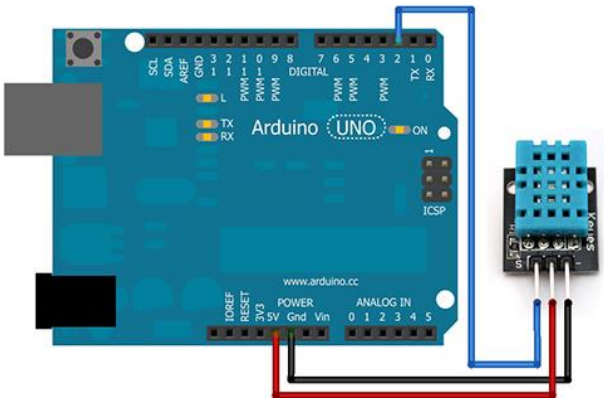
3.3. Температурни сензор DHT-11

DHT-11 је сензор за мерење температуре и влажности ваздуха са једножичним дигиталним интерфејсом. Сензор је фабрички калибрисан и нису потребне додатне компоненте за његову примену и самим тим се одмах након повезивања и укључивањем одговарајуће библиотеке може употребити за мерења. Састоји се од капацитивног сензора влажности ваздуха, термистора за мерење температуре и електронике за

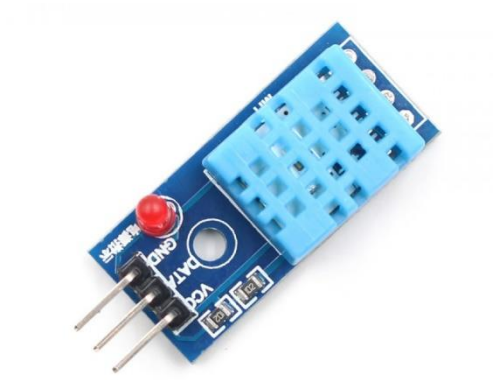
комуникацију са микроконтролером. Садржи 3 пина преко којих се конектује на извор напајања и неке друге уређаје/сензоре уколико је потребно.

Техничке карактеристике:

- Напајање: 5V
- Потрошња струје: 2,5mA
- Опсег температуре: 0-50C +- 2C
- Опсег влажности:20-80% RH прецизност +-5%RH



Слика 8: *Повезивање сензора DHT11 и Ардуина*



Слика 7: *Сензор DHT11*

GND пин се повезује на уземљење, Data пин на дигитални пин, а VCC на напон.

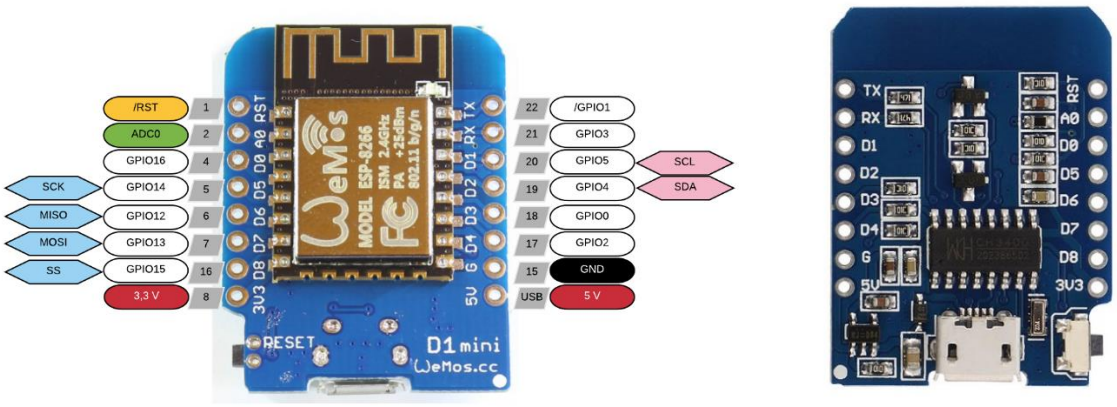
Да би сензор функционисао како треба, поред тачно имплементираног кода, потребно је пре свега инсталирати библиотеку за сензор DHT-11 и убацити је у код. [9]

3.4. 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, веома сличан NodeMCU модулу.

Највећа разлика је у величини. Wemos D1 је мањи, и има новију верзију Wifi 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.



Слика 9: *Wemos D1 mini са интегрисаним ESP8266-12f*

Спецификације:

- Омогућава 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-a)
- 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)

Wemos D1 поседује прекидач напајања који омогућава напајање плоче и до 24 V. 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 3.2 V. [10]

3.5. Сензор MAX30100

Сензор MAX30100 садржи 2 LED диоде, црвену и инфраред (IR), фотодетектор и звучни сигнал уз помоћ којег се детектује пулсна оксиметрија и срчани откуцаји. Прикупљени подаци од IR и црвеног светла се складиште у FIFO баферима до 64 бајта. Омогућава два операциона стања. Стање откуцаја срца и стање откуцаја срца и ниво кисеоника у крви. За прво стање где се детектује откуцај срца укључена је само IR LED, док је у дуплом моду где се рачуна и откуцај и ниво кисеоника укључују и IR и црвена LED.

Слика 10: *Сензор за мерење кисеоника у крви MAX30100*

Такође, плочица садржи интегрисан филтер од 60Hz. Иако може да препозна фреквенцију(буку) на електричном воду, она и даље не може да препозна буку из спољашности и флуктуације. Од LED диода, преко прста се преноси црвено и инфраред светло, а фотодетектор интегрисан у чипу детектује апсорбцију светлости од две одвојене таласне дужине.

MAX30100 је I2C уређај, стога кроз код је потребно укључити Wire библиотеку за повезивање. Физички, MAX30100 се са управљачким јединицама повезује преко специјалних пинова који су способни да читају податке преко SCL и SDA пинова, што на Ардуину представљају A4 и A5 пинови.

SDA служи за пренос података, а SCL за тактни сигнал. Пин GND(уземљење) се конектује на GND, а VCC (напон) на 5 V.

Оксигенисана крв апсорбује више инфрацрвеног светла и пропушта више црвене светлости док дезоксигенисана крв апсорбује црвену светлост и пропушта више инфрацрвене светлости. То представља главну функционалност сензора MAX30100-он детектује ниво апсорбције за оба светлосна извора и складишти их у бафер преко кога се може читати преко I2C. [6]

Назив пина	**Функција пина**	**Конекција**
VCC	Напајање	5V
GND	Уземљење	GND
SCL	Серијални такт	SCL
SDA	За пренос података	SDA

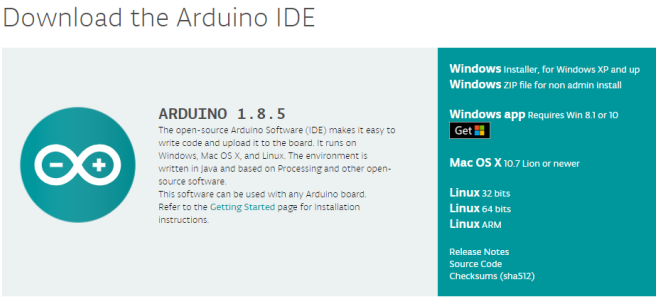
Табела 4: *Пинови сензора MAX30100*

3.6. Развојно окружење Arduino

Open source Arduino software (IDE) олакшава писање и аплоадовање 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 на било ком програмском језику. Такође, постоје и многобројне бесплатне библиотеке које се инсталирају како би писање софтвера било што једноставније.

3.6.1. Инсталација

Arduino софтвер се инсталира тако што се покрене инсталација за одговарајући оперативни систем која се налази на оригиналном сајту Arduina у одељку SOFTWARE.



Слика 11: *Инсталација Arduino IDE софтвера*

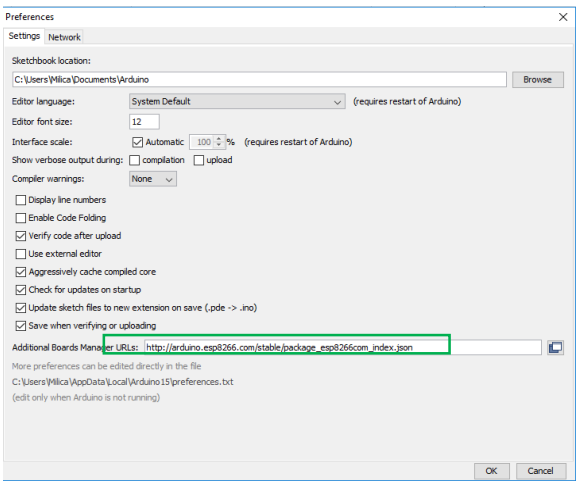
Након тога се врши оптимизација окружења у зависности од потребе пројекта који се преко овог окружења остварује.

3.6.2. Подешавање радног окружења

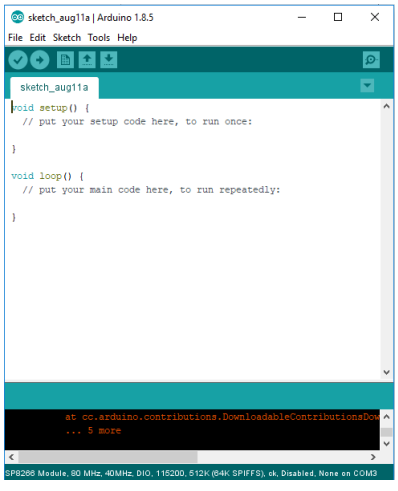
Да би се омогућило правилно функционисање програма, потребно је прилагодити одређене компоненте софтвера са потребама пројекта који се извршава. Прво што треба да се уради јесте прављење новог пројекта преко опције **File-New** (**Слика 12**) и снимање фајла преко опције **File-Save as**.

Затим се преко опција **File-Preferences-Settings** долази до прозора где се у правоугаоник **Aditional Boards Managers URL** (**Слика 13**) копира следећи линк:

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



Слика 13: *Прозор за додавање URL линка*

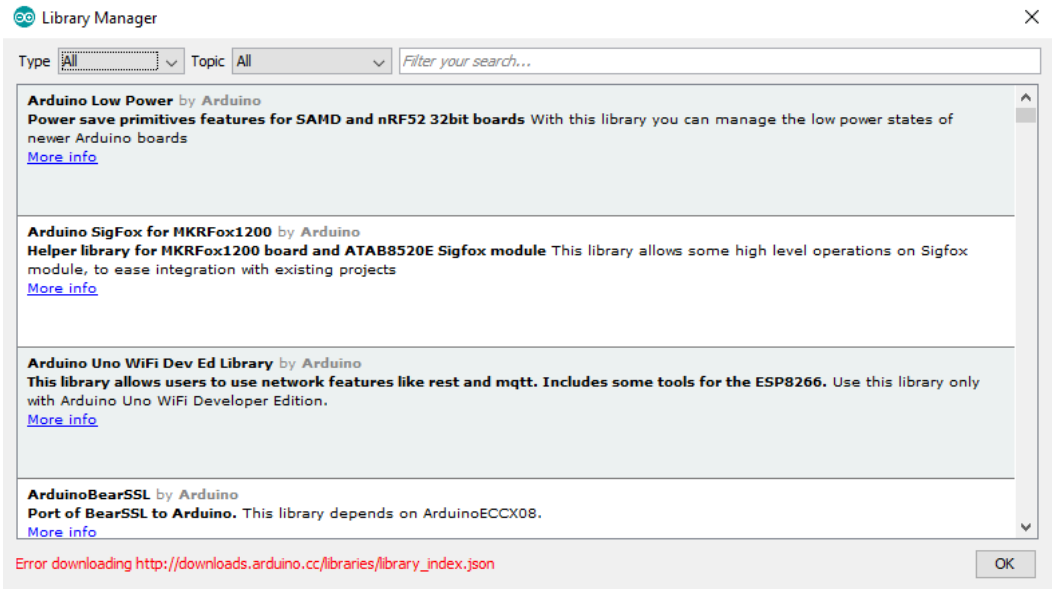


Слика 12: *Покретање новог пројекта*

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 Menager-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.

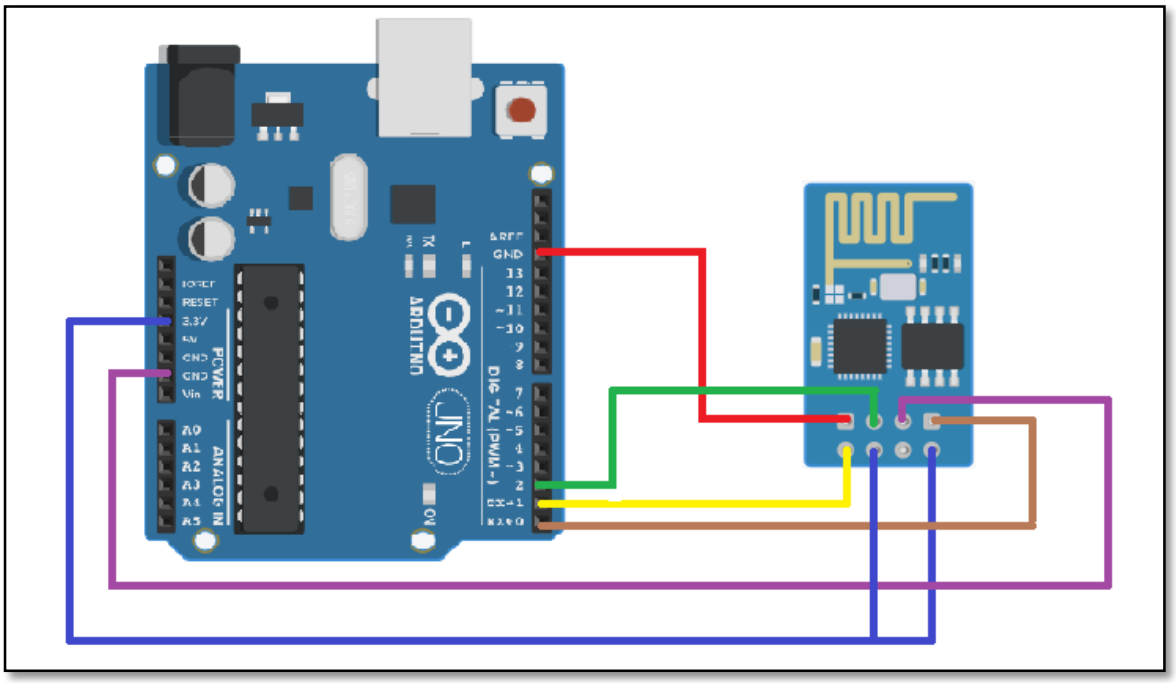
Такође, у зависности од потреба пројекта, односно од сензора који се користе, потребно је инсталирати одређене библиотеке које су бесплатне и инсталирају се тако што се преко опција **Sketch-Include library-Manage libraries** дође до прозора(**Слика 14**) и преко претраживача пронађе одговарајућа, а 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. [11]



Слика 14: *Прозор за инсталацију библиотека*

3.6.3. Флешовање ESP8266-01

Да бисмо омогућили правилну имплементацију кода на модул ESP8266-01, прво је потребно одрадити неколико битних ставки. Прва ставка је физичко повезивање, а затим и флешовање самог Wifi модула као и прилагођавање радног окружења потребама модула. На слици (**Слика *15***) се може видети шема повезивања модула са платформом Ардуино Уно. Модул ESP8266-01 се састоји од 8 пинова који се повезују са одговарајућим улазима/излазима ардуина.



Слика 15: *Повезивање модула ESP8266-01 са Ардуином*

Povezivanje je sledeće:
VCC – – > 3.3V
GND – – > GND
CH_PD – – > 3.3V
RST – – > GND za resetovanje
GPIO0 – – > GND
TX – – > TX

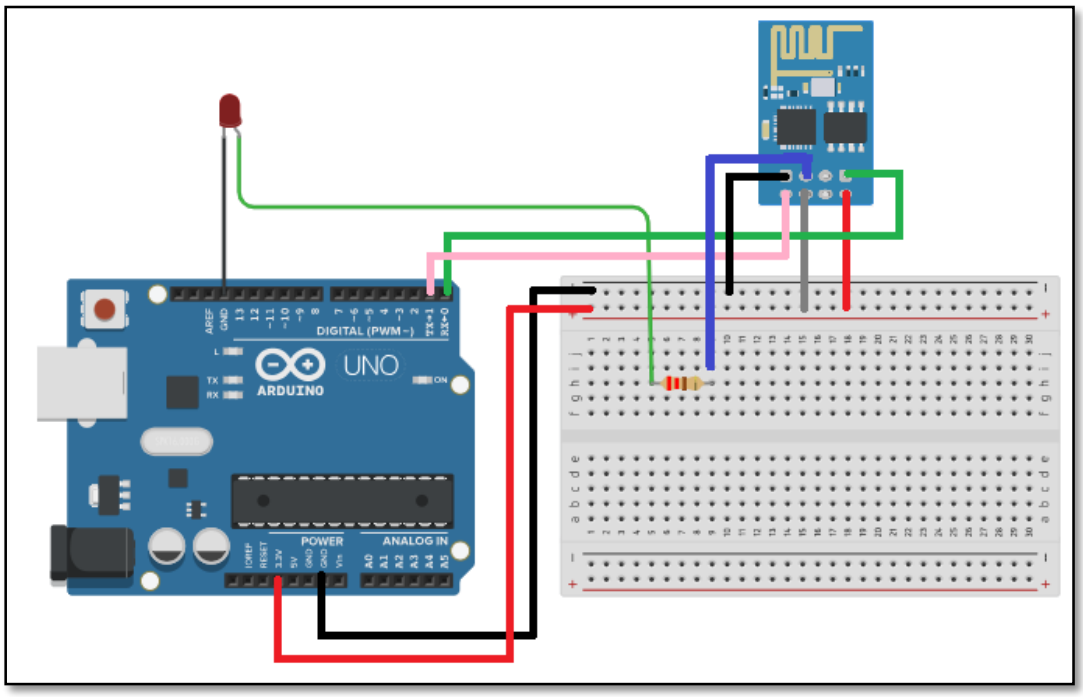
Приказано је флешовање уз пример за блинковање лед лампица јер се тако може видети конекција између лампице и модула преко пина GPIO2.

Након физичког повезивања компоненти потребно је прилагодити радно окружење. Као тестни код, односно код за флешовање ћемо користити најједноставнији код за блинковање лед лампица на одређен временски период, који се и сам налази у листи примера програма Ардуина. [14]

Напомена: ***Потребно је извадити интегрално коло из Ардуина***.

Поред повезаног Wifi модула и Ардуина, да бисмо флешовали модул преко блинка, потребно је LED лампицу повезати са осталим компонетама.

На слици 16 је представљен шематски приказ повезивања.



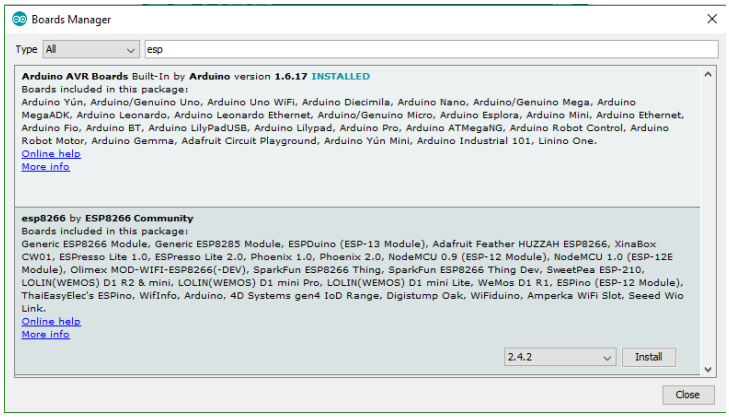
Слика 16: *Повезивање LED диоде са модулом ESP8266-01*

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 17**), 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.

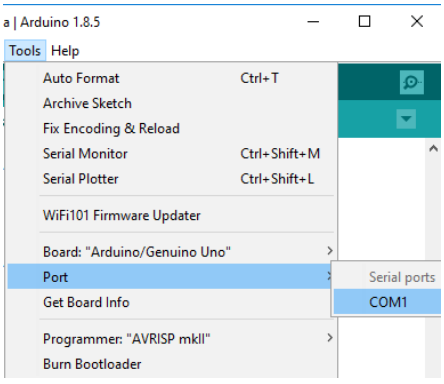
Koraci:

1. **Покретање новог пројекта**(**Слика 12**)
2. **Подешавање порта, платформе, модула** (**Слика 17**)(**Слика 18**)
3. **Инсталација и укључивање библиотеке за модул ESP8266**(**Слика 21**)
4. **Имплементација кода**(**Код 2**)

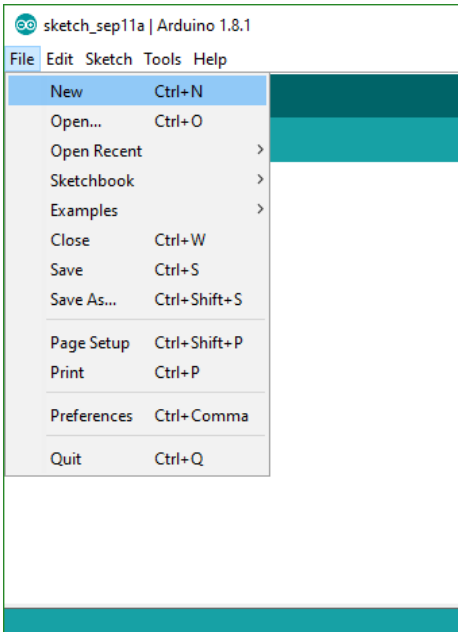
Потребно је да се одабере одговарајући порт (Провера се врши у **Device Menager-u**), који се налази у опцијама **Tools-Port.**(**Слика 18**)



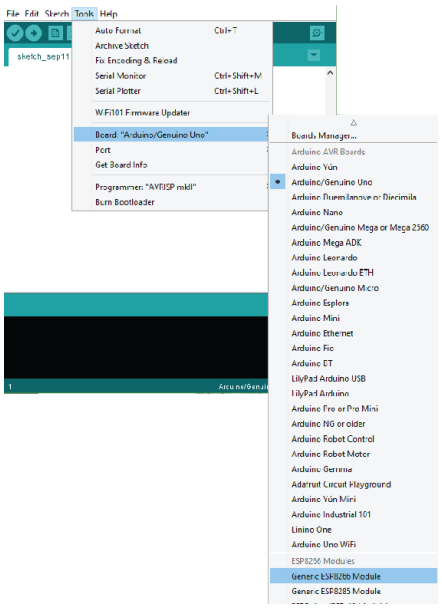
Слика 18: *Инсталирање платформе ESP8266*



Слика 17: *Подешавање порта*

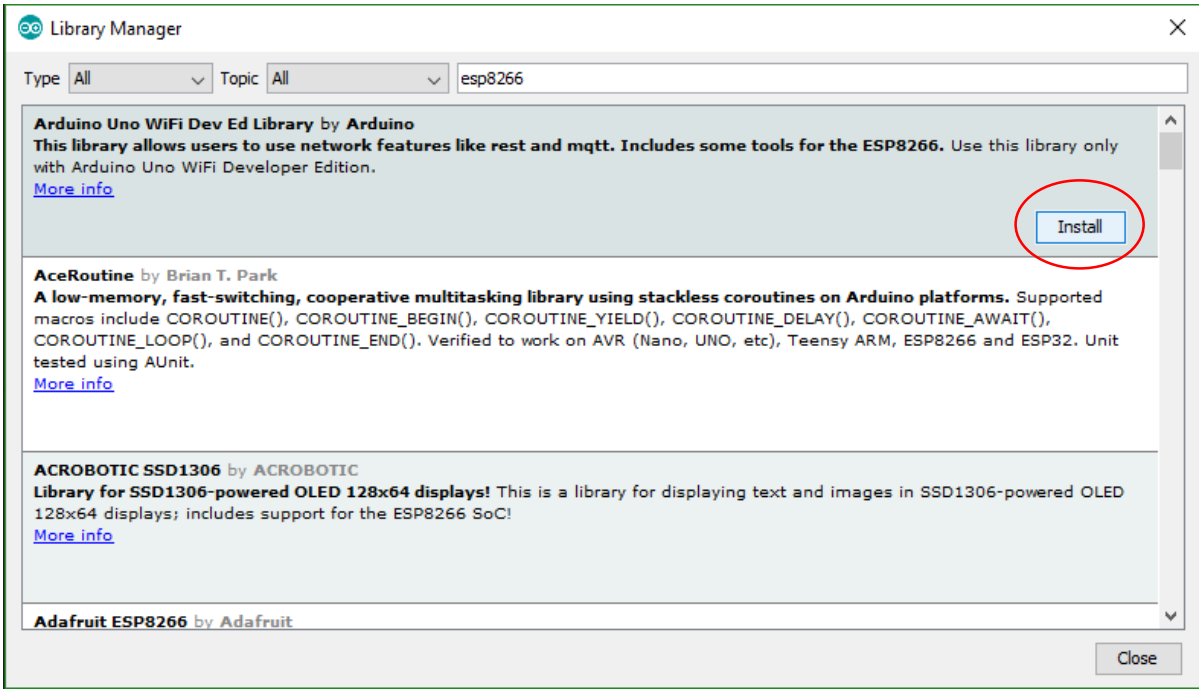


Слика 20: *Прављење новог пројекта*



Слика 19: *Укључивање платформе ESP8266*

Instalacija biblioteke se vrši preko opcija **Sketch-Include library-Manage libraries** dođe do prozora (**Слика 21**) i preko pretraživača pronađe biblioteka za ESP8266. Nakon toga potrebno je tu istu biblioteku uključiti u projekat tako što se preko opcija **Sketch-Include library** iz liste padajućeg menija izabere instalirana biblioteka modula ESP8266.



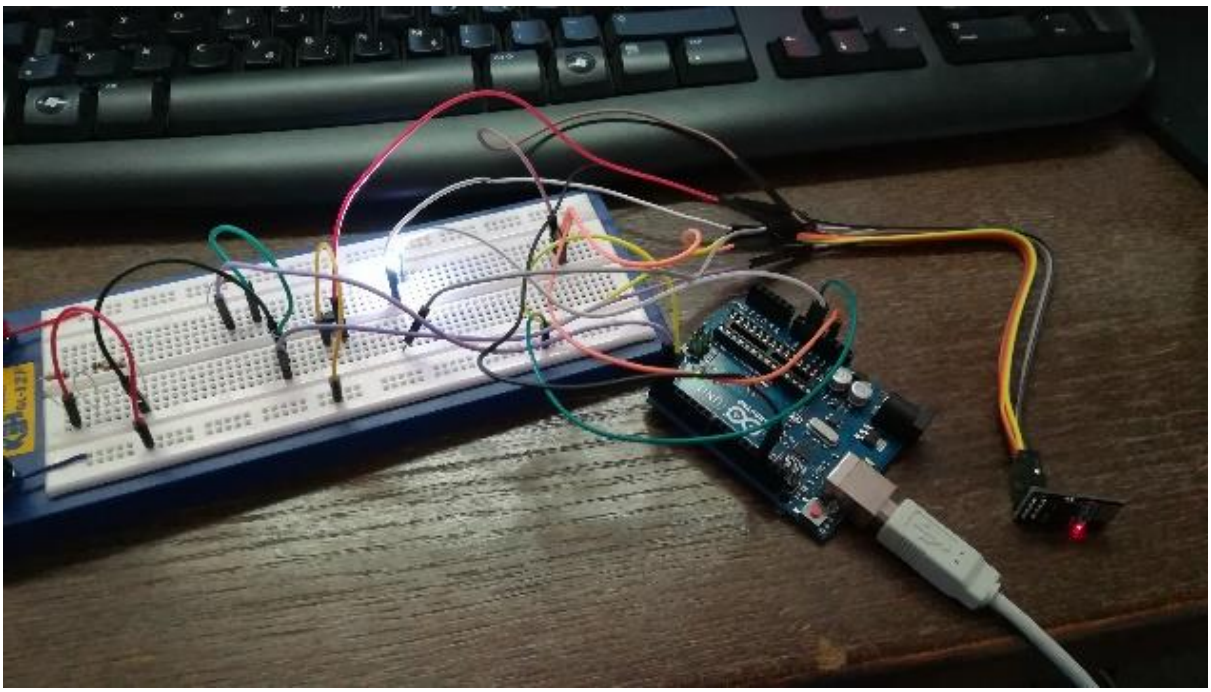
Слика 21: *Инсталација библиотеке за модул ESP8266*

Код за блинкање лампице је следећи:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Код 2: *Имплементација кода за блинкање LED диоде*

На слици (**Слика 22**) се може видети успешно имплементиран програм за блинкање лампице.



Слика 22: *Успешно блинкање LED диоде*

3.7. ANDROID

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.

Слика 23: *Логотип Androida*

Поред тога што је отвореног кода, Андроид се лако може надограђивати, и програмери не морају да имају лиценцу за њега. Са свим алатима неопходним за прављење апликација, Андроид представља комплетан развојни оперативни систем. Изворни код Андроид-а програмерима је доступан и лако се може разумети како функционише.

3.7.1. Структура Androida

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. [12]

3.7.2. ANDROID STUDIO

Android Studio је званично интегрисано развојно окружење (IDE) за развој Android 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 Android уређаје. Претходно коришћено радно окружење јесте Eclipse.У овом раду је коришћен Android Studio верзије 3.1.4. Почетни прозор Андроид апликације изгледа као на слици 24.

Слика 24: *Почетни прозор Андроид Студија*

3.7.3. Основне компоненте

Андроид Студио нам омогућава креирање апликације за различите типове уређаја. Свака Андроид апликација састоји се од разних функционалности. Као примере функционалности можемо навести уношење и исправљање белешке, репродуковање музичке датотеке, активирање аларма или уношење новог телефонског контакта. Те функционалности могу се расподелити на четири Андроидове компоненте, приказане у табели, а свакој одговара различита основна класа у језику Јава.

Функционалност	Основна Јава класа	Примери
Операција коју корисник обавља	Activity	Уношење белешке
Позадински процес	Service	Репродуковање музике
Примање порука	BroadcastReceiver	Активирање аларма
Уношење и учитавање података	ContentProvider	Отварање новог телефонског контакта

Табела 5: *Пример функционалности Андроидових компоненти*

Основне компоненте представљају градивне елементе сваке Андроид апликације. Сваки од ових елеменате има своју специфичну улогу у опису понашања апликације. Све компоненте једне апликације су дефинисане у њеном AndroidManifest.XML фајлу. На Андроид платформи постоје четри различитих врста компоненти:

Активности (енг. Activity)- Компонента Activity представља појединачни екран преко којег корисници комуницирају са апликацијом. Једна апликација може имати више активности и све оне се међусобно независне. Апликација има своју "главну" активност која се покреће када корисник покрене апликацију. Једна активност може покренути неку другу активност или пак неку другу компоненту,на пример, сервис. За сваку активност постоји животни циклус који дефинише како се она креира и како се она уништава. Редослед позивања метода, које дефинишу животни циклусактивности, је регулисан од стране Андроид ОС.

Сервиси (**енг. Service**) – За разлику од активности које су везане за неки екран, сервиси представљају компоненте које се извршавају у позадини. Сервиси су намењени операцијама које се дуго извршавају. Све компоненте могу да покрену неки сервис, и он ће наставити да „живи" и када корисник напусти апликацију. Сервис се може јавити у два облика, може бити покренут или везан. Покренут сервис се покреће методом startService() која се позива из неке друге компоненте. Стартован сервис постоји у позадини чак иако је компонента која га је покренула уништена. Оваква врста сервиса се користи за операције као што су скидање неког садржаја са интернета без враћања резултата компоненти која га је позвала. Други облик сервиса је везани сервис и клијент се везује за сервис методом bindService(). Везан сервис обезбеђује компонентама да међусобно комуницирају. Трајање сервиса, покренутог на овај начин, директно зависи од животног века компоненти са којима је везан. Уколико је сервис везан са више компоненти, сервис ће „живети" све док последња компонента са којом је везан не буде уништена. На Андроид ОС постоје две класе сервиса, то су Service и IntentService. Service представља основну класу сервиса и може да утиче на перформансе апликације уколико се у њему извршавају дуге операције. IntentService је сервис који се извршава ван главне нити и он је предвиђен за извршавање операција у позадини. [17]

Пријемници (енг. Broadcast Receiver) – Основна намена ове компоненте је да прихвата Intent објекте који се шаљу широм система послате методом sendBroadcast(). Пријемници могу бити регистровани динамички у осталим компонентама или статички у

AndroidManifest.xml фајлу. Приликом регистровања пријемника дефинише се и IntentFilter објекат како би он добијао само одређене Intent објекте.

Провајдери садржаја (енг. Content provider) – Провајдери садржаја служе за приступ централном складишту података. Примарна намена ове компоненте је омогућавање дељење података између апликација/процеса. Да би подаци неке апликације били доступни некој другој, потребно је да та апликација дефинише интерфејс према својим подацима. Интерфејс се дефинише помоћу ContetnProvider класе. Клијентска апликација, која користи те податке, мора да инстанцира објекат класе ContetnProvider који ће садржати исте методе као и ContetnProvider објекат апликације чијим подацима желимо да приступимо. ContetnResolver методе обезбеђују „КРАО" (креирај, поврати, ажурирај и обриши) операције над дељеним подацима.

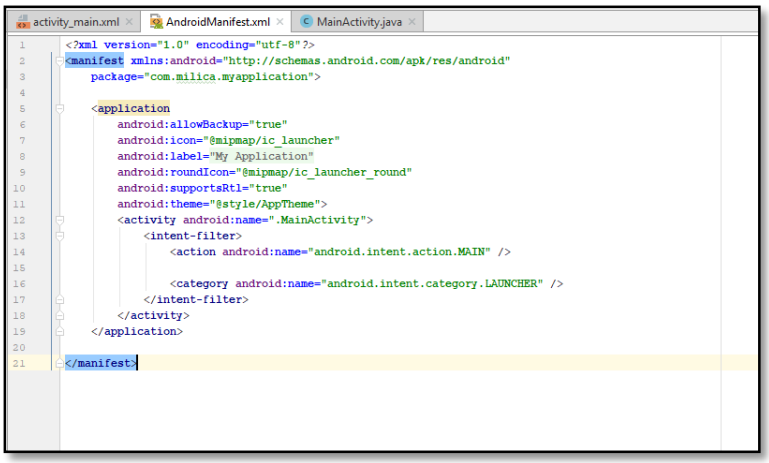
Intetnt механизам представља најабстрктинији ниво IPC(комуникација између процеса) на Андроид ОС. Intent представља поруку којом се може затражити нека акција од основних Андроид компоненти које имају филтер за ту поруку. Интент механизам је у потпуности независтан од имплементације апликације. На Андроид оперативном систему никад не кажемо да желимо да пошаљемо мејл неком одређеном апликацијом, уместо тога ми само кажемо да желимо да пошаљемо мејл. Андроид ОС ће се сам побринути за то, јер он већ унапред зна које апликације служе за слање мејла на основу филтера за Intent објекте који су дефинисани у AndroidManifest.xml фајлу тих апликација. Интенти служе за асинхрону комуникацију између компоненти и због своје једноставности су најчешће коришћени облик IPC-а. Интенти се састоје из два дела акције и података. Под апстракним се мисли на то да компоненте, које треба да изврше акцију, не морају бити дефинисане у Интент објекту. Интент објекти се могу замислити као поруке које се шаљу широм оперативног система како би извршиле одређену акцију на некој копоненти која има одговарајући филтер за ту акцију. Кад се пошаље интент, Андроид ОС проналази све компоненте које садрже филтер за ту акцију. Уколико постоји више компоненти са тим филтером, Андроид ОС препушта кориснику да изабере жељену компоненту. Постоје две врсте Интент објеката експлицитни и имплицитни. Експлицитни гађају одређену компоненту по имену и ограничен је само на компоненте унутар једне апликације. С обзиром на то да се компоненте једне апликације могу налазити на различитим процесима, и експлицитни интенти имају своју улогу у IPC-у. Са друге стране имплицитни, покушавају да пронађу компоненте по акцији и нису ограничени на компонете једне апликације. [15]

Сваки пројекат апликације мора имати AndroidManifest.xml file (**Слика *25***) (са тачно тим именом) који се налази у корену изворног скупа пројекта. Датотека Manifest описује пројекат оперативном систему Андроид. Да би оперативни систем могао да јој приступи, апликација мора да декларише све своје компоненте у једној XML датотеци чије је име AndroidManifest.xml. Осим тога, та датотека садржи описе овлашћења и понашања који су потребни за рад апликације. Манифест садржи дозволе које су апликацији неопходне ако је потребно да приступи заштићеним деловима система или другим апликацијама. Такође, декларише и све дозволе које друге апликације морају да имају ако желе да приступе садржају те апликације. [16]

AndroidManifest.xml фајл садржи следеће информације о апликацији:

- име пакета апликације;
- број верзије апликације који се користи (да би Андроид ОС проверио да ли постоји новија верзија апликације на Google Play-у);
- име верзије апликације је стринг вредност, која се углавном користи за приказивање броја верзије кориснику;
- android:minSdkVersion атрибут < uses-sdk > елемента одређује минималну верзију оперативног система на којој апликација може радити;
- у овом фајлу се дефинише лого апликације која ће се приказати у главном менију уређаја, као и име апликације;
- основне комоненте као и филтери за Intent објекте који их покрећу;
- дефинише се компонента која ће се прва покренути приликом стартовања апликације;
- све дозволе (енг. permission) које апликација мора да има да би несметано радила и приступала одређеним ресурсима;
- листа библиотека које апликација користи.[15]

У Android Studio развојном окружењу, када се прави апликација, датотека манифеста се сама креира, а већина битних елемената се додају у току креирања апликације.

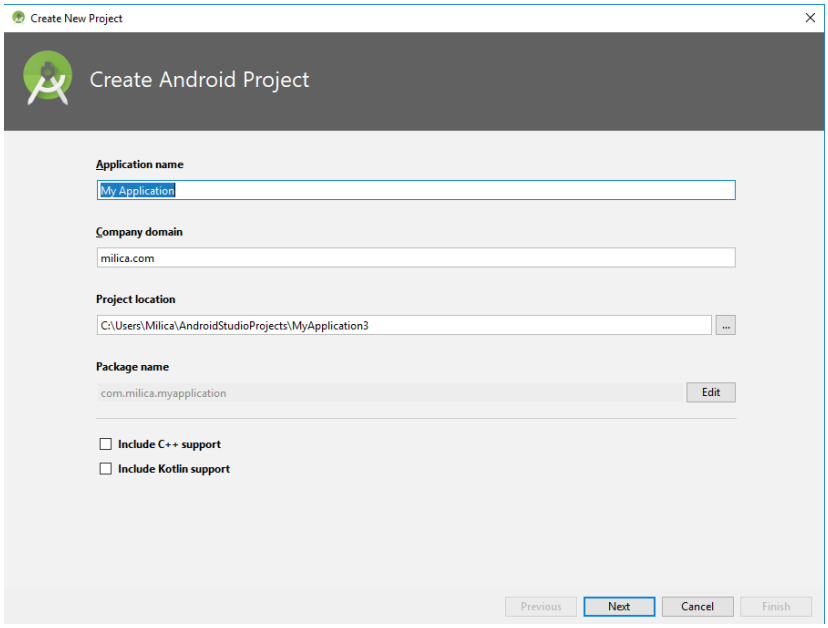


Слика 25: *Приказ датотеке AndroidManifest.xml*

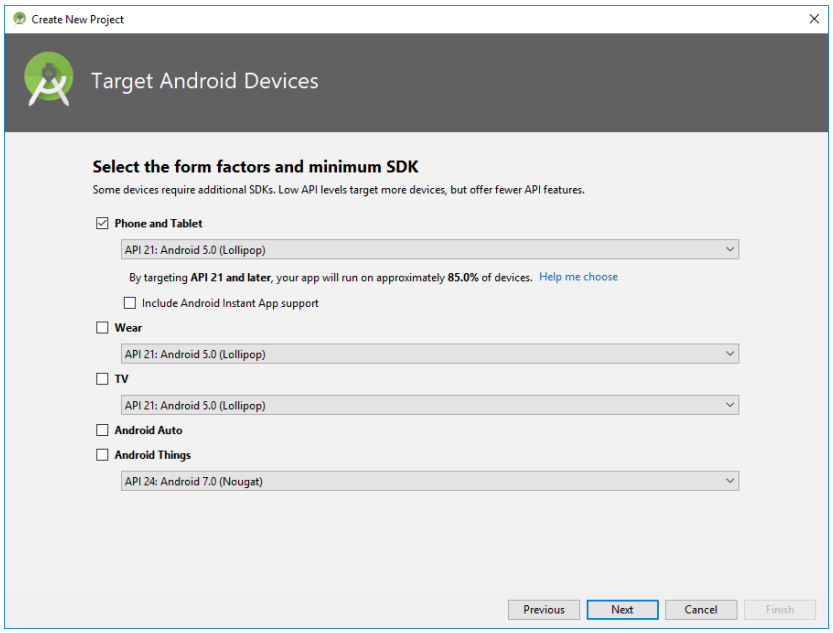
Чаробњак за прављење новог пројекта нуди избор за који уређај желимо да правимо нашу апликацију, а тиме ће нас упитати коју верзију API-а желимо да користимо. Поред ових могућности, у чаробњаку имамо и опције да изаберемо одређени Activity, и подесимо га по нашој вољи.

После креирања пројекта, потребно је приметити да је структура директоријума мало другачија, него до сада у Eclipse-у. За ово је „крив“ Gradle систем изградње, који се користи у Android Studio-у. У принципу, све функционише као и до сада, само су неки директоријуми премештени у „src/“, па је тако лакше сналажење. Ово ће пуно поједноставити рад на пројекту, у смислу да програмери неће бити збуњени које

датотеке могу да мењају, а који се мењају/пишу приликом изградње извршних датотека. Прављење пројекта се врши преко опције **file-new-new project.**(**Слика *26***)

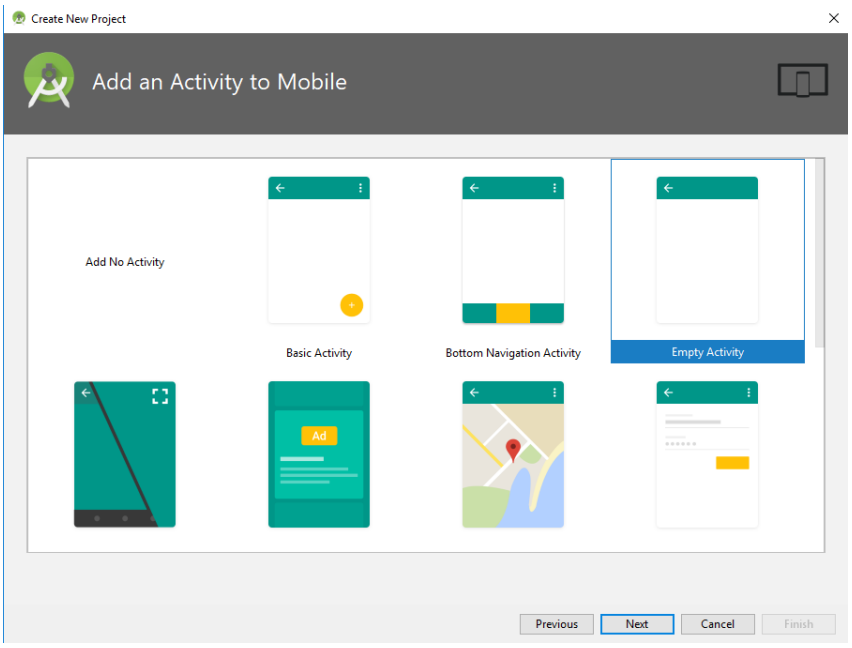


Слика 26: *Креирање новог пројекта*



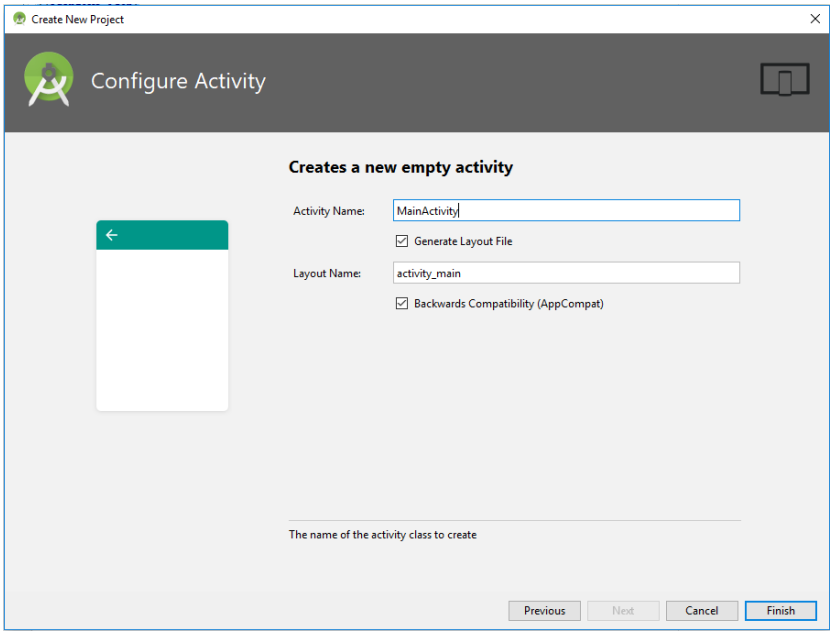
Слика 27: Подешавање API-а

API (eng. Application Programing Interface) представља програмски интерфејс и потребно је водити рачуна приликом одабира API-а, јер ако се изабере API за развој апликације са новијом верзијом од Android телефона на коме ће се извршавати апликација, апликација неће радити. Зато се препоручује одабир старије верзије API-а.



Слика 28: Одабир активитија

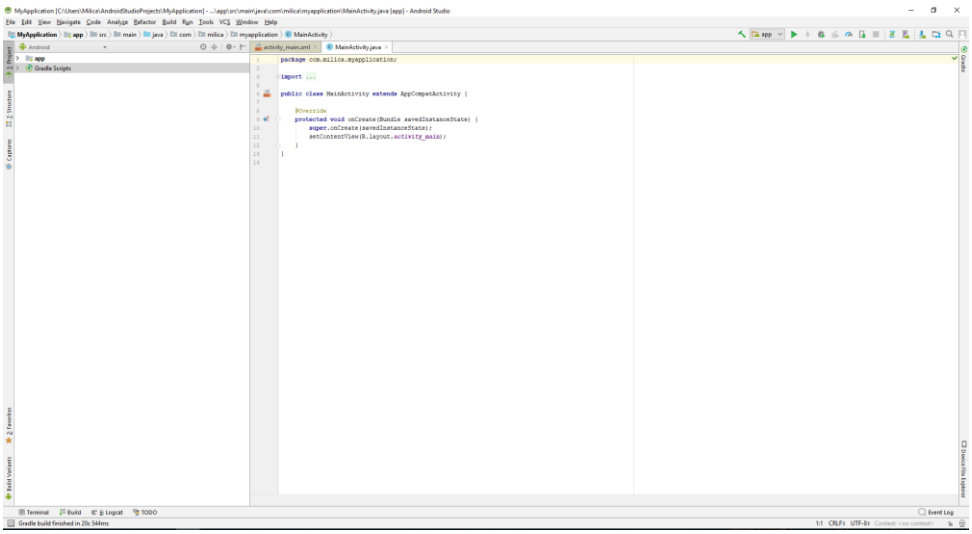
Након извршених подешавања API-а и кликом на дугме **Next,** појављује се прозор (Слика 28) са понуђеним изгледима екрана. Уколико ни један од понуђених не представља оно што желимо, одлучујемо се за Empty Activity. То је углавном случај јер се добија празан Activity, а по потреби користимо додатне опције и попуњавамо га жељеним садржајем.



Слика 29: *Прозор за доделу назива Активитија и лејаута*

У последњем прозору (**Слика *29***) који се појављује, приказују се опције за назив главног активитија и лејаута. Након доделе, кликом на дугме **Finish** креирамо нови пројекат.

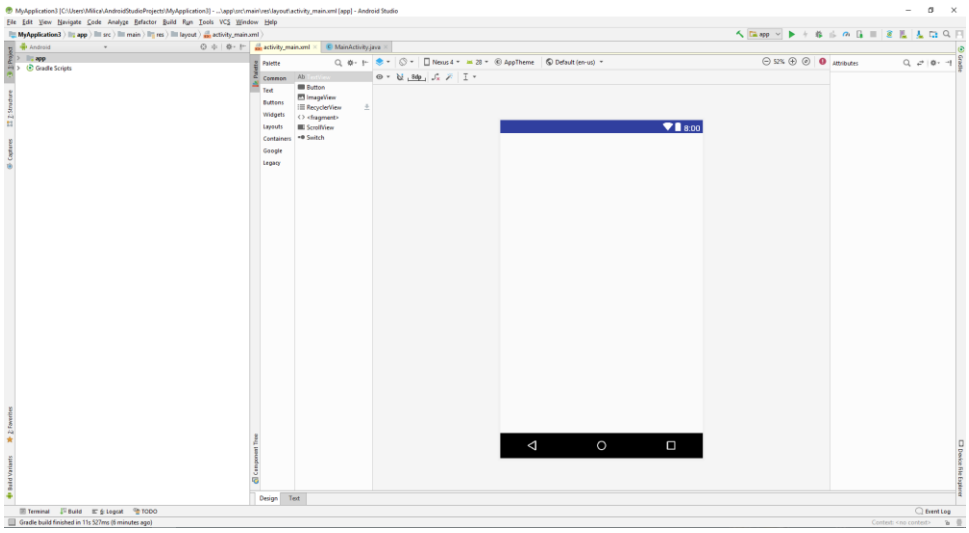
Креирањем новог пројекта добијамо изглед екрана као на слици 29.



Слика 30: *Нови пројекат*

Поред главног екрана, постоји и **Activity_main.xml** који служи за приказ екрана апликације, односно задужен је за изглед главног екрана. У доњем левом углу прозора постоје две опције у оквиру **Activity_main.xml**. Једна је **Text,** а друга је **Design** (**Слика *31*).**

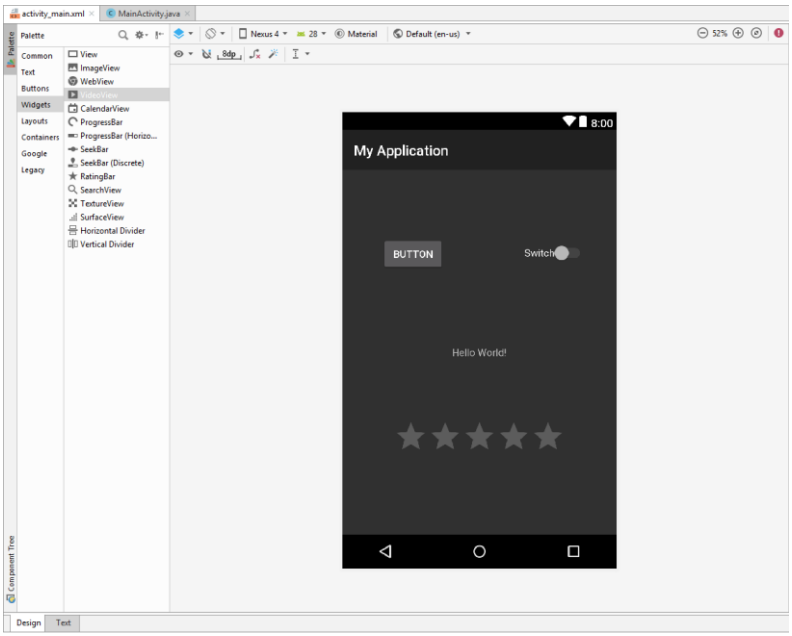
На слици 30. видимо екран који се појављује када се уђе у опцију Design. Екран је приказ празног екрана који смо одабрали у току креирања пројекта. У њега додајемо текст, дугмиће, слике и самим тим подешавамо изглед почетног екрана. Додавање се врши превлачењем жењених елемената на екран који се налазе у листи палете са леве стране.



Слика 31: *Изглед екрана Activity_main.xml*

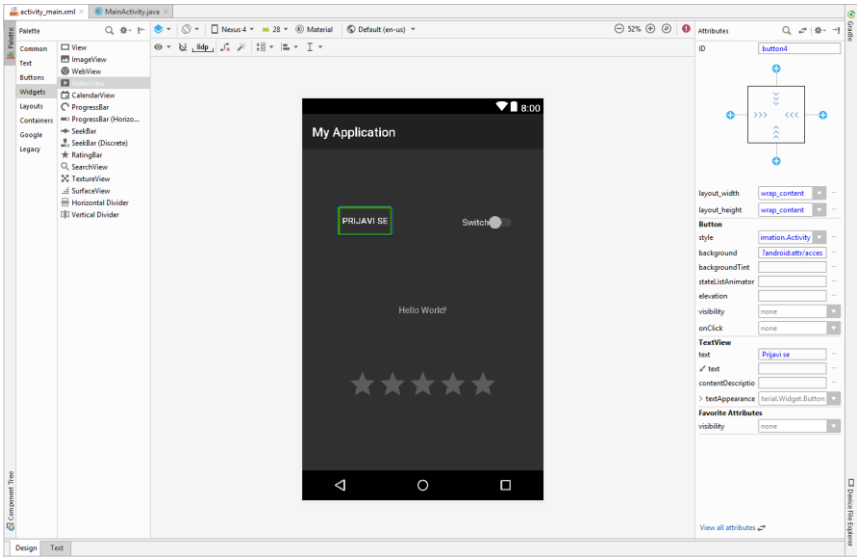
У горњем делу екрана постоје опције преко којих се може мењати тема екрана, може се изабрати телефон преко којег ће се приказивати апликација, мењати оријентацију, подешавају маргине итд.

На слици 31 можемо видети како изгледа екран коме смо променили тему и у који смо додали одређене елементе .



Слика 32: *Екран са промењеном темом и додатим елементима*

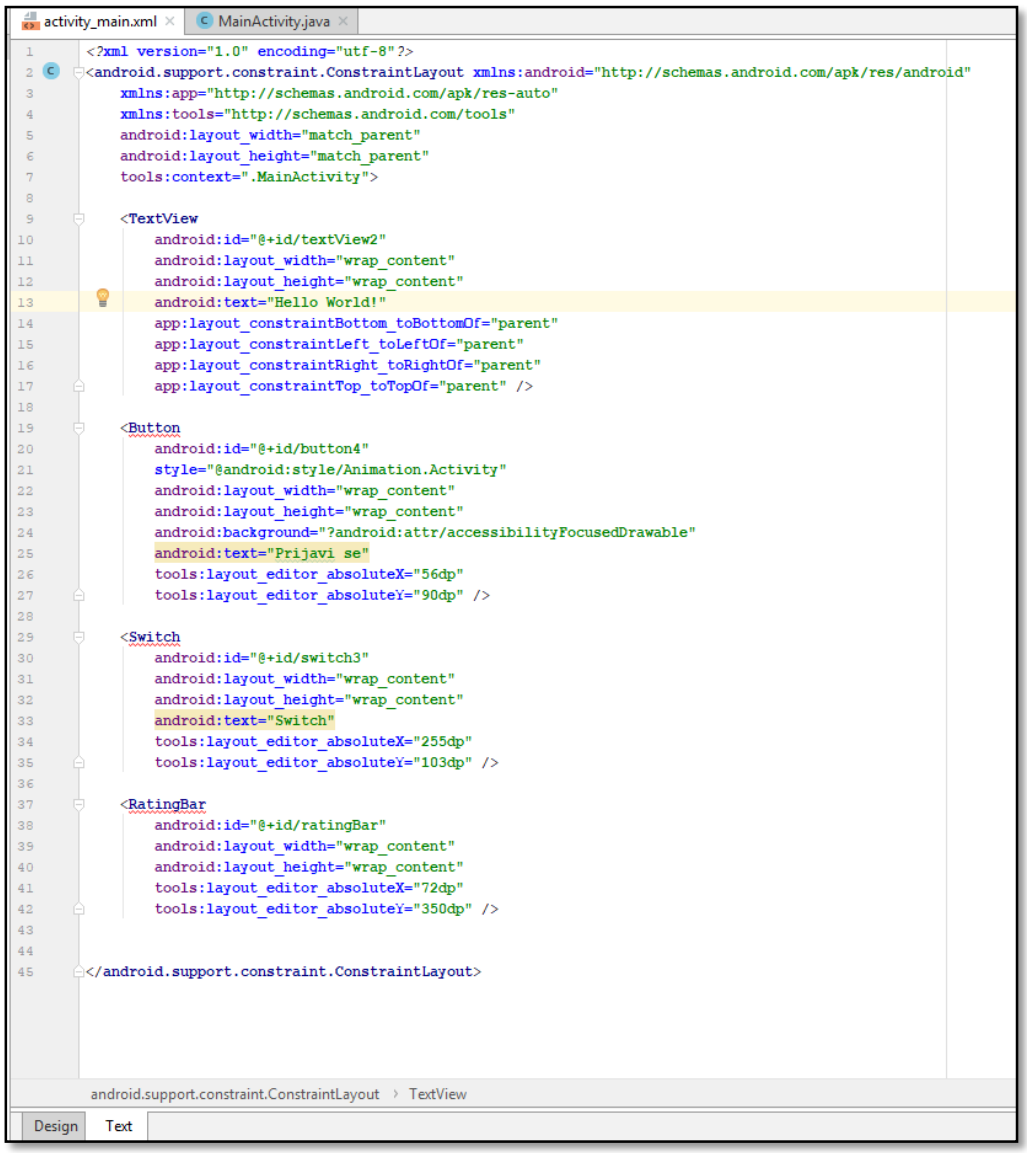
Да би се вршиле измене елемената (widgets), потребно је да се кликне на њих и у десном делу се појаве опције које нуде велики број измена и различитих подешавања за тај widget (***Слика 33***). Ми смо у примеру (***Слика 33***) променили назив дугмета из BUTTON у ПРИЈАВИ СЕ и обојили смо ивице дугмета у зелену боју.



Слика 33: *Измена widget-а*

Када се уђе у опцију **Text,** појављује се текстуални приказ layout-а. Видимо да се у тексту налазе **TextView** уз помоћ којег смо на екрану приказали **Hello world!,** ту су и

дугме **Пријави се,** switch и RatingBar. Све оне измене које смо урадили у Design прозору када кликнемо на widget, можемо урадити и овде тако што ћемо додати одређене тагове и атрибуте у овај xml фајл, као што се види у примеру на слици 33.



Слика 34: *Текстуални приказ изгледа екрана*

У овом поглављу смо приказали основне елементе Андроид Студија који су потребни за основну употребу окружења и прављење једноставних апликација. Поред основних елемената, постоје многи други који у овом раду неће бити описани, али имају примену у изради великих апликација.

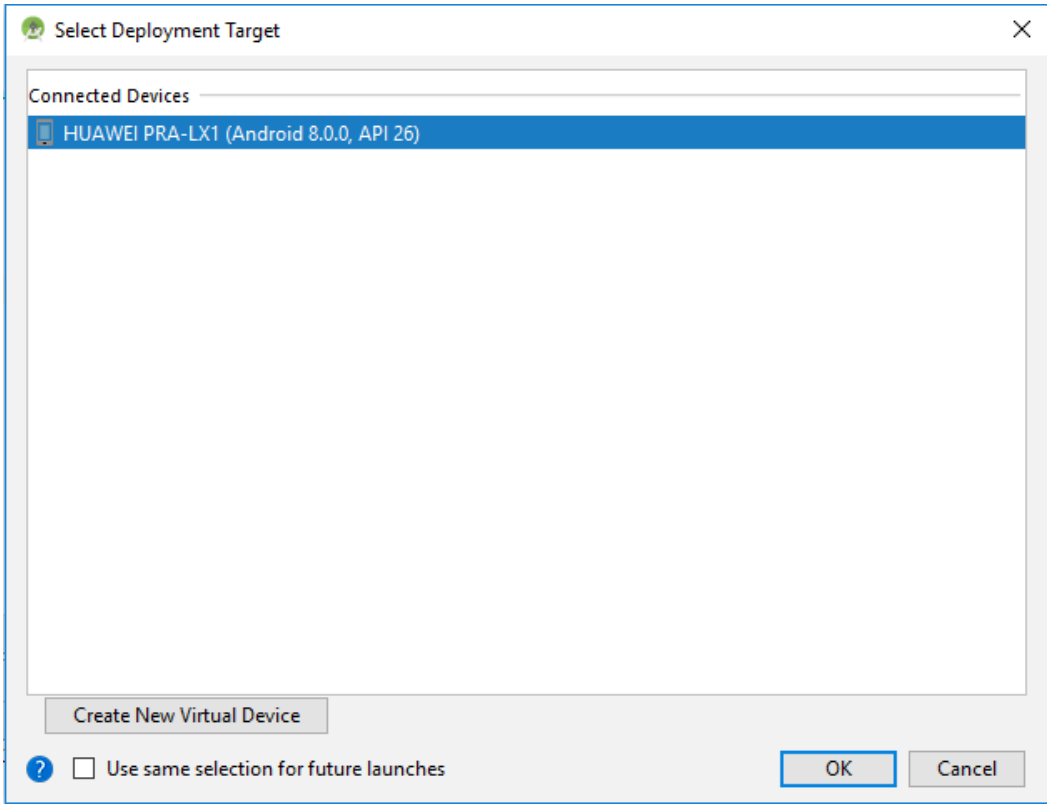
3.7.4. Повезивање са телефоном

Да бисмо Андроид развојно окружење повезали са мобилним уређајем, потребно је у самом телефону поставити систем у developer стање. То се врши преко опције **Settings-System-About phone** и неопходно је седам пута кликнути на опцију **Build number** како би уређај постао developer.

Након те поставке, преко usb кабла повезујемо мобилни уређај са рачунаром.

Повезивање се врши тако што се кликне на иконицу која се  налази у горњем десном углу развојног окружења.

Појављује се прозор(***Слика 35***). Кликом на ок повезјемо мобилни уређај и развојно окружење.



Слика 35: *Повезивање мобилног уређаја са окружењем Android Studio*

Након покретања, на екрану уређаја се појављујује све оно што смо у Android окружењу релизовали.

4. РЕАЛИЗАЦИЈА

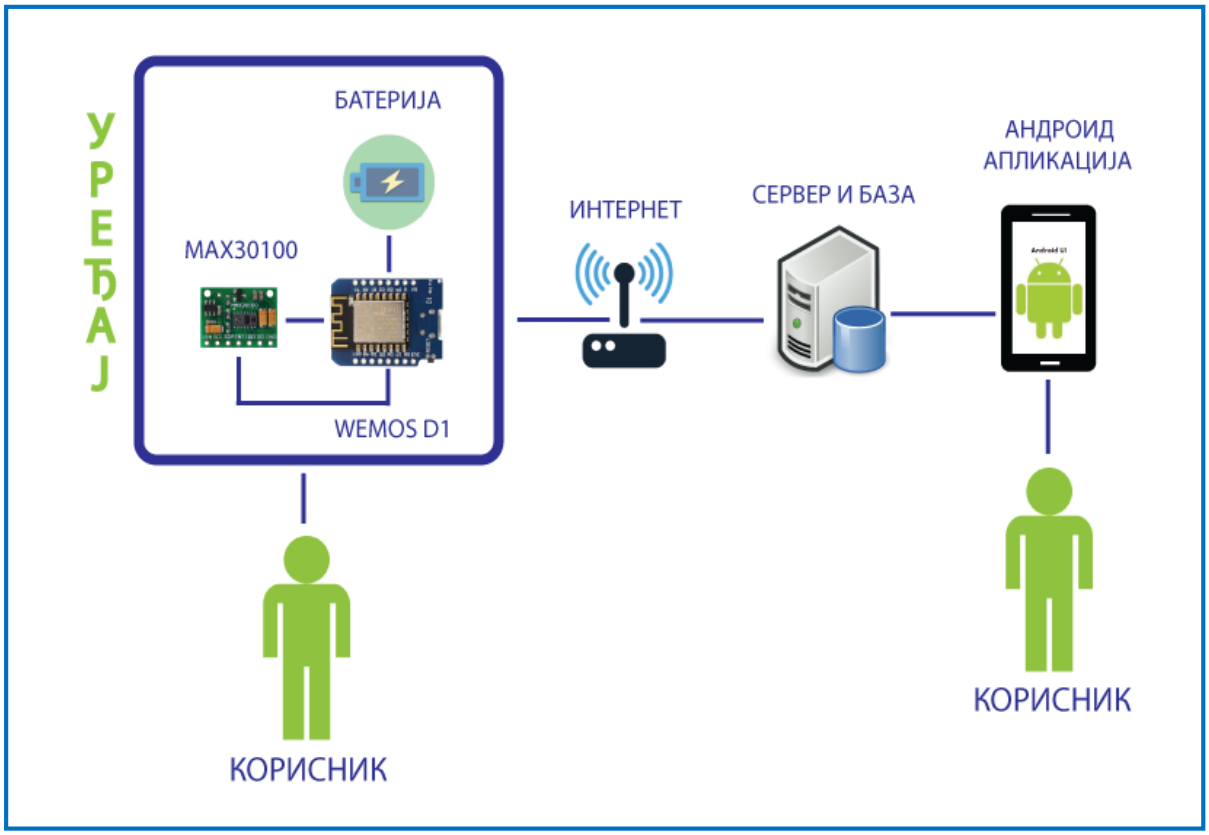
4.1. Уређај OXY

Циљ овог пројекта јесте да се направи преносни уређај што мањих димензија који ће служити да уз једноставно коришћење може да прати своје или туђе здравствено стање које је повезано са нивоом кисеоника у крви.

Преко Андроид апликације, корисник ће моћи да прати ниво кисеоника у крви, а поред тога ће бити обавештен нотификацијом сваки пут када кисеоник опадне испод границе нормале.

На слици је графички приказано повезивање целокупног система уређаја који смо назвали ***Oxy*.**

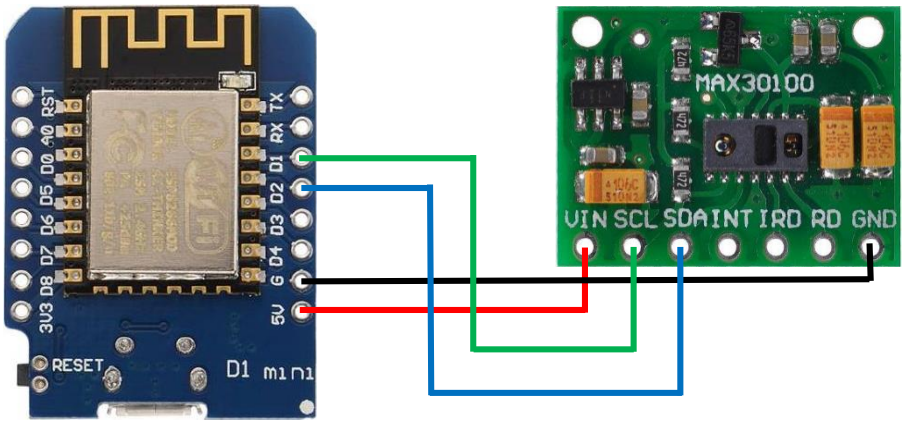
Сам уређај се састоји од 3 битна елемента, а то су: сензор за мерење кисеоника у крви MAX30100, Wemos D1 mini са интегрисаним Wifi модулом ESP8266-12f и батерија као извор напајања. Корисник уређај користи тако што га стави на кажипрст. WiFi модул преко интернета шаље податке у базу. Други корисник може да инсталира Андроид апликацију на телефону, и путем ње, у реалном времену прати стање корисника који носи уређај.



Слика 36: *Шематски приказ пројекта*

4.1.1. Конекција WEMOS d1 mini / MAX30100

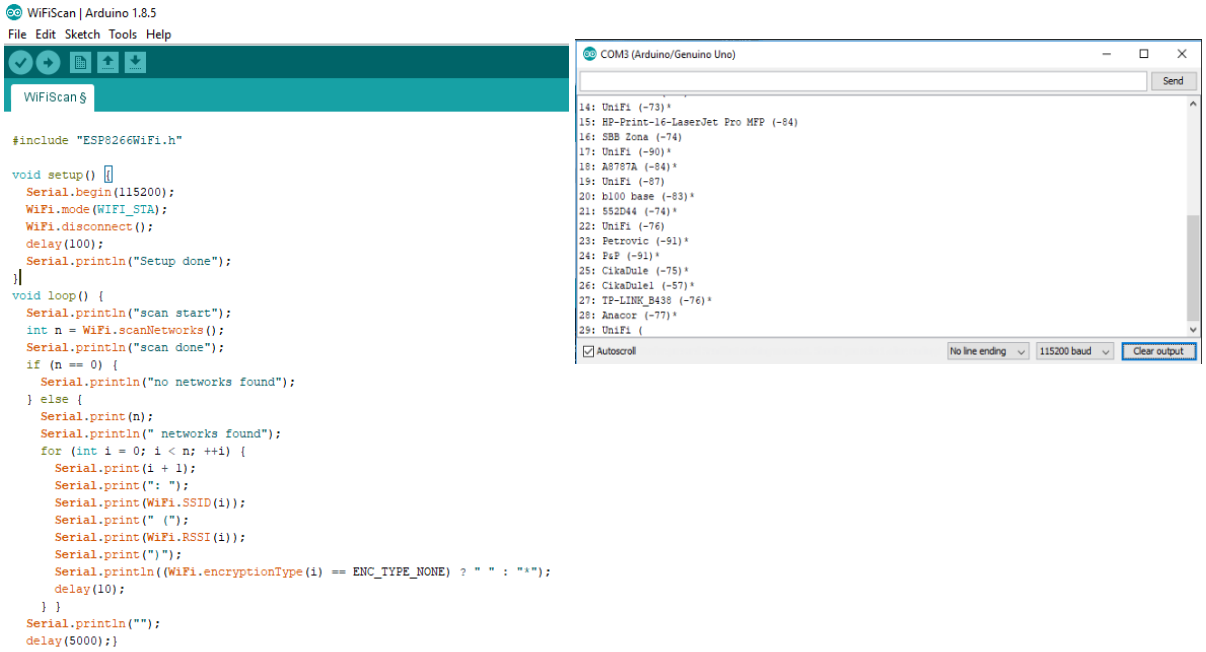
Повезивање модула Wemos D1 mini и сензора MAX30100 се врши као на слици 26.



Слика 37: *Повезивање сензора MAX30100 са Wemos D1 mini*

Након физичког повезивања сензора и модула, следи флешовање модула, односно пребацивање кода помоћу кога ESP8266-12F исписује све доступне мреже. Најпре тестирамо модул кодом који је преузет са званичног сајта Ардуина.

Пример: Слика 27



Слика 38: *Приказ кода за детектовање доступних мрежа и њихово исписивање на сериал монитору*

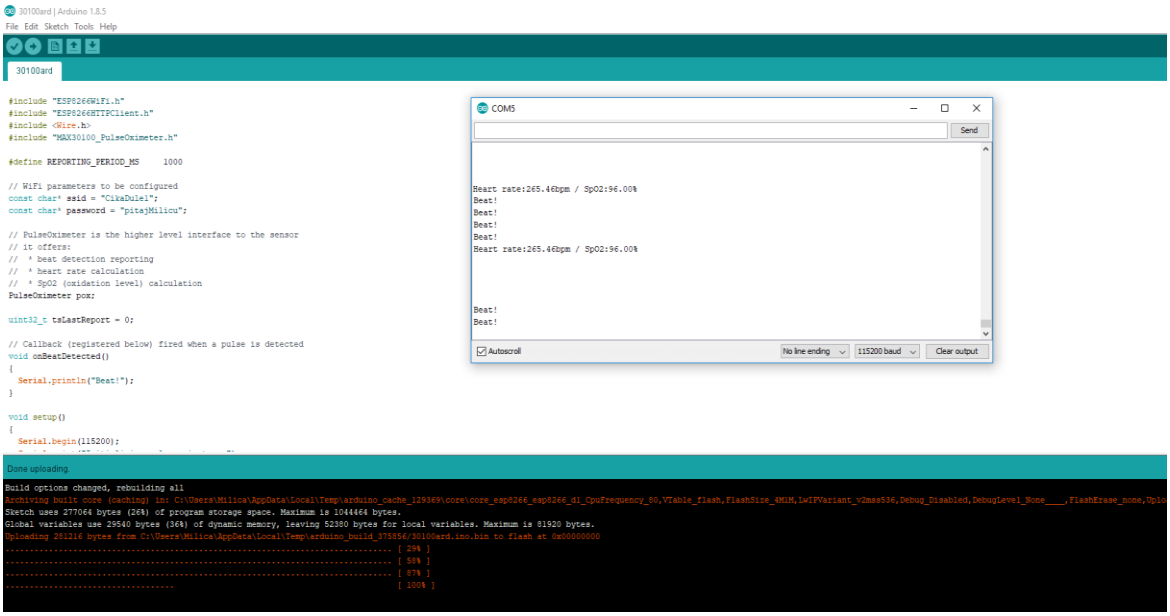
Када смо успешно тестирали модул ESP8266-12f, наредни корак јесте имплементација кода за сензор MAX30100 и код за конектовање ESP8266 на интернет мрежу (**Код *3***). Такође, потребно је инсталирати библиотеке за сензор Max30100 и Esp8266 и укључити их у код преко наредбе **#include.**

```
#include "ESP8266WiFi.h"
#include "ESP8266HTTPClient.h"
#include <Wire.h>
#include "MAX30100_PulseOximeter.h"

// Parametri WIFI mreze
const char* ssid = "naziv";
const char* password = "sifra";

// PulseOximeter je klasa za rad sa senzorom
PulseOximeter pox;

void setup()
{
  Serial.begin(115200);
  Serial.print("Initializing pulse oximeter..");

  // Inicijalizacija PulseOximetar objekta
  // Neuspeh je uglavnom zbog loseg I2C povezivanja
  if (!pox.begin()) {
    Serial.println("FAILED");
    for (;;);
  } else {
    Serial.println("SUCCESS");
  }
  delay(1000);
  WiFi.begin(ssid, password);
  //beskonacna petlja dok se WiFi ne konektuje.
  //Kada uspe, prelazi na loop automatski.
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
}
```

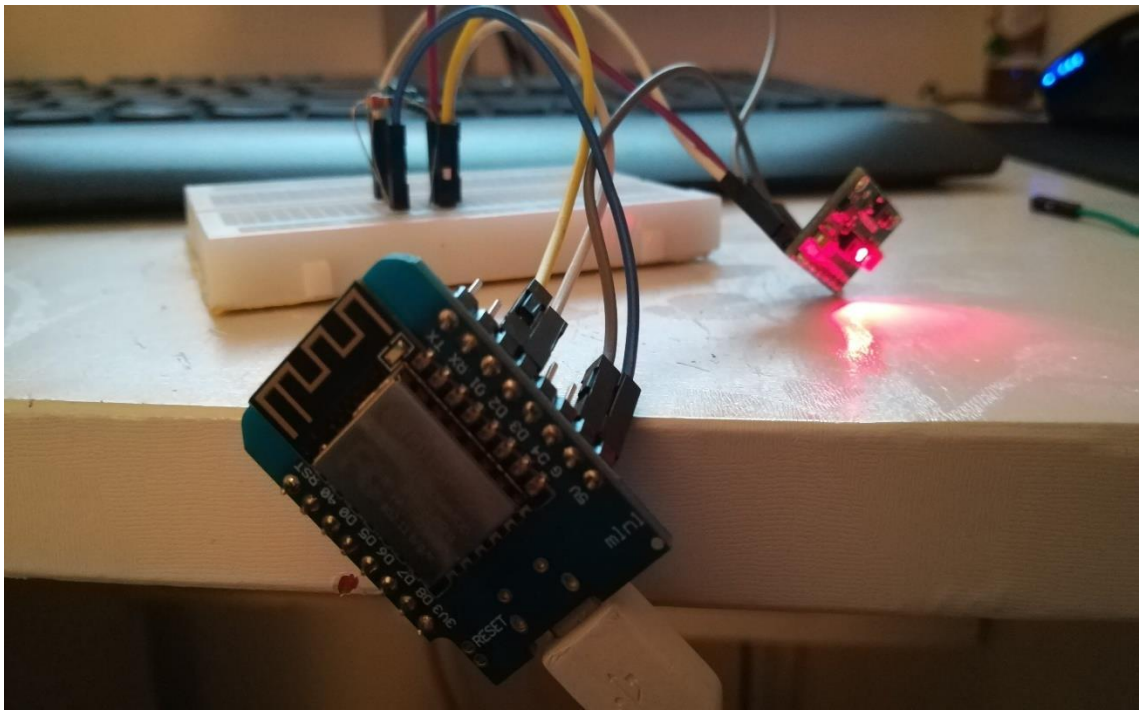
```
void send(String key, double val) {
  HTTPClient http;

  http.begin("http://192.168.0.32/sensorserver/data.php");      //adresa na koju se salje zahtev
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");  //zadavanje header-a

  int httpCode = http.POST("sensor=Max01&key=" + key + "&val=" + String(val, 2)); //slanje podataka POST-om
  String payload = http.getString();                  //uzimanje odgovora

  Serial.println(httpCode);   //Stampanje koda odgovora na serijski monitor
  Serial.println(payload);    //Stampanje odgovora na serijski monitor

  http.end();  //Zatvaranje konekcije
}

void loop()
{
    pox.update(); // Ocitaj nova merenja

    float oxy = pox.getSpO2(); //Ocitaj SpO2 sa senzora
    Serial.print("bpm / SpO2:");
    Serial.print(oxy);
    Serial.println("%");
    send("SpO2", oxy); //Posalji SpO2 na server

    ESP.deepSleep(30e6);//spavaj 30 sekundi
}
```

Код 3: *Имплементација кода за сензор MAX30100 и ESP8266*

Након имплементираног кода и његовог покретања преко **upload** дугмета, на serial монитору можемо видети резултат (**Слика *39***).



Слика 39: *Успешно аплоадован код и приказ резултата на сериал монитору*



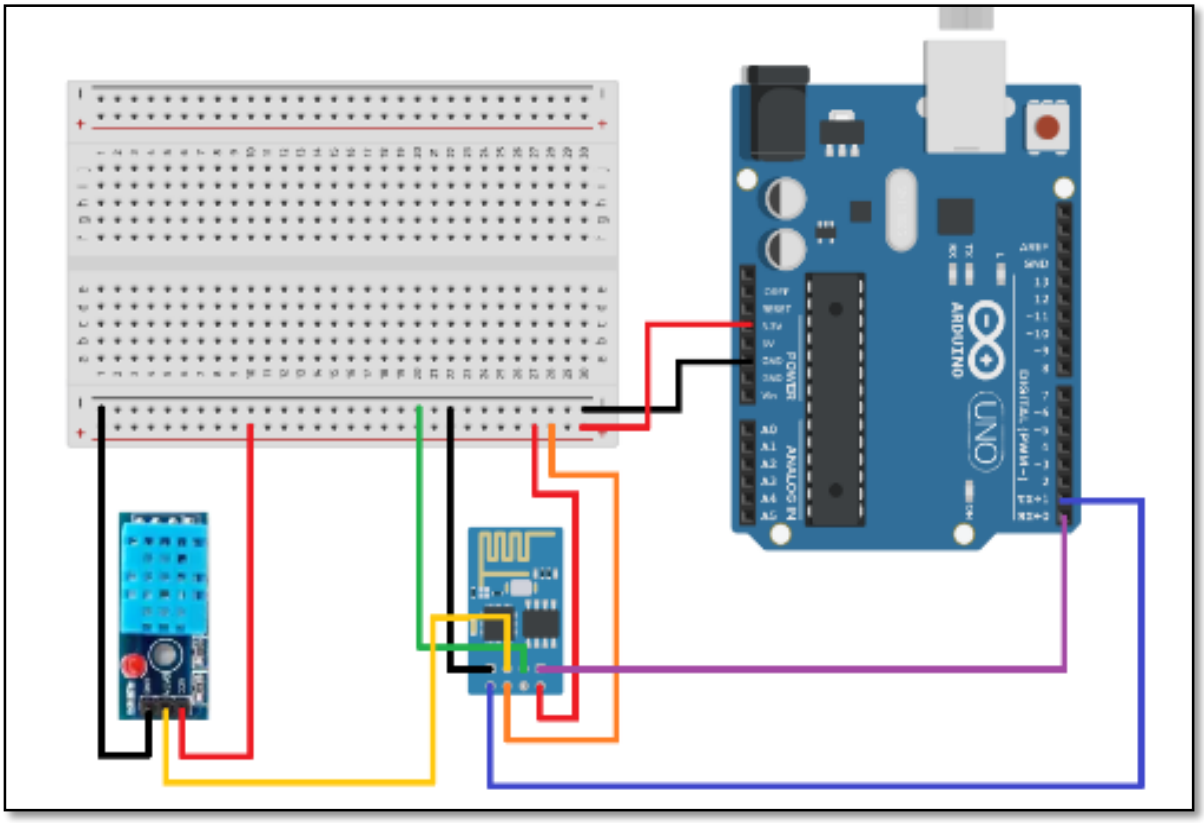
Слика 40: *Успешно повезани MAX30100 и Wemos D1 mini*

4.2. Уређај за мерење температуре и влажности ваздуха

4.2.1. Конекција ESP8266-01/DHT11

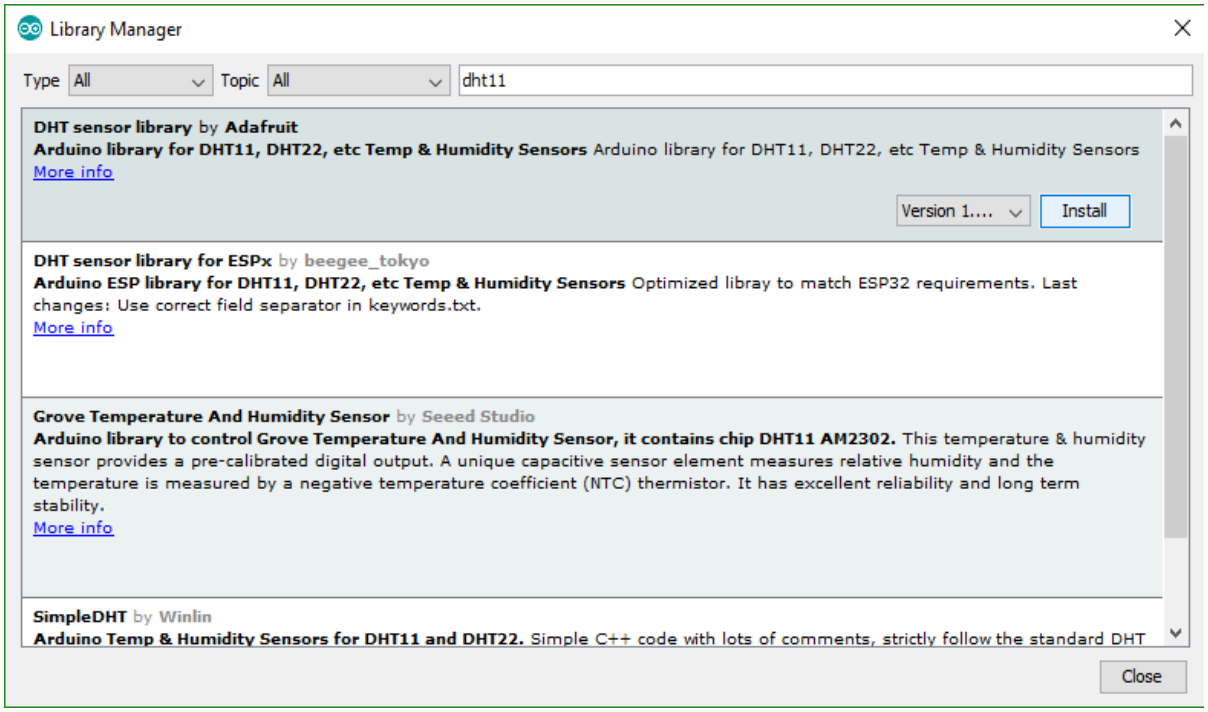
Након успесно повезаног и флешованог модула ESP8266-01, да бисмо у потпуности имали све потребне елементе за прављење пројекта, потребно је повезати и сензор за мерење температуре и влажности ваздуха са ардуином као и са модулом ESP8266-01 јер ће он читати податке са сензора и слати на сервер, одакле ће се кроз апликацију добијати вредности које ће свако ко има приступ апликацији моћи да користи.

Физичко повезивање (**Слика *41***) се остварује тако што се пин VCC на сензору повезује на напон од 3.3 V, а пин GND на GND. Да би се остварила конекција између модула ESP8266-01 и сензора DHT-11 потребно је повезати DATA пин температурног сензора са пином GPIO2 Wifi модула који служи за размену података.

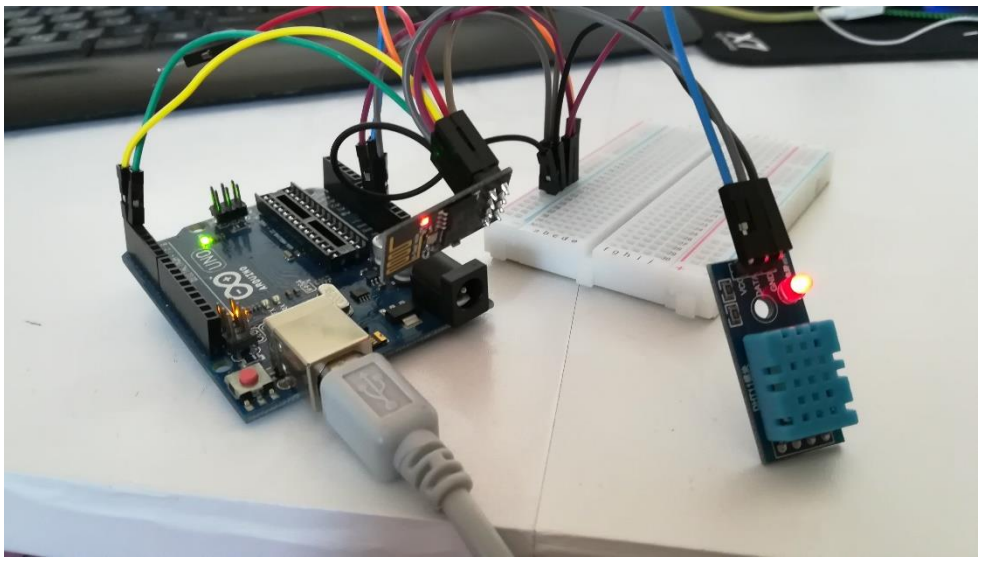


Слика 41: *Повезивање сензора DHT11 и модула ESP8266*

Инсталирање библиотеке се врши преко опције **Sketch**->**Include Library**->**Menage Libraries**. У прозору (**Слика *42***) преко search претражујемо DHT-11 и преко дугмета **install** инсталирамо библиотеку. Након тога, потребно је да се укључи у код преко наредбе **#include** DHT.h. Поред укључене библиотеке за температурни сензор, потребно је у код укључити и библиотеку за Wifi модул ESP8266-01 истим поступком.

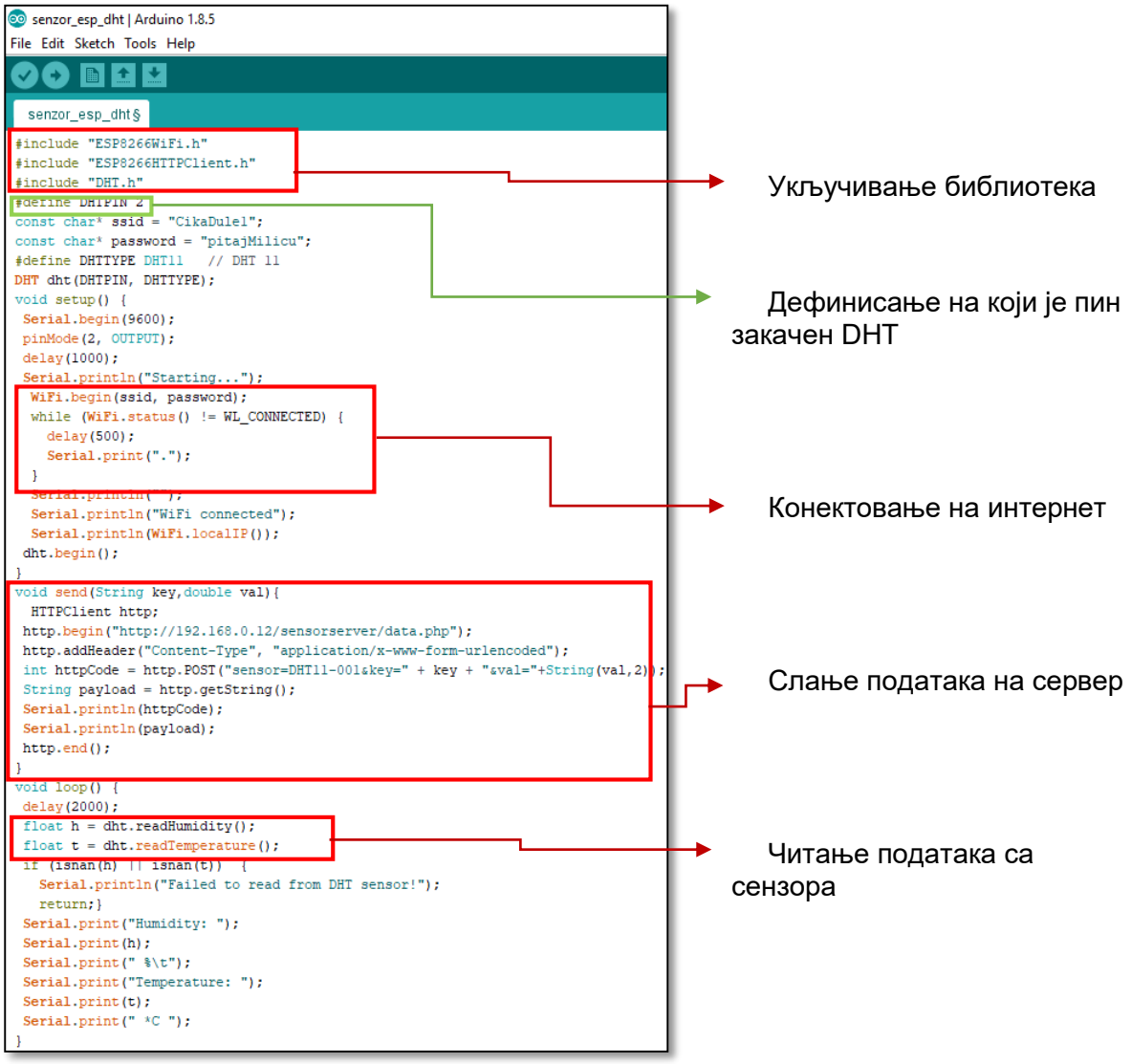


Слика 42: *Инсталирање библиотеке за сензор DHT11*

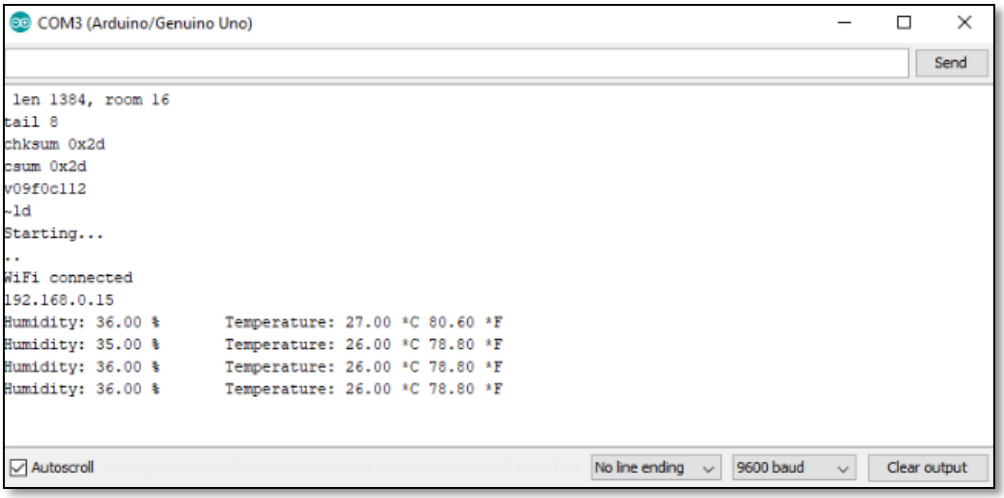


Слика 43: *Успешно повезан сензор DHT-11 са модулом ESP8266-01*

На слици видимо комплетан код који је коришћен у реализацији пројекта који је за циљ имао рачунање температуре и влажности ваздуха преко сензора DHT-11 који је повезан са WiFi модулом ESP8266-01.



Код 4: *Комплетан код за програмирање DHT11 и ESP8266*



Слика 44: *Приказ резултата на сериал монитору*

4.3. База података

База података је реализована као једна табела, у којој су смештени сви подаци.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	*None*		AUTO_INCREMENT
2	kljuc	varchar(255)	utf8_unicode_ci		No	*None*		
3	value	double			No	*None*		
4	time	datetime			No	CURRENT_TIMESTAMP		
5	sensor	varchar(255)	utf8_unicode_ci		No	*None*		

Слика 45: *Приказ табеле у PhpMyAdmin-u*

Табела data садржи колоне:

- **id** – аутоматски генерисана колона.
- **kljuc** – величина коју сензор мери (температура, влажност, ниво кисеоника у крви...)
- **value** – измерена вредност
- **time** – време када је вредност измерена
- **sensor** – јединствени назив сензора (уређаја) на коме је измерена вредност.

Одлучено је да не постоји додатна табела са подацима о сензорима (уређајима), јер би то отежало систем. Оваква база нам омогућава једноставно додавање нових уређаја у систем, без било какве интервенције на серверу или над базом. На слици испод видимо пример података у табели.

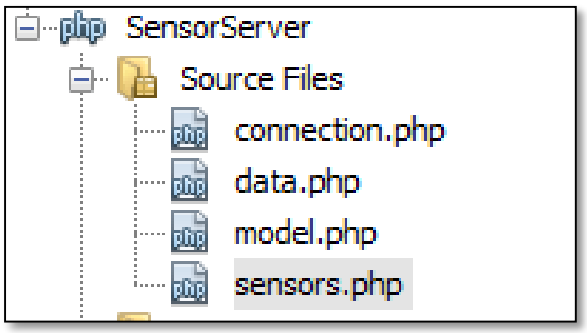
	id	kljuc	value	time	sensor
Edit Copy Delete	5187	humidity	57	2018-09-16 22:10:19	DHT11-001
Edit Copy Delete	5186	temperature	23	2018-09-16 22:10:19	DHT11-001
Edit Copy Delete	5185	humidity	57	2018-09-16 22:10:14	DHT11-001
Edit Copy Delete	5184	temperature	23	2018-09-16 22:10:14	DHT11-001
Edit Copy Delete	5183	humidity	57	2018-09-16 22:10:12	DHT11-001
Edit Copy Delete	5182	temperature	23	2018-09-16 22:10:12	DHT11-001
Edit Copy Delete	5181	humidity	57	2018-09-16 22:10:09	DHT11-001
Edit Copy Delete	5180	temperature	23	2018-09-16 22:10:09	DHT11-001
Edit Copy Delete	5179	humidity	57	2018-09-16 22:10:07	DHT11-001
Edit Copy Delete	5178	temperature	23	2018-09-16 22:10:07	DHT11-001
Edit Copy Delete	5177	humidity	57	2018-09-16 22:10:05	DHT11-001
Edit Copy Delete	5176	temperature	23	2018-09-16 22:10:05	DHT11-001
Edit Copy Delete	5175	humidity	57	2018-09-16 22:10:00	DHT11-001
Edit Copy Delete	5174	temperature	23	2018-09-16 22:10:00	DHT11-001

Слика 46: *Приказ табеле са измереним вредностима који се у њу уписују*

4.4. Сервер

Серверски део програма има дужност да омогући складиштење података са свих уређаја на једном месту. Реализован је као PHP апликација, која памти податке у MySQL базу података. Нема кориснички интерфејс, крајњи корисник није ни свестан постојања овог дела апликације, већ се са њим комуницира путем HTTP захтева.

За развој смо користили Wamp радно окружење, и NetBeans IDE.



Слика 47: *Структура пројекта реализована у NetBeans IDE*

Структура пројекта је једноставна, састоји се из четири фајла. У фајлу connection.php се налазе само параметри за конектовање на базу.

```
class Connection {

    const HOST = 'localhost';
    const DBNAME = "sensordata";
    const USERNAME = 'root';
    const PASSWORD = '';
    public $handler;

    public function __construct() {
        try {
            $this->handler = new PDO("mysql: host=".self::HOST.
                    ";dbname=".self::DBNAME,  self::USERNAME,
                    self::PASSWORD,
                    array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"));
            $this->handler->setAttribute(PDO::ATTR_ERRMODE,
                    PDO::ERRMODE_EXCEPTION);
        }
        catch (PDOException $exc) {
            echo $exc->getMessage();
        }
    }
}
```

Код 5: *Имплементација класе* ***Connection***

Класа Model у себи садржи објекат конекције, и методе за рад са подацима. Метода Get прихвата као параметре сензор и величину (други параметар није обавезан), и за њих из базе враћа 50 најновијих вредности. Метода GetAllLastValues не прихвата

параметре,а враћа нам као резултат последње вредности са свих сензора, заједно са свим подацима о тим мерењима (ид,време...).

Метода Add прихвата као улазне параметре сензор, кључ и вредност, и уписује их у базу (колоне са временом и ид-ом података се аутоматски попуњавају, ово је реализовано на нивоу базе).

```
   class Model {

       private $h;
       public function __construct() {
           $this->h = new Connection();
       }
       public function Get($sensor, $key){
           $addition = "";
           if($key != NULL)
               $addition = "and kljuc like '$key'";
           $q = "select * from data where sensor like '$sensor' $addition order
by time desc limit 50";
           try {
               $rez = $this->h->handler->query($q);
               return $rez->fetchAll(PDO::FETCH_ASSOC);
           } catch (Exception $e) {
               echo $e->getMessage();
               throw new Exception();
           }
       }
       public function GetAllLastValues(){
           $q = "SELECT a.* FROM data a INNER JOIN ( SELECT *, MAX(time) rev
FROM data GROUP BY sensor ) b ON a.sensor = b.sensor AND a.time = b.rev";
           try {
               $rez = $this->h->handler->query($q);
               return $rez->fetchAll(PDO::FETCH_ASSOC);
           } catch (Exception $e) {
               echo $e->getMessage();
               throw $e;
           }
       }
       public function Add($sensor,$key,$val) {
           $query = "insert into data(kljuc,value,sensor) values
('$key',$val,'$sensor');";
           try {
               $this->h->handler->query($query);
           } catch (Exception $e) {
               echo $e->getMessage();
               throw new Exception();
           }
   }
```

Код 6: *Имплементација класе* ***Model***

Фајлови data.php и sensor.php су намењени за коришћење од стране клијента. Састоје се од провере који је захтев извршио клијент, и на основу тога уписују податке у базу које је клијент проследио. Додавање новог мерења се ради **POST** методом, где је неопходно да POST параметар „sensor“ постоји. Уколико клијент жели да учита последње измерене податке за неки сензор, неопходно је да пошаље **GET** захтев, са

```
$m = new Model();
if(isset($_POST['sensor'])){
    $m->Add($_POST['sensor'],$_POST['key'],$_POST['val']);
}
else
//read sensor data and return json serialized
if(isset($_GET['sensor'])){
    $k = NULL;
    if(isset($_GET['key']))
        $k = $_GET['key'];
    echo json_encode($m->Get($_GET['sensor'],$k));
}
else
if(isset($_GET['lastData'])){
    echo json_encode($m->GetAllLastValues());
}
```

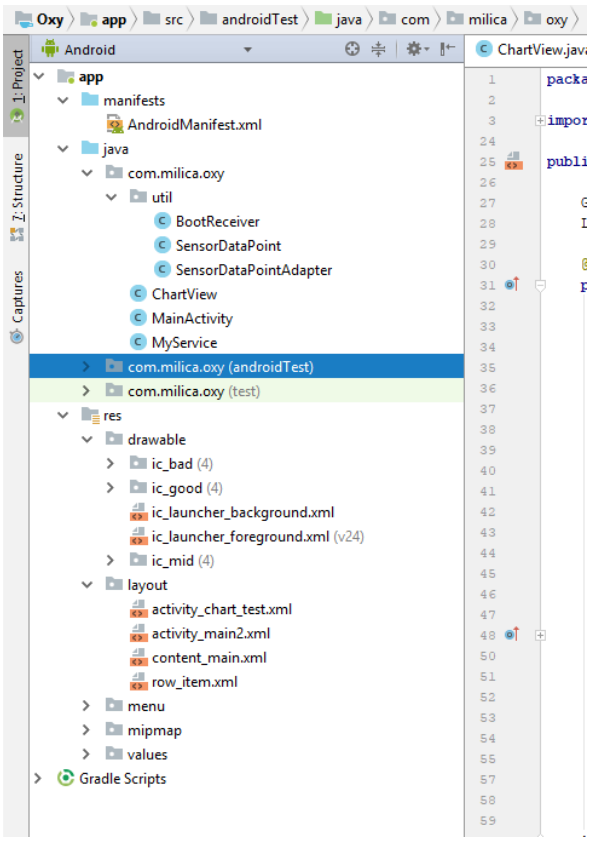
Код 7: *Додавање и читање података из базе*

задатим „sensor“ параметром. Исто важи за учитавање последњих вредности са свих уређаја и GET параметар „lastData“.

4.5. Корисничка апликација

Након успешног физичког повезивања елемената, имплементације кода и добијених жељених резлтата, на ред је дошла Андроид апликација уз помоћ које ће свако ко је има инсталирану на свом мобилном телефону моћи да прати стање корисника који носи уређај, било да је у питању сензор за мерење кисеоника у крви или пак темпертатурни сензор. Апликација је рађена у Android Studio развојном окружењу.

На следећој слици је приказана структура апликације:



Слика 48: *Структура апликације у Андроид развојном окружењу*

Апликација се састоји из два главна екрана (Activity), **MainActivity** и **ChartView**. На првом је приказан списак свих уређаја, са њиховим последњим мерењима и временом када је вредност измерена. Кликом на неки од њих, корисник прелази на следећи екран, где може видети графички приказ последњих измерених вредности у неком периоду. Како би се апликација освежавала најновијим вредностима, направљен је сервис **MyService**, који периодично учитава податке са сервера, и шаље кориснику нотификацију уколико је неки од уређаја забележио вредност испод задате границе. Потребно је да се осигурамо да ће апликација слати кориснику нотификације и када није активна, па је зато направљен **BootReciever**. Његов задатак је да покрене сервис када се укључи телефон, независно од покретања апликације. Класа **SensorDataPoint** служи да држи податке о једном мерењу са уређаја. **SeriesDataPointAdapter** има улогу да прикаже податке о једном мерењу у листи. [16]

```
   public class SensorDataPoint  {
    public long id;
    public String kljuc;
    public String value;
    public String time;
    public String sensor;

    public SensorDataPoint(){

    }

    public Date getDate(){
        try {
            return (new SimpleDateFormat("yyyy-MM-dd kk:mm:ss")).parse(time);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return null;
    }

    public String getNiceDate(){
        return new SimpleDateFormat("dd.MM kk:mm:ss").format(getDate());
    }
}
```

Код 8: *Имплементација класе DataPoint*

```
public class MyService extends Service {

 private Handler mHandler;
 public static String SERVER_URL = "http://192.168.0.32/sensorserver/";

 public MyService() {
 }

 public static final long DEFAULT_SYNC_INTERVAL = 5 * 1000;

 private Runnable runnableService = new Runnable() {
     @Override
     public void run() {
         syncLastData();
         mHandler.postDelayed(runnableService, DEFAULT_SYNC_INTERVAL);
     }
 };

 @Override
 public int onStartCommand(Intent intent, int flags, int startId) {
     mHandler = new Handler();
     mHandler.post(runnableService);

     return START_STICKY;
 }
...

}
```

Код 9: *Имплементација сервиса за учитавање података*

Класа SensorDataPoint представља структуру података какву добијамо са сервера. Поред података, садржи и две једноставне методе за превођење из стринга у датум, као и за форматирани испис датума.

При покретању сервиса се позива метода onStartCommand. Ово смо искористили да позовемо у новој нити методу за учитавање података са сервера (runnableService). На крају вратимо START_STICKY, што говори Android оперативном систему да, ако угаси наш сервис из било ког разлога, неопходно је да га поново укључи чим се за то остваре услови. Како бисмо омогућили стално освежавање података, runnableService у методи run, по позиву учитавања са сервера (syncLastData), позива наредбу postDelayed, и шаље јој сам себе, што омогућава да се поново изврши, уз одлагање од 5 секунди.

```
private synchronized void syncLastData() {
    RequestQueue queue = Volley.newRequestQueue(this);
    String url = MyService.SERVER_URL + "data.php?lastData=1";

    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    Gson gson = new Gson();
                    Log.d("RESPONSE",response);
                    SensorDataPoint[] dataPoints = gson.fromJson(response,
SensorDataPoint[].class);

                    for (int i = 0; i < dataPoints.length; i++){
                        if(new Double(dataPoints[i].value) < 90 &&
dataPoints[i].kljuc.equalsIgnoreCase("SpO2")) {
                            CreateNotification(dataPoints[i]);
                        }
                    }
                    sendMessageToActivity("lastData", response);
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e("error",error.toString());
        }
    });
    queue.add(stringRequest);
}
```

Код 10: *Функција за учитавање последњих података*

Функција syncLastData користи библиотеку Volley за учитавање података са сервера. У овом случају се креира StringRequest објекат, који се додаје на queue. Овај објекат, као један од параметара прихвата Response.Listener, који дефинише шта се ради са одговором од сервера. Користимо библиотеку Gson како бисмо добили објекте од Json стринга. Затим се пролази петљом кроз све податке, и врши се провера да ли је вредност испод 90, као и да ли се ради о мерењу кисеоника (у општем случају, апликација може да приказује различита мерења). У случају да јесте, кориснику се приказује нотификација. У сваком случају се позива метода sendMessageToActivity, која шаље податке Activity-ју за приказ.

Да бисмо могли да учитамо податке путем интернета, неопходно је да апликација тражи од корисника права приступа интернету. Ово се изводи додавањем следеће линије у фајл AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET " />
```

```
   public void CreateNotification(SensorDataPoint dp){
    Intent intent = new Intent(this, MainActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity(getApplicationContext(),
0, intent, 0);

    Notification.Builder builder  = new Notification.Builder(this)
            .setContentTitle("Low oxygen")
            .setContentText("Critical oxygen level at " + dp.time + " on " +
dp.sensor)
            .setContentIntent(pIntent)
            .setAutoCancel(true)
            .setSmallIcon(R.mipmap.ic_launcher)
            .setSound(Settings.System.DEFAULT_NOTIFICATION_URI);
    NotificationManager notificationManager =
            (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        builder.setChannelId("com.myApp");
    }
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
                "com.myApp",
                "My App",
                NotificationManager.IMPORTANCE_DEFAULT
        );
        if (notificationManager != null) {
            notificationManager.createNotificationChannel(channel);
        }
    }
    notificationManager.notify(0, builder.build());
}
private void sendMessageToActivity(String key, String data) {
    Intent intent = new Intent(key);
    intent.putExtra("data", data);
    LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
}
```

Код 11: *Имплементација кода за креирање нотификације*

На слици изнад је приказан код за креирање нотификације и слање поруке према Activity-ју. Код је преузет из званичне документације, уз једну битну измену. За новије верзије Android оперативног система (Opeo па на даље), неопходно је креирати канал за нотификације (NotificationChannel) како би оне функционисале. Овај проблем је незгодан јер се не јавља никаква грешка, већ се само не појављује нотификација, па треба обратити пажњу. Metoda sendMessageToActivity креира Intent са задатим кључем, убацује у њега податке који јој се проследе, и шаље их коришћењем LocalBroadcastManager-a.

```
   public class BootReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {
            Log.d("boot","booted!");
            Intent i = new Intent(context, MyService.class);
            context.startService(i);
        }
    }
}
```

Код 12: *Имплементација класе bootReceiver*

Како би се сервис за стално освежавање података са сервера укључивао увек при паљењу телефона, креирана је класа BootReciever, која наслеђује BroadcastReceiver. Она садржи само једну методу (onReceive), која се аутоматски позива на разне догађаје који се десе.

Ми извршавамо проверу да ли је тај догађај успешно покретање система (ACTION_BOOT_COMPLETED), и ако јесте, покрећемо наш сервис. Да би ово радило, неопходно је да апликација тражи од корисника права приступа. Ово се изводи додавањем следеће линије у фајл AndroidManifest.xml:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

```
public class MainActivity extends AppCompatActivity {
  ArrayList<SensorDataPoint> list = new ArrayList<>();
  SensorDataPointAdapter adapter;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main2);

      final ListView listview = (ListView) findViewById(R.id.listview);

      LocalBroadcastManager.getInstance(this).registerReceiver(
              mMessageReceiver, new IntentFilter("lastData"));

      list = new ArrayList<SensorDataPoint>();
      adapter = new SensorDataPointAdapter(this, list);
      listview.setAdapter(adapter);

      listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
          @Override
          public void onItemClick(AdapterView<?> parent, final View view,
                                  int position, long id) {
              final SensorDataPoint item = (SensorDataPoint)
parent.getItemAtPosition(position);
              Intent i = new Intent(getApplicationContext(), ChartView.class);
              i.putExtra("sensor",item.sensor);
              i.putExtra("key",item.kljuc);
              startActivity(i);
          }
      });
      startService(new Intent(getApplicationContext(), MyService.class));
  }
  private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {
          SensorDataPoint[] data = new
Gson().fromJson(intent.getStringExtra("data"),
                  SensorDataPoint[].class);
          try {
              adapter.getData().clear();
              adapter.getData().addAll(Arrays.asList(data));
              adapter.notifyDataSetChanged();
          }catch (Exception x){
              x.printStackTrace();
          }
      }
  };
}
```

Код 13: *Имплементација кода почетног екрана*

Код за MainActivity је једноставан, неопходно је да прикаже списак свих последњих мерења са уређаја. У методи onCreate, региструје messageReciever, који је задужен за пријем порука од сервиса. Затим креира објекат типа SensorPointAdapter, и прослеђује га листи за преглед података (ова класа ће бити касније описана). Затим методом setOnItemClickListener, дефинише се шта ће се десити на клик на једну ставку листе. Преузимамо објекат на који је кликнуто (item), креирамо нови Intent, у њега смештамо податке о томе на коју је ставку у листи кликнуто. При креирању Intent-а смо проследили тренутну активност, као и класу Activity-ја који желимо да покренемо, па ће метода startActivity покренути баш жељени екран.

mMessageReceiver је објекат типа BroadcastReceiver, који је задужен за пријем података од сервиса, и освежавање података у листи. Користимо Gson библиотеку како бисмо превели податке из стринга у низ објеката. Затим из адаптера почистимо све податке, убацимо нове, и позовемо методу која га обавештава да су се подаци променили, па ће се и листа која је приказана кориснику освежити.

Класа SensorDataPointAdapter дефинише како се приказује једна ставка у списку на главној страни. Код ове класе је преузет са интернет странице:

https://www.journaldev.com/10416/android-listview-with-custom-adapter-example-tutorial , и минимално измењен тако да приказује податке из класе SensorDataPoint.

```
   public class ChartView extends AppCompatActivity {

    GraphView graph;
    LineGraphSeries<DataPoint> series = new LineGraphSeries<>();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chart_test);
        Bundle extras = getIntent().getExtras();
        graph = (GraphView) findViewById(R.id.graph);
        if (extras != null) {
            String sensor = extras.getString("sensor");
            String kljuc = extras.getString("key");
            graph.setTitle(sensor + " " + kljuc + " [" +
SensorDataPointAdapter.getUnit(kljuc) + "]");
            syncData(sensor, kljuc);
        }
        graph.addSeries(series);
        graph.getGridLabelRenderer().setLabelFormatter(new
DefaultLabelFormatter(){
            @Override public String formatLabel(double value, boolean isX){
                if(isX){
                    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
                    return sdf.format(new Date((long)value));
                }
                return super.formatLabel(value,isX);
            }
        });
        graph.getGridLabelRenderer().setPadding(120);
```

Код 14: *Активити са дијаграмом*

Класа ChartView представља Activity на коме је приказан графички приказ историјских података измерених на уређају. Садржи само променљиве graph и series. У методи onCreate преузима Intent којим је креирана, и учитава податке који су јој прослеђени (extras), и методом graph.setTitle поставља наслов графика. Затим методом syncData

учитава податке који треба да се прикажу на графику. Остатак кода представља додавање серије на график, и форматирање графика. За цртање графика користили бесплатну библиотеку GraphView.

```
   private void UpdateDataOnChart(SensorDataPoint data[]){
     DataPoint[] d = new DataPoint[data.length];

     for(int i = 0; i < data.length; i++){
         try {
             Date date = (new SimpleDateFormat("yyyy-MM-dd
kk:mm:ss")).parse(data[i].time);
             d[data.length - 1 - i] = new DataPoint(date, new
Float(data[i].value).floatValue());
         } catch (ParseException e) {
             e.printStackTrace();
         }
     }
     series.resetData(d);
}
```

Код 15:*Освежавање података на графику*

У досадашњем тексту смо већ објашњавали учитавање података са сервера путем библиотеке Volley, па нећемо улазити у детаље методе syncData, пошто је написана на исти начин. Једино што се разликује јесте позив методе UpdateDataOnChart. Ова метода прихвата низ објеката SensorDataPoint, који су учитани са сервера. Креирамо низ објеката типа DataPoint, са којима ради graphView, затим пролазимо петљом кроз податке учитане са сервера, и један по један трансформишемо у DataPoint. Битно је да податке смештамо у нови низ у окренутом редоследу, пошто библиотека graphView захтева сортирање података од најмањег времена ка највећем, а са сервера добијамо прво „најсвежије“ податке (од већег времена ка мањем).

4.6. Коришћење апликације

Апликација која је пројектована назвали смо Oxy. Садржи два екрана. Први(почетни) екран служи за исписивање података које апликација добија са сервера. Обзиром да су за овај пројекат коришћена два сензора(MAX30100 и DHT11), тако се и њихови подаци као што су назив сензора, време читања податка, вредности исписују на екрану у виду листе.

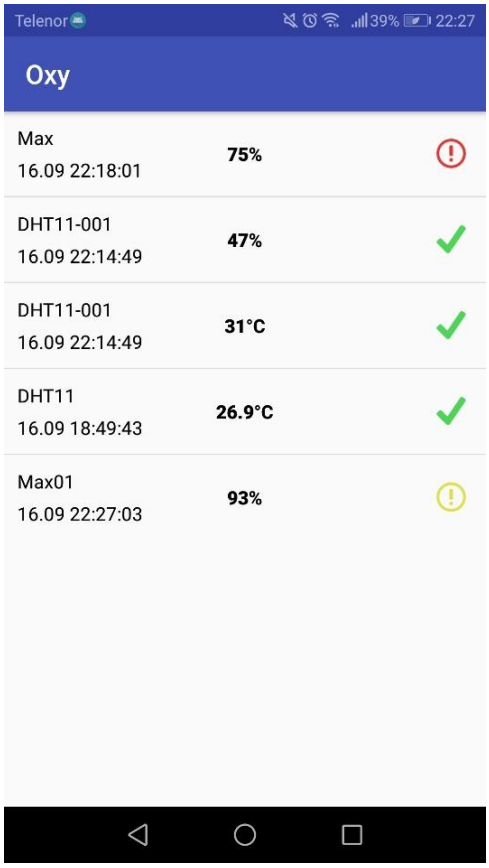
Идеја је да се у зависности од количине кисеоника у крви који се јави код особе која носи уређај појави знак упозорења у случају да ниво кисеоника падне испод границе нормале. То је реализовано тако да се гласна нотификација појави на телефону када се прими вредност испод 90% јер то значи да је стање озбиљно и да је потребно хитно реаговање у циљу побољшања здравственог стања.

Поред вредности у листи на почетном екрану, постављене су три врсте иконица које такође обележавају прагове толеранције нивоа кисеоника.

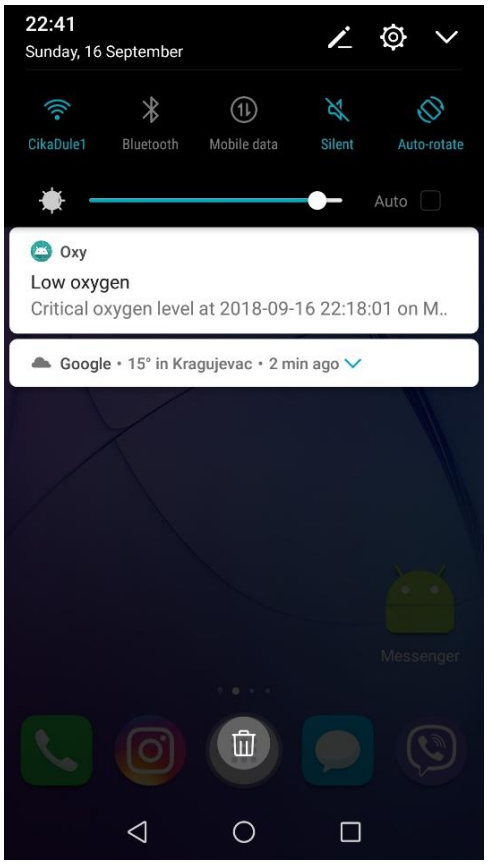
Када се поред добијене вредности појави иконица зелене боје, то значи да нема опасности и да су добијене вредности од 95-100%.

Ако се појави жута иконица, то значи да је ниво кисеоника пао испод 95% и да је у опсегу од 95-90% што значи да се нешто дешава у организму, да је потребно обратити пажњу, али да није аларманто.

Када се појави црвена иконица поред добијене вредности у облику узвичника, њу обавезно прати и гласна нотификација која стиже кориснику и означава хитно реаговање надлежног особља јер то значи да је ниво кисеоника опао испод 90% што представља знак да нешто озбиљно није у реду.



Слика 49: *Почетни екран апликације*

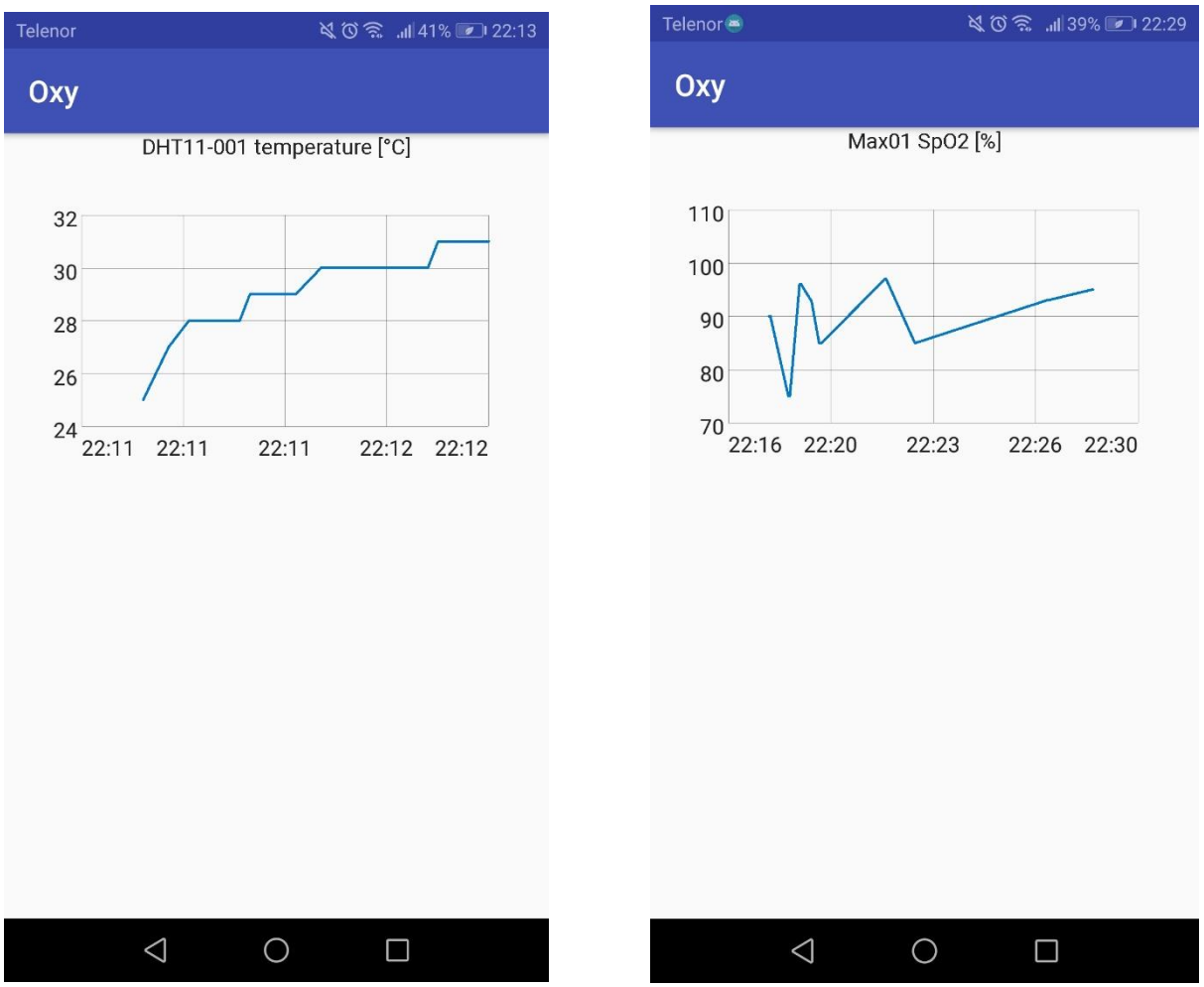


Слика 50: *Екран са нотификацијом*

Што се тиче сензора за мерење температуре и влажности ваздуха, црвена иконица означава да је температура прешла 39 степени, жута да је изнад 37, а зелена означава нормалну температуру до 37 степени.

На слици(***Слика 36***) је приказан екран у тренутку када стигне обавештење у виду нотификације. Када се кликне на обавештење, појављује се почетни екран на коме се виде детаљи.

Ако желимо да испратимо стање корисника у неком одређеном периоду, кликом на сензор из листе одлазимо на нови екран који путем дијаграма приказује како су се вредности мењале у времену (**Слика *51***).



Слика 51: *Приказ дијаграма температуре и SpO2*

4.7. Потрошња батерије

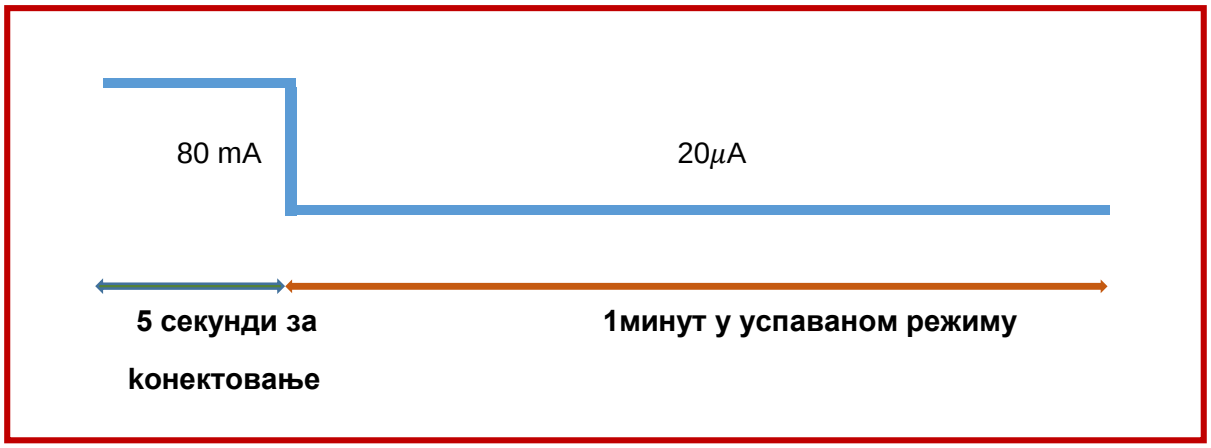
На основу постојеће документације о компонентама које су коришћене за израду пројекта оксиметра, извршен је прорачун који приказује дужину трајања батерије ако се користи функција за deep sleep.

За напајање уређаја коришћен је power bank od 3000mAh, са улазом и излазом од 5 V.

Капацитет батерије:

$$3000mA * 3600\ sec = 10800000mAsec$$

Следећа шема приказује колико милиампера модул esp8266 потроши за време које му је потребно да се конектује на рутер, а колико троши док се налази у deep sleep-у.



Потрошња док је у deep sleep-у:

$$20\mu * 60\ sec = \mathbf{1.2mAsec}$$

Потрошња док се конектује:

$$80mA * 5\ sec = \mathbf{400mAsec}$$

За један циклус потроши:

$$400mAsec + 1.2mAsec = \mathbf{401.2mAsec}$$

Број циклуса које оствари са једном батеријом износи:

$$\frac{10800000\ \text{mAsec}}{401.2\text{mAsec}} = \mathbf{26920\ ciklusa}$$

Трајање батерије добијамо множењем броја циклуса са трајањем једног циклуса:

$$Trajanje = 26920 * 65sec = \mathbf{1749800\ sec}$$

$$Trajanje\ u\ satima = 1749800sec/3600 = \mathbf{486.0556\ h}$$

$$Trajanje\ baterije\ u\ danima = \frac{486.0556}{24} = \mathbf{20.25\ dana}$$

5. ЗАКЉУЧАК

Направили смо уређај за мерење кисеоника у крви који ће преко Wireless модула ESP8266 комуницирати са Андроид апликацијом и уз то додали и сензор који мери температуру и влажност ваздуха. Свако ко има инсталирану апликацију на свом смарт телефону може да приступи подацима који пристижу са сензора, независно од места на коме се налазе.

За постојећа решења уређаја који мере ниво кисеоника у крви путем бежичне мреже коришћени су Bluetooth модули и они се увелико налазе на тржишту. Њихова мана је та што да би систем функционисао, уређај и телефон морају бити у непосредној близини, док за наш уређај то није случај јер смо користили Wifi модул ESP8266 који нам омогућава да и на другој страни планете можемо пратити стање особе која носи уређај Oxy.

Даља побољшања овог решења би могла представљати омогућавање додатних опција кориснику путем апликације, моделирање и израда кућишта за уређај и батерије, као и постављање дисплеја на сам уређај. Потребно је даље истраживање о начину спољног напајања и одабрати боље решење које укључује дуже трајање батерије и величину исте јер је једна од идеја овог рада била да се направи уређај што мањих димензија.

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

Литература

[1] http://medikor.hr/wp/disanje-kisik-i-pulsna-oksimetrija/

[2] https://www.natural-sciences.ru/ru/article/view?id=8639

[3] https://energymicroblog.files.wordpress.com/2012/11/figure-1.png

[4]https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf

[5]http://incenter.medical.philips.com/doclib/enc/fetch/586262/586457/Understanding_Pulse_Oximetry.pdf%3Fnodeid%3D586458%26vernum%3D2

[6] https://www.instructables.com/id/Finger-Pulse-Oximeter-Using-MAX30100/

[7] https://store.arduino.cc/arduino-uno-rev3

[8] https://www.espressif.com/en/products/hardware/esp8266ex/overview

[9] https://learn.adafruit.com/dht/using-a-dhtxx-sensor

[10] https://www.wemos.cc/

[11] https://www.arduino.cc/

[12] https://www.knjizara.com/pdf/142076.pdf

[13] https://developer.android.com/

[14] https://www.arduino.cc/en/Tutorial/Blink

[15]http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2015_08_25_Aleksandar_Ilic/rad.pdf

[16] James Steele, Nelson To: Андроид: Израда апликација помоћу пакета Андроид SDK; Микро књига, Београд, 2011.

[17] James Talbot, Justin McLean: Програмирање Андроид апликација; Cet, 2014.

[18] Bert van Dam: Ардуино уно: 45 пројеката за почетнике и стручњаке; ЕНО,2016.

[19] Simon Monk: Ардуино: увод у програмирање, превод 2. издања; Микро књига, Београд, 2017.

[20] Prof Dr Dogan Ibrahim i Ahmet Ibrahim: ESP8266 i MicroPython; ЕНО, 2017.