UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED

VYUŽITIE SMART ZARIADENÍ PRE VYHODNOTENIE FYZIOLOGICKÝCH STAVOV POUŽÍVATEĽA

DIPLOMOVÁ PRÁCA

2021 **Bc. Dmytro Virich**

UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED

VYUŽITIE SMART ZARIADENÍ PRE VYHODNOTENIE FYZIOLOGICKÝCH STAVOV POUŽÍVATEĽA

DIPLOMOVÁ PRÁCA

Študijný program:	9.2.9 Aplikovaná informatika
Študijný odbor:	Aplikovaná informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. Ing. Zoltán Balogh, PhD.

Nitra 2021 Bc. Dmytro Virich

Univerzita Konštantína Filozofa v Nitre
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Dmytro Virich
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: Diplomová práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Využitie Smart zariadení pre vyhodnotenie fyziologických stavov používateľa

Anotácia: V súčasnosti existujú rôzne technické riešenia ako aj metódy, pomocou ktorých je možné získať fyziologické funkcie používateľa. Najčastejšie sa z pohľadu technického riešenia využívajú prístupy vychádzajúce z aplikovania senzorov (GSR, Heart sensors, temperature and others), prípadne využitia webkamery pre snímanie tváre a jej častí.
Cieľom diplomovej práce je navrhnúť a vytvoriť pomocou dostupných smart zariadení (Smart watch) systém pre identifikovanie a klasifikáciu emocionálneho stavu používateľa pomocou dostupných fyziologických funkcií. Cieľom je vytvorenie systému na monitorovanie základných životných funkcií pri zachovaní neinvazívnosti, ktorého súčasťou je používateľské rozhranie na báze IoT. Navrhne sa vhodný postup na meranie fyziologických funkcií pomocou dostupných smart zariadení. Súčasťou bude aj webová, prípadne mobilná aplikácia na vizualizáciu a vyhodnocovanie získaných dát.
Charakter práce: aplikačný
Požiadavky na obsah podľa charakteru práce: popis riešeného problému, návrh systému/hardvérového riešenia a pod. (modely, ..), metodika vývoja/tvorby, implementácia, popis vytvoreného riešenia, testovanie;
Požiadavky na vedomosti a zručnosti študenta: dobrá znalosť anglického jazyka, programovanie v ľubovoľnom jazyku, prehľad v IoT, tvorba mobilnej aplikácie, znalosť programovania Arduina a Raspberry Pi.

Školiteľ: doc. Ing. Zoltán Balogh, PhD.
Oponent: prof. Ing. Milan Turčáni, CSc.
Katedra: KI - Katedra informatiky

Dátum zadania: 02.10.2019

Dátum schválenia: 15.10.2019

RNDr. Ján Skalka, PhD., v. r.
vedúci/a katedry

ABSTRAKT

VIRICH, Dmytro: Využitie Smart zariadení pre vyhodnotenie fyziologických stavov používateľa. [Diplomová práca]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied. Školiteľ: doc. Ing. Zoltán Balogh, PhD. Stupeň odbornej kvalifikácie: Magister odboru Aplikovaná informatika. Nitra: FPV, 2021. 82s.

Diplomová práca sa zaoberá identifikáciou a klasifikáciou emocionálneho stavu používateľa pomocou dostupných fyziologických funkcií. Na identifikáciu je možné použiť moduly a senzory, ako je napríklad GSR, EKG, snímač telesnej teploty a ďalšie možné snímače. Hlavnou podmienkou je zachovanie neinvazívnosti pri meraní. Príkladom môže byť bezkontaktný infračervený snímač GY-906, ktorý bol použitý pre bezkontaktné meranie telesnej teploty používateľa, čím sa zachováva princíp neinvazívnosti.

V úvodnej teoretickej časti sa zaoberáme analýzou súčasného stavu IoT a klasifikácie ľudských emócii pomocou dostupných fyziologických funkcií, ich popis.

V praktickej časti sa zaoberáme vytváraním systému pre meranie dôležitých fyziologických funkcií človeka. Podrobne popisujeme z ktorých komponentov sa skladá vytvorený systém a aký konkrétny software bol použitý.

V závere sa venujeme tomu ako navrhnutý systém funguje, kde v systéme sa ukladajú dáta, aby sme ich mohli poslať do databázy a zobraziť získané merania a grafy vo webovej aplikácii.

Kľúčové slová: Raspberry Pi, IoT, Arduino, Mikrokontrolér, Internet vecí, Senzory, ATMEGA328P.

ABSTRACT

VIRICH, Dmytro: Use of Smart devices to evaluate the physiological states of the user. [Diploma Thesis]. Constantine the Philosopher University in Nitra. Faculty of Natural Sciences. Supervisor: doc. Ing. Zoltán Balogh, PhD. Degree of Qualification: Master of Applied Informatics. Nitra: FNS, 2021. 82 p.

The diploma thesis deals with the identification and classification of the user's emotional state using available physiological functions. Modules and sensors such as GSR, ECG, body temperature sensor and more can be used for identification. The main condition is to maintain non-invasive measurements. An example is the non-contact infrared sensor GY-906, which was used for non-contact measurement of the user's body temperature, thus supporting the principle of non-invasiveness.

In the introductory theoretical part we deal with the analysis of the current state of IoT and the classification of human emotions using available physiological functions and describe them.

In the practical part we deal with the creation of a system for measuring important physiological functions of a person. We describe in detail which components the created system consists of and what specific software was used.

Finally, we focus on how the proposed system works, where the system stores data to then send them to a database and display the resulting measurements and graphs in a web application.

Keywords: Raspberry Pi, IoT, Arduino, Microcontroller, Internet of Things, Sensors, ATMEGA328P.

OBSAH

Zoznam skratiek a značiek

SDA – Serial Data – Sériové údaje

SCL – Serial Clock – Sériové hodiny

I2C – Inter-Integrated Circuit – Medzi integrovaný obvod

PWM – Pulse Width Modulation – Impulzová šírková modulácia

EKG – Electrocardiogram

RFID – Radio Frequency Identification – Rádiofrekvenčná identifikácia

IoT – Internet of Things – Internet vecí

SoC – System on Chip – Systém na čipe

ARM – Advanced RISC Machines – Pokročilé výpočty RISC

LED – Light-Emitting Diode – Dióda vyžarujúca svetlo

GSR – Galvanic Skin Response – Galvanický prúd na pokožku

HTML – HyperText Markup Language – Hypertextový Značkovací Jazyk

CSS – Cascading Style Sheet – List Kaskádových Štýlov

FOV – Field Of View – Zorné Pole

IDE – Integrated Development Environment – Integrované Vývojové Prostredie

M2M – Machine to Machine Communication – Komunikácia medzi strojmi

ECS – Electrocardiosignal – Elektrokardiosignál

EDA – Electrodermal activity – Elektrodermálna aktivita

SC – Skin conductance – Vodivosť pokožky

SNS – Sympathetic nervous system – Sympatický nervový systém

POE – Power over Ethernet – Napájanie cez Ethernet

RAM – Random Access Memory – Pamäť s priamym prístupom

ADC – Analog-to-Digital Converters – Analógovo-digitálny prevodník

SDK – Software Development Kit – Súprava na vývoj softvéru

AVD – Android Virtual Device – Virtuálne zariadenie Android

ÚVOD

Diplomová práca sa zaoberá existujúcimi technickými metódami a riešeniami, pomocou ktorých je možné merať dôležité fyziologické funkcie používateľa. Cieľom diplomovej práce je navrhnúť a vytvoriť pomocou dostupných smart zariadení (Smart watch) systém pre identifikovanie a klasifikáciu emocionálneho stavu používateľa pomocou dostupných fyziologických funkcií. Cieľom je vytvorenie systému na monitorovanie základných životných funkcií pri zachovaní neinvazívnosti, ktorého súčasťou je používateľské rozhranie na báze IoT. Navrhneme vhodný postup na meranie fyziologických funkcií pomocou dostupných smart zariadení. Súčasťou bude aj webová, prípadne mobilná aplikácia na vizualizáciu a vyhodnocovanie získaných dát.

V prvej kapitole sa zaoberáme klasifikáciou a popisom dôležitých fyziologických funkcií človeka ako sú teplota tela, meranie srdcovej frekvencie, saturácia krvi kyslíkom, elektrokardiografia, galvanický odpor kože. Popis toho, čo je IoT, malá história a súčasný stav vývoja IoT.

V ďalšej kapitole sa venujeme existujúcim modulom a senzorom na čítanie dôležitých fyziologických funkcií človeka, pozeráme sa na miniatúrny jednodoskový počítač Raspberry Pi a Arduino Nano čo je kompaktná doska postavená na mikrokontroléri ATMega328. Zaoberáme sa vlastnosťami a špecifikáciami snímačov, miniatúrneho počítača a mikrokontroléra. Popisujeme možnosť pripojenia Raspberry Pi a Arduino Nano medzi sebou, možnosti pripojenia senzorov tak, aby sme zabezpečili ich správne fungovanie. Zaoberáme sa tvorbou diagramov a schémami zapojenia ktoré popisujú, ako jednotlivé moduly a senzory fungujú a ako správne pripojiť každý z prvkov zariadenia. Dôležitou úlohou bolo uložiť dáta prijaté z modulov a senzorov s ich ďalším odoslaním do databázy na ich zobrazenie vo webovej aplikácii. Zaoberáme sa vlastnosťami a špecifikáciami vybraného databázového riešenia Firebase Realtime Database. Popisujeme možné nastavenia databázy a možnosti ukladanie dát. Detailne sa venujeme tvorbe a realizácií webovej a mobilnej aplikácie. Vytvárame jej vizuálnu časť. Popisujeme a vytvárame samostatné časti webovej a mobilnej aplikácie, ktoré zodpovedajú za zobrazovanie hodnôt z databázy a vytváranie grafov EKG a GSR na základe týchto hodnôt. Na konci sa zaoberáme navrhnutým systémom, ako aj jeho testovaním.

1 ANALÝZA SÚČASNÉHO STAVU

Zdravie je najdôležitejšou súčasťou života každého človeka. Väčšina ľudí žije hektickým životom, v ktorom je týždenná alebo dokonca mesačná kontrola u lekára nemožná úloha. Bez sledovania vášho zdravia nie je možné zistit, či ste zdravý alebo chorý človek. Monitorovanie zdravia je hlavným problémom súčasného sveta. Kvôli nedostatku náležitého monitorovania zdravia trpí človek vážnymi zdravotnými problémami. Prispievajú k tomu zlé zdravotné služby, prítomnosť veľkých rozdielov medzi vidieckymi a mestskými oblasťami, nedostupnosť lekárov a zdravotných sestier. Jednou z hlavných úloh dnešného vývoja medicíny je distribúcia a vývoj inteligentných zariadení, ktoré sú potrebné na sledovanie stavu pacienta v reálnom čase alebo na získanie dôverných údajov s cieľom ich následnej analýzy pre lekársku diagnostiku.

Rozpoznávanie emócií môže poskytnúť vedecký základ pre sledovanie emočného zdravia a skríning fyziológie a duševných chorôb súvisiacich s emóciami. Emócie sa nevyjadrujú iba psychologickým správaním, ale aj sériou fyziologických zmien. Tieto fyziologické zmeny nie sú ľuďmi subjektívne kontrolované. Fyziologické signály tak môžu objektívnejšie odrážať skutočné pocity subjektov (Xiefeng et al., 2019).

Internet vecí (IoT) je nová revolúcia internetu, ktorá predstavuje rastúcu oblasť výskumu. Internet vecí sa neustále vyvíja. Ďalší potenciál sa hodnotí kombináciou prístupov a konceptov zo súvisiacich technológií, ako sú cloud computing, internet budúcnosti, big data, robotika a sémantické technológie (Vermesan a Friess, 2013). IoT môže byť pre aplikácie v zdravotníctve skutočne prospešný. Môžeme použiť senzory, ktoré dokážu merať a monitorovať rôzne medicínske parametre v ľudskom tele (Sethi, Sarangi, 2017).

S nárastom používania nositeľných senzorov a inteligentných telefónov sa tieto vzdialené monitorovanie zdravotnej starostlivosti vyvíja rýchlym tempom. Monitorovanie zdravia pomocou internetu vecí pomáha predchádzať šíreniu chorôb a tiež správne diagnostikovať zdravotné stavy používateľa, aj keď je lekár vo veľkej vzdialenosti. Monitorovací systém zdravia pacientov IoT je všeobecný pojem pre akýkoľvek zdravotnícky prístroj, ktorý má prístup na internet a dokáže merať niekoľko ukazovateľov zdravia pacienta pripojeného k prístroju, ako napríklad srdcový tep, krvný tlak, EKG, kroky atď. Pod touto definíciou sa rozumejú zariadenia ako inteligentné hodinky, sledovače fitnes, inteligentné telefóny až po drahé nemocničné zariadenia, ktoré sa dajú

pripojiť na internet. Takéto zariadenie môže zaznamenávať, prenášať a varovať ak dôjde k náhlej zmene zdravotného stavu pacienta.

Automatické rozpoznávanie emócií sa zvyčajne vykonáva meraním rôznych parametrov ľudského tela alebo elektrických impulzov v nervovom systéme a analýzou ich zmien (Dzedzickis, Kaklauskas a Bucinskas, 2020).

Táto práca je venovaná vývoju kompaktného zariadenia a implementácii efektívneho systému monitorovania zdravia založeného na IoT. Navrhovaný systém monitoruje dôležité parametre ľudského zdravia, ako napríklad meranie pulzu, meranie telesnej teploty, meranie saturácie krvi kyslíkom, čítanie galvanickej odozvy kože, meranie srdcového rytmu s možnosťou po meraní bezdrôtovo poslať získané údaje prostredníctvom Wi-Fi do databázy. K prijatým údajom je možné kedykoľvek pristupovať pomocou webovej aplikácie alebo aplikácie vytvorenej pre mobilný operačný systém Android.

1.1 HISTÓRIA A POUŽITIE IOT

Internet vecí sa vyvíja rýchlym tempom a naďalej je najnovším a najobľúbenejším konceptom vo svete informačných technológií. Internet vecí možno tiež považovať za globálnu sieť, ktorá umožňuje komunikáciu medzi ľuďmi, ľuďmi a vecami, čo je všetko na svete, a poskytuje každému objektu jedinečnú identitu (Madakam, 2015).

Internet vecí predstavuje úzku integráciu skutočného a virtuálneho sveta, v ktorom sa komunikácia medzi ľuďmi a zariadeniami uskutočňuje pomocou rôznych zariadení a senzorov, ktoré sú navzájom prepojené káblovými aj bezdrôtovými komunikačnými kanálmi a umožňujú im zhromažďovať, analyzovať, spracovávať a prenášať údaje do iných objektov pomocou softvéru, aplikácií alebo technických zariadení. Hlavnou úlohou IoT je zabezpečiť ľudský život pohodlne a bezpečne. Internet vecí sprevádza všetky činnosti používateľov a je zameraný na výsledky.

Platformu ekosystému IoT možno v zásade rozdeliť do troch úrovní, a to na úroveň snímania (pre generovanie údajov), komunikačnú úroveň (pre pripojenie a prenos údajov) a úroveň pre správu (pre zber, ukladanie, úpravu a správu údajov). Úroveň snímania zahŕňa mobilné alebo stacionárne zariadenia, ktoré môžu generovať údaje v rôznych formátoch, zatiaľ čo úroveň komunikácie zahŕňa existujúce a vznikajúce drôtové alebo bezdrôtové komunikačné siete. Na úrovni riadenia sú potrebné technológie na zber, ukladanie a analýzu údajov (Bello, 2016).

Základom internetu vecí je technológia komunikácie medzi strojmi, ktorá má aj označenie M2M. To je, keď stroje používajú mobilné siete na vzájomnú výmenu

informácií alebo ich jednostranný prenos. Technológia M2M sa používa v systémoch bezpečnosti a ochrany zdravia, v oblasti výroby, komunálnych službách, energetike a bankových systémoch.

Softvérové platformy IoT sa líšia v nasledujúcich technických charakteristikách:

- Škálovateľnosť – maximálny možný počet zariadení IoT pripojených k platforme.
- Jednoduchosť použitia – flexibilita aplikačného programovacieho rozhrania a ľahká práca so zdrojom.
- Spôsoby nasadenia – súkromné alebo verejné cloudové úložisko.
- Bezpečnosť – zabezpečenie ochrany informácií šifrovaním, monitorovaním prístupu a pod.
- Databáza – organizačná štruktúra na ukladanie informácií prenášaných zo zariadení.

Jadrom internetu vecí sú tieto technológie:

- Identifikačné nástroje. Každý objekt vo fyzickom svete, ktorý sa zúčastňuje na internete vecí, musí mať jedinečný identifikátor. Na automatickú identifikáciu objektov je možné použiť rôzne existujúce systémy: vysokofrekvenčný systém, pri ktorom je ku každému objektu pripojený vysokofrekvenčný štítok, optický (čiarové kódy, dátová matica, QR kódy), infračervené štítky atď.
- Meracie nástroje. Zabezpečenie transformácie informácií o externom prostredí na údaje vhodné na ich prenos do spracovateľských zariadení. Môžu to byť buď samostatné snímače teploty, snímače svetla atď. alebo zložité meracie systémy. Na dosiahnutie autonómie meracích prístrojov je žiadúce, aby bolo realizované napájanie snímačov pomocou alternatívnej energie.
- Prostriedky na prenos údajov. Na prenos údajov je možné použiť ktorúkoľvek z existujúcich technológií. V prípade použitia bezdrôtových sietí sa osobitná pozornosť venuje zlepšeniu spoľahlivosti dátového prenosu. Pri používaní káblových sietí sa aktívne využíva technológia prenosu dát po elektrických vedeniach, pretože veľa zariadení, ako sú automaty, bankomaty atď., je pripojených k energetickým sieťam.
- Zariadenia na spracovanie údajov. Hlavnou súčasťou internetu vecí nie sú senzory a prostriedky na prenos dát, ale cloudové systémy, ktoré poskytujú veľkú šírku pásma a sú schopné rýchlo reagovať na určité situácie. Napríklad,

aby sa dalo z údajov získaných zo senzorov zistiť, že v dome nikto nebol päť minút a predné dvere zostali otvorené.

- Výkonné zariadenia. Jedná sa o zariadenia schopné prevádzať digitálne elektrické signály z informačných sietí v činnosť. Napríklad na to, aby sa dal vykurovací systém v dome zapnúť prostredníctvom smartfónu, musí mať príslušné zariadenie. Výkonné zariadenia sú často konštrukčne kombinované so snímačmi.

Myšlienka, že zariadenia si môžu navzájom vymieňať informácie bez ľudského zásahu, sa objavila už dávno. Aj na konci 70. rokov sa diskutovalo o možnosti úplnej automatizácie prenosu dát. V tom čase sa tento prístup nazýval všadeprítomné výpočty.

V roku 1990 John Romkey, jeden z tvorcov protokolu TCP/IP, pripojil svoj vlastný hriankovač k internetu a umožnil jeho diaľkové zapínanie a vypínanie.

Pojem internet vecí, ktorý v roku 1999 vytvoril Ashton, priťahoval a priťahuje množstvo výskumného a priemyselného záujmu (Perrone, 2019). V tom istom roku bolo vytvorené aj samotné výskumné centrum, ktoré sa zaoberalo rádiofrekvenčnou identifikáciou (RFID) a senzorickými technológiami. Technológia RFID umožňuje pripevniť na objekty malé štítky obsahujúce dôležité informácie a umožňuje ich čítanie z diaľky.

Práve vďaka týmto oblastiam, sa stala táto koncepcia rozšírená. Ashton mohol byť prvý, kto použil výraz internet vecí, ale koncept pripojených zariadení, najmä pripojených strojov, existuje už dlho. Napríklad stroje navzájom komunikovali od doby, keď sa koncom 30. rokov 19. storočia vyvinuli prvé elektrické telegrafy. Iné technológie, ktoré vstúpili do IoT bol rádiový prenos hlasu, bezdrôtové technológie (Wi-Fi) a softvér na kontrolu a získavanie údajov (SCADA). V roku 1982 sa stal prvým pripojeným inteligentným zariadením automat Coca-Cola na Carnegie Mellon University. Pomocou miestnej siete univerzity mohli študenti zistiť, aké nápoje sa v sklade nachádzali a či boli studené.

V rokoch 2000 až 2010 sa začali objavovať úspešné projekty internetu vecí a hromadne sa začali používať v realite. Týmto spôsobom bolo vyvinutých veľa spotrebných zariadení internetu vecí, od fitness trackerov až po inteligentné žiarovky a inteligentné dvere. Okrem toho sa začali rozvíjať rozsiahle projekty založené na technológiách IoT - inteligentné mestá, inteligentná výroba, inteligentná doprava, bezpilotné vozidlá a oveľa viac.

V období od roku 2008 až do roku 2009 podľa analytikov spoločnosti Cisco Corporation počet zariadení pripojených k sieti WWW prekročil počet obyvateľov Zeme.

Internet vecí umožňuje spoločnostiam automatizovať procesy a znižovať náklady na pracovnú silu. To znižuje objem odpadu, zlepšuje kvalitu poskytovaných služieb a znižuje náklady na výrobu a logistiku. Z technického hľadiska internet vecí nie je výsledkom jednej novej technológie. Namiesto toho je k dispozícii niekoľko ďalších technických pokrokov, ktoré pomáhajú prekonávať priepasť medzi virtuálnym a fyzickým svetom (Mattern, Floerkemeier, 2010). IoT ovplyvňuje všetky odvetvia.

IoT v poľnohospodárstve. Všetky mechanizmy sú vybavené senzormi. Keď traktor zorá pôdu, senzor analyzuje stav pôdy v konkrétnej oblasti, systém sám vypočíta potrebné množstvo a typ hnojív a aplikuje ich na zem. V procese rastu rastlín senzory monitorujú hladinu vlhkosti, hladinu živín a škodcov. Len čo jeden z ukazovateľov nie je normálny, systém okamžite urobí rozhodnutie a všetko opraví. Ak senzor zistí, že je zem suchá, systém zapne zavlažovanie presne tam, kde je to potrebné, a v požadovanom množstve. Tento prístup zvyšuje výnosy a šetrí zdroje. Holandsko ako malá krajina s vysokou hustotou obyvateľstva je jedným zo svetových lídrov v pestovaní potravín - je to možné vďaka IoT.

IoT v priemysle. Zavedenie tejto technológie umožňuje spoločnostiam ušetriť na údržbe a opravách zariadení. Použitie špeciálnych senzorov poskytne aktuálne údaje o stave zariadenia a zabráni poruchám a prestojom, maximalizuje výkon zariadenia, zníži náklady na záruku, zvýši produktivitu a zlepší zákaznícke služby.

IoT v domácnosti. Domáce aplikácie IoT zahŕňajú automatizáciu domácich prác a správu energie, ako aj zabezpečenie a ochranu. Tieto aplikácie poskytujú priame výhody používateľom a tiež ďalším zainteresovaným stranám, ako sú napríklad komunálne služby (Tsiatsis, 2018).

IoT v energetike. Merače, ktoré samy prenášajú údaje a zobrazujú spotrebu každého elektrického spotrebiča, sú prospešné pre poskytovateľa služieb aj pre spotrebiteľa. Kúrenie, ktoré sa zapne hodinu pred príchodom domov, šetrí až 50% peniazí používateľa. Niektoré riešenia pre inteligentné domácnosti dokonca ukazujú, koľko konkrétna žiarovka alebo zariadenie pre domácnosť spotrebuje elektriny v danú minútu. To všetko robí život človeka nielen pohodlnejším, ale umožňuje aj hospodárne využívať energetické zdroje.

IoT v zdravotníctve. Špeciálne senzory a inteligentné zariadenia budú monitorovať stav pacientov v reálnom čase. Získané údaje umožňujú IoT pracovať s umelou inteligenciou na presnú diagnostiku a plánovanie liečby, zvyšovanie bezpečnosti z výsledkov pacientov a optimalizujú poskytovanie zdravotnej starostlivosti. Nové technológie budú predchádzať infarkt, porážke a vznik krvných zrazenín.

IoT v logistike. Logistický priemysel je jedným z kľúčových priemyselných odvetví, ktoré budú mať prospech z vývoja a funkčnosti IoT. Zavedenie prenosných skenerov, ktoré digitalizovali proces doručenia. Implementácia systému senzorov, ktoré monitorovali integritu tovaru a výkon dodávky.

1.2 Získavanie biometrických údajov

1.2.1 Telesná teplota

Meranie telesnej teploty je dôležité pre stanovenie možnej odchýlky od normy. Zvýšenie teploty naznačuje bolestivé procesy v tele. Preto regulácia teploty umožňuje zistiť chorobu v počiatočných štádiách. Hodnota teploty závisí od miesta merania a typu teplomeru. Za normálne hodnoty teploty sa považujú nasledujúce:

- 35,3-37,8 °C – pre rektálnu teplotu.
- 36,0-37,4 °C – pre orálnu teplotu.
- 36,0-37,8 °C – pre teplotu v uchu.
- 35,8-37,6 °C – pre teplotu na čele.

Väčšina metód merania teploty vyžaduje určitý druh fyzického kontaktu teplotného snímača s predmetom, ktorého teplota sa má merať. Ale s pokrokom v technológiách sa objavujú aj nové spôsoby merania teploty. Preto, ak je teraz potreba merať teplotu objektu bez fyzického kontaktu s ním, dá sa na to použiť infračervený teplomer. Princíp činnosti infračervených teplomerov je jednoduchý - všetky telesá pri teplotách nad 0° Kelvina (absolútna nula) vyžarujú infračervené žiarenie, ktorú dokáže snímať snímač infračerveného teplomeru. Konštrukcia infračerveného teplomeru má optický systém, ktorý zameriava infračervené žiarenie emitované objektom. Infračervený snímač potom prevádza žiarenie na elektrický signál, ktorý sa potom môže prenášať do mikrokontroléra na interpretáciu a zobrazenie v jednotkách teploty.

Obr.1 Príklad infračerveného teplomeru[1]

Výhody merania telesnej teploty pomocou infračerveného teplomeru:

1. Bezpečnosť. Teplomer neobsahuje sklo a ortuť a nie je nositeľom infekcií. Preto možno zariadenie nazvať najbezpečnejším.
2. Ľahké použitie v každodennom živote.
3. Môže byť použitý na bodové meranie a nepretržité skenovanie povrchu veľkých objektov.
4. Čas merania. Infračervený teplomer zobrazí používateľovi výsledok merania za pár sekúnd.
5. Široký teplotný rozsah, ktorý je možné merať. To umožní používateľovi merať nielen teplotu ľudského tela, ale aj teplotu iných predmetov (jedlo, nápoje, voda vo vani, vzduch v miestnosti, zviera).
6. Presnosť údajov. Chyba pri meraní teploty bezkontaktným infračerveným teplomerom je až 0,5 °C (v priemere asi 0,2-0,3). Okrem toho sa telesná teplota človeka môže zmeniť o 1 stupeň za pár minút, je to prirodzený proces.

1.2.2 Tep srdca

Pulz sú rytmické vibrácie stien krvných ciev, ktoré sa vyskytujú pri kontrakciách srdca. Meranie srdcového rytmu je veľmi dôležité pre diagnostiku kardiovaskulárnych chorôb. Je dôležité sledovať zmeny srdcového rytmu, aby nedošlo k preťaženiu tela, najmä pri športe. Jedným z pochopiteľných parametrov srdcového rytmu je frekvencia pulzu.

[1] Zdroj: https://i8.rozetka.ua/goods/19321123/222218803_images_19321123495.jpg

Merané v počte tepov za minútu. Srdcová frekvencia má širokú škálu, pretože závisí od mnohých parametrov: vek, váha, pohybové návyky, dokonca aj denná doba. Preto sa napríklad ráno srdcová frekvencia zrýchľuje a v noci sa naopak spomaľuje.

Srdcová frekvencia predstavuje čistý účinok parasympatických nervov, ktoré spomaľujú srdcovú frekvenciu, a sympatických nervov, ktoré ju urýchľujú (Dzedzickis, Kaklauskas a Bucinskas, 2020). V závislosti od frekvencie, sa pulz rozlišuje nasledovne:

- mierna frekvencia — 60-90 úderov za minútu.
- zriedkavá frekvencia (*pulsus rarus*) — menej 60 úderov za minútu.
- častá frekvencia (*pulsus frequens*) — viac ako 90 úderov za minútu.

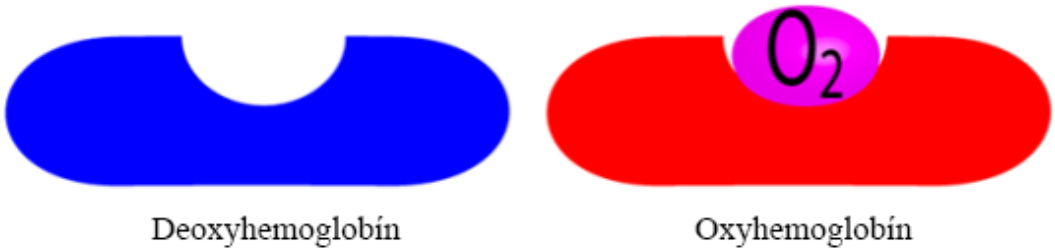
U dospelého človeka sa za normu považuje pulz v rozmedzí 60 - 100 úderov za minútu. Vzhľadom na to, že srdcová frekvencia a telesná teplota sú najdôležitejšími ukazovateľmi ľudského zdravia, mal by človek tieto ukazovatele pravidelne kontrolovať. A ak náhle ukazovatele prekročia normálny rozsah, mal by sa poradiť s lekárom.

Existuje niekoľko spôsobov, ako merať srdcovú frekvenciu. Pulz je možné merať manuálne. K tomu je potrebné použiť stopky aby označiť časové obdobie a zmerať počet arteriálnych pulzácií počas tohto obdobia. Ale táto metóda je nepohodlná. Pulz je možné merať aj pomocou špeciálnych prístrojov: pulzný oxymeter, monitor srdcového tepu, inteligentné hodinky a športové sledovače. Najbežnejším spôsobom merania srdcovej frekvencie pomocou nositeľných zariadení je optická alebo fotopletysmografia. Optická pletysmografia je založená na meraní pulzu človeka pomocou senzora a špeciálnych LED diód. Optický snímač na zadnej strane prístroja vyžaruje svetlo na zápästie pomocou diód LED, ktoré je absorbované telesnými tkanivami vrátane krvi, zatiaľ čo optický snímač skúma svetlo, ktoré sa odráža späť. Pritom krv absorbuje viac svetla ako napríklad pokožka. Zmena množstva krvi v cievach ovplyvňuje hladinu absorpcie svetla, ktorú zaznamenáva senzor. Špeciálny algoritmus na základe prijatých údajov určuje srdcovú frekvenciu.

1.2.3 Saturácia krvi kyslíkom

Bunky tela potrebujú energiu na pohyb, syntézu bielkovín a tvorbu ďalších chemikálií. To je základ každého života. Krv je zo 40% tvorená bunkami, zvyšných 60% tvorí plazma (ľahká tekutina vyrobená z vody, solí a minerálov). Prevažná väčšina buniek sú erytrocyty, ktoré sa tiež nazývajú červené krvinky. Tvoria 99% všetkých krvných buniek (približne 20 - 25 miliárd kusov). Každý takýto erytrocyt obsahuje viac ako 250

miliónov molekúl hemoglobínu. Hemoglobín je bielkovina, ktorá prenáša kyslík krvou. Nachádza sa vo vnútri červených krviniek a dodáva im červenú farbu. Je to táto molekula, ktorá je schopná viazať sa s kyslíkom a preniesť ho do všetkých tkanív tela. Jedna molekula hemoglobínu môže niesť 4 molekuly kyslíka a v takom prípade bude 100% nasýtená. Hemoglobín, ktorý k sebe pripojil kyslík, sa mení na oxyhemoglobín (OxHb) a ten, ktorý sa odpojil kyslík má názov deoxyhemoglobín (HHb).



Obr.2 OxHb a HHb[2]

Saturácia krvi kyslíkom (SpO2) je to percento okysličeného hemoglobínu v krvi. Toto je dôležitý ukazovateľ stavu ľudského dýchacieho systému. Konkrétnejšie ide o percento okysličeného hemoglobínu v pomere k celkovému množstvu hemoglobínu v krvi. SpO2 je odhad arteriálnej saturácie kyslíkom alebo SaO2, ktorý sa vzťahuje na množstvo okysličeného hemoglobínu v krvi.

Pôvodné sa saturácia krvi kyslíkom merala invazívne, to znamenalo penetráciu tkaniva. V súčasnosti je možné SpO2 merať pomocou pulznej oxymetrie, nepriamej neinvazívnej metódy (to znamená, že nie je potrebná penetrácia tkaniva). Funguje to tak, že oxymeter vyžaruje a absorbuje svetelné vlny, ktoré prechádzajú krvnými cievami (alebo kapilárami) v prste. Variant svetelnej vlny prechádzajúcej prstom poskytuje nameranú hodnotu SpO2, pretože stupeň nasýtenia kyslíkom spôsobuje zmeny farby krvi. Táto hodnota je uvedená v percentách. Ak je váš oximeter 98%, znamená to, že každá červená krvinka je z 98% okysličená a 2% z okysličeného hemoglobínu. Pohybuje sa od 95 do 100% normálnej hodnoty SpO2.

Dobré okysličenie krvi je nevyhnutné na zabezpečenie energie potrebnej pre svaly, ktorá sa zvyšuje keď človek športuje. Hladina SpO2 môže mierne kolísať a klesnúť pod normál, čo sa deje pri fyzickej aktivite a je to úplne normálne. Ak je hodnota SpO2 u človeka nižšia ako 95%, môže to byť príznakom zlého okysličenia krvi alebo hypoxie. Počas hypoxie tkanivá strácajú jasne červenú farbu, ako pri dobrom okysličovaní, a stávajú sa tmavočervenými alebo modrými.

[2] Zdroj: https://docplayer.ru/docs-images/84/90513396/images/13-0.jpg

Existujú dva typy hypoxémie. Akútna hypoxémia sa vyskytuje počas relatívne krátkeho časového obdobia, zatiaľ čo chronická hypoxémia trvá dlhšie. Príznaky akútnej hypoxémie sa môžu líšiť od chronickej hypoxémie, ale bežnými sú dýchavičnosť, kašeľ, zmätenosť, rýchly srdcový rytmus aj zmeny farby kože. Chronickú hypoxémiu je ťažké odhaliť, pretože telo je schopné kompenzovať pokles kyslíka v krvi. Medzi príznaky patrí pľúcna hypertenzia, pravostranné zlyhanie srdca a ďalšie. Ako sa píše na stránke consumer.huawei.com, vyššie riziko hypoxémie môžu mať:

- Ľudia, ktorí majú nabitý program: Nie každý si môže vychutnať luxus rovnováhy medzi pracovným a súkromným životom. Existuje veľa ľudí v rôznych odvetviach, ktorí sústavne pracujú. Bez adekvátneho odpočinku môže permanentný stres spôsobiť, že mozog spotrebúva viac kyslíka, než je jeho obsah v krvi.
- Ľudia, ktorí sa pohybujú vo vysokohorskom prostredí: Vysoké nadmorské výšky môžu spôsobiť nízku hladinu nasýtenia kyslíkom alebo desaturáciu krvi. Stáva sa to kvôli nízkemu atmosférickému tlaku vo vysokých nadmorských výškach. Ak teda chodíte na túry na miesta ako sú napríklad Machu Picchu v Peru alebo Lhasa v Tibete, každý nádych vám poskytne menej kyslíka, než vaše telo potrebuje.
- Starší ľudia: Starnutie srdca a pľúc vedie k nedostatočnému prísunu a príjmu kyslíka. Ľudské telo sa pri prenose kyslíka spolieha na krv. Ak sa zníži vekom objem krvi, zníži sa tiež koncentrácia kyslíka.
- Ľudia, ktorí chrápu: Chrápanie môže spôsobiť krátkodobú alebo dlhodobú dýchaciu obštrukciu, ktorá môže viesť k hypoxii, ktorá vedie k zlej kvalite spánku a môže viesť dokonca k poškodeniu orgánov.

Pulzná oximetria sa zvyčajne meria z prsta alebo ušného lalôčika. Tieto časti tela sú dobre spojené s venóznou krvou, a preto je objem krvných častí vysoký.

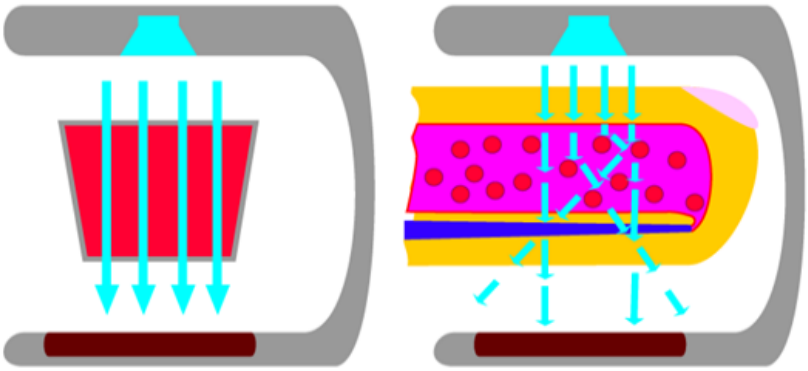
Množstvo svetla absorbovaného prstom závisí od mnohých fyzikálnych vlastností a tieto vlastnosti ovplyvňujú výpočet nasýtenia kyslíkom. Množstvo absorbovaného svetla závisí od:

- Koncentrácie látky absorbujúcej svetlo.
- Dĺžky dráhy svetla v absorbujúcej látke.
- OxHb (oxyhemoglobín) a HHb (deoxyhemoglobín) absorbujú červené a infračervené svetlo rôznymi spôsobmi.

Pri výpočte nasýtenia kyslíkom môžu nastať niektoré problémy. Vo fyzike má Lambertov zákon veľmi prísne kritériá. Napríklad svetlo prechádzajúce vzorkou by malo cestovať rovno ako lúče svetla.

Obr.3 Lúče, ktoré prechádzajú podľa zákona[3]

Ale v reálnom živote sa tak nestane. Krv nie je čisto červená tekutina. Namiesto toho je naplnená rôznymi nepravidelnými predmetmi, ako sú červené krvinky atď. To spôsobí, že sa svetlo rozptýli namiesto toho, aby išlo priamo.



Obr.4 Prechod lúčov v skutočnosti[4]

1.2.4 Elektrokardiografia (EKG)

Elektrokardiografia je metóda registrácie a výskumu elektrických polí generovaných pri práci srdca. Je pomerne lacná, ale cenná metóda elektrofyziologickej inštrumentálnej diagnostiky v kardiológii. Priamym výsledkom elektrokardiografie je elektrokardiogram (EKG) - grafické znázornenie rozdielu potenciálov vznikajúcich z práce srdca a vyvedeného na povrch tela.

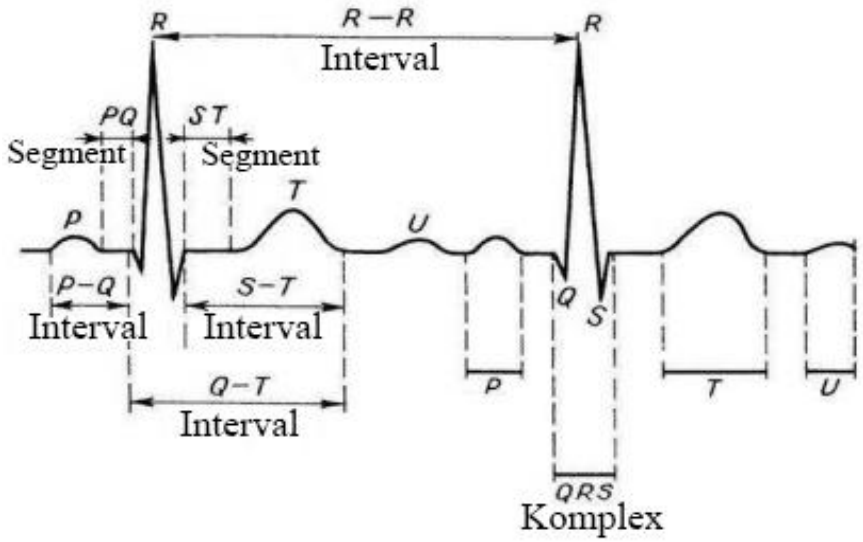
Elektrická aktivita je spojená s impulzmi, ktoré prechádzajú srdcom a určujú frekvenciu a rytmus pulzu. Môže poskytnúť presný odhad srdcového rytmu, intervalu R-R a variability srdcového rytmu. Zmeny srdcového rytmu sa používali ako indikátory rôznych emócii (Zong a Chetouani, 2009).

[3] Zdroj: https://docplayer.ru/docs-images/84/90513396/images/18-0.jpg
[4] Zdroj: https://docplayer.ru/docs-images/84/90513396/images/18-1.jpg

Po prvé, signál EKG je výsledkom činnosti srdca, ktoré má nervové zakončenia z autonómneho nervového systému, ktorý riadi správanie každej emócie (Bexton, Vallin, Camm, 1986). Po druhé, snímače EKG možno použiť ako nositeľné zariadenia (Nemati, 2012). Nakoniec má väčšiu amplitude v porovnaní s inými biosenzormi (Rattanyu, Mizukawa, 2011). Hlavné výhody EKG sú:

- Vysoký informačný obsah a spoľahlivosť.
- Bezbolestnosť a bezpečnosť.
- Rýchlosť prebiehajúcich výskumov.
- Žiadne kontraindikácie.

Elektrokardiosignál, alebo krátke označenie ECS je signál, ktorý je prejavom kontraktilnej aktivity srdca a dá sa ľahko zaznamenať pomocou povrchových elektród umiestnených na končatinách, alebo hrudníku. ECS je možno najrozšírenejším, najuznávanejším a najpoužívanejším biomedicínskym signálom. Typický ľudský elektrokardiosignál, znázornený na obrázku 5, pozostáva z piatich pozitívnych a negatívnych kmitov - zubov zodpovedajúcich cyklu srdcovej činnosti. Tieto vrcholy sú označené latinskými písmenami P, Q, R, S, T, a hrudné zvody (perikardiálne) – V (V1, V2 V3, V4, V5, V6). Najdôležitejšie body signálu EKG sú vrcholy: P, Q, R, S, T a U. Tri vrcholy (P, R, T) smerujú nahor (pozitívne vrcholy) a dva (Q, S) smerujú dole (negatívne vrcholy). Amplitúda píkov sa meria v milivoltoch (mV), 1 mV zodpovedá odchýlke 1 centimetru od izoelektrickej čiary. Šírka vrcholov a trvanie intervalov sa merajú v sekundách.



Obr.5 Príklad signálu a zobrazenie hlavných parametrov EKG[5]

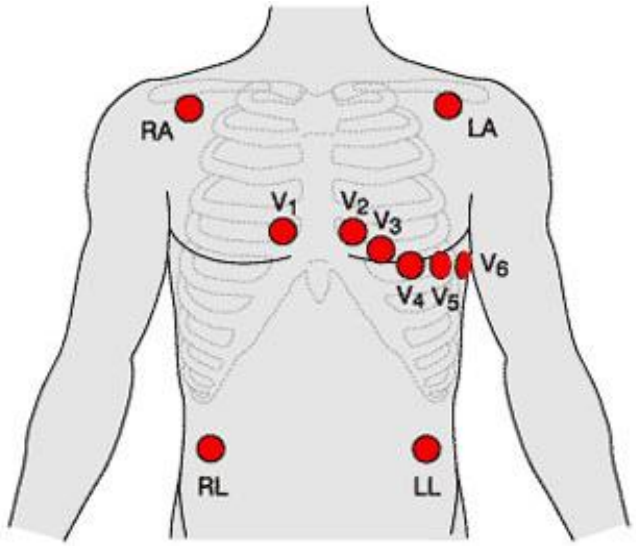
[5] Zdroj: Vlastný zdroj

V nasledujúcej tabuľke sú uvedené hlavné parametre, ktoré sa často používajú na hodnotenie signálov EKG.

Parameter	**Trvanie**	**Amplitúda mV**	**Popis**
P	0,06 - 0,1 sec.	0,05 - 0,25	Parameter odráža periódu impulzu predsiene.
PR	0,12 - 0,20 sec.	-	Zodpovedá času prechodu impulzu predsieňami a atrioventrikulárnym spojom do komorového myokardu. Jeho dĺžka sa líši v závislosti od veku, telesnej hmotnosti a frekvencie rytmu.
QRS	0,06 - 0,1 sec.	-	Zaznamenáva sa to počas ventrikulárneho impulzu, jeho trvanie odráža intraventrikulárne vedenie. Amplitúda komplexných vrcholov sa môže výrazne líšiť a závisí od polohy srdca v hrudníku, elektród.
QT	0,24 - 0,4 sec.	-	Odráža proces impulzu komôr. Jeho trvanie závisí od srdcovej frekvencie: keď sa rytmus zvýši, skráti sa, a keď sa spomalí, predĺži sa.

Tabuľka 1: Hlavné parametre elektrokardiogramu a ich popis

Pozície troch hlavných snímačov sú rozmiestnené na ľavom ramene (LA), pravom ramene (RA) a na ľavej nohe (LL). Pravá noha (RL) je spojená iba drôtom, ktorý by sa mal použiť ako zem pre vzájomne prepojené snímače. Iba s týmito tromi senzormi môžu lekári použiť metódu zvanú 3-zvodové EKG, ktorá trpí nedostatkom informácií o niektorých častiach srdca, ale je užitočná v prípade núdzových situáciách vyžadujúcich rýchlu analýzu (Dzedzickis, Kaklauskas a Bucinskas, 2020).



Obr.6 Správne umiestnenie elektród[6]

[6] Zdroj: https://www.wikilectures.eu/images/b/b4/Ecg_placement.jpeg

Tak pomocou EKG sa identifikujú choroby ako napríklad:

- Arytmia.
- Tachykardia.
- Bradykardia.
- Infarkt myokardu.
- Ischemická choroba srdca.
- Hypokaliémia.
- Pľúcna embólia.
- Dystrofické zmeny v myokarde atď.

1.2.5 Galvanická odozva kože (GSR)

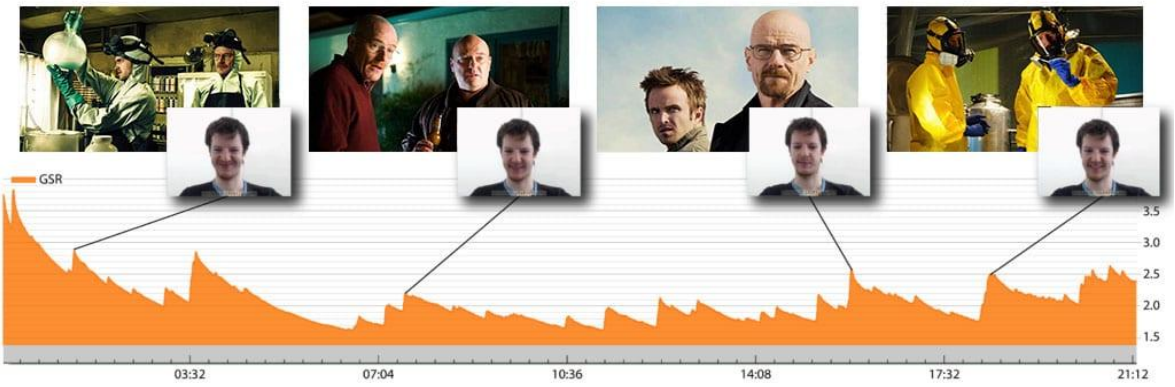
Galvanická reakcia kože, alebo krátke označenie GSR, známa tiež ako elektrodermálna aktivita (EDA) alebo vodivosť pokožky (SC), je kontinuálne meranie elektrických parametrov ľudskej pokožky. Najčastejšie táto technika ako hlavný parameter používa kožné vedenie (Dzedzickis, Kaklauskas a Bucinskas, 2020).

Vznik elektrickej aktivity kože je spôsobený hlavne aktivitou potných žliaz v ľudskej koži, ktoré sú zase pod kontrolou sympatického nervového systému (SNS). Mnoho empirických štúdií ukázali, že elektrická aktivita kože je vysoko reaktívny psychofyziologický indikátor, ktorý môže odlišne odrážať rôzne stupne emočného vzrušenia, vrátane rôznych typov duševnej činnosti.

Galvanická odozva kože sa považuje za súčasť orientačného reflexu, obranných, emocionálnych a iných reakcií tela spojených so sympatickou inerváciou, mobilizáciou adaptívno-trofických zdrojov a je výsledkom činnosti potných žliaz.

GSR sa široko používa na štúdium činnosti autonómneho nervového systému, stanovenie charakteristík psychofyziologických reakcií a štúdium osobnostných vlastností. GSR sa široko používa v psychofyziologických, fyziologických, klinických a fyziologických štúdiách ako vysoko citlivý, jednoduchý a technicky ľahko určený indikátor úrovne aktivity sympatického nervového systému, ako aj na hodnotenie neuropsychiatrického stresu človeka.

Pretože signál GSR obsahuje užitočné informácie týkajúce sa jeho amplitúdy a frekvencie, zvyčajne sa analyzuje v časovej a frekvenčnej oblasti pomocou rôznych metód a extrakcie štatistických parametrov, ako sú: medián, priemer, štandardná odchýlka, minimum, maximum a tiež pomer minima a maxima (Udovičić et al., 2017).



Obr.7 Príklad signálu GSR a jeho rozdiely pre rôzne ľudské emócie[7]

Meranie a štúdium galvanickej odozvy pokožky (GSR) sa začalo prvýkrát na konci 19. storočia, keď francúzsky lekár Fere zaregistroval zmenu odporu kože, keď cez ňu prešiel slabý prúd. Ruský fyziológ Tarkhanov zaznamenal rozdiel potenciálov medzi rôznymi oblasťami kože. Všetky tieto objavy sa stali takmer súčasne. Tieto objavy vytvorili základ pre dve metódy registrácie GSR: exosomatickú (meranie odporu kože) a endosomatickú (meranie elektrických potenciálov samotnej pokožky).

Elektródy na registráciu GSR sú zvyčajne dva snímače, ktoré sú pripevnené k ruke alebo prstom. V súčasnej dobe, aby sa zabránilo polarizácii, t.j. vzniku elektrického náboja na povrchu elektródy, sú rozšírené elektródy s povlakom chloridu strieborného, aj keď takmer všetky firmy vyrábajúce polygrafy postupne prechádzajú na pochrómované oceľové alebo mosadzné elektródy. Na meranie stavu odporu kože a jej elektrickej vodivosti sa používajú hlavne končeky prstov, pretože to umožňuje zhrnutie činnosti dvoch bodov. GSR sa zvyčajne zaznamenáva z prvého článku druhého a štvrtého prsta pravej ruky.

GSR je jedným z najcitlivejších, najefektívnejších a ľahko zaznamenateľných ukazovateľov emočného napätia, ktorý sa používa na inštrumentálnu diagnostiku emočného napätia. V mnohých výskumoch sa zistilo, že emočný stres spôsobuje významné zmeny krvného tlaku a srdcovej frekvencie.

[7] Zdroj: https://media.imotions.com/images/20190411124918/GSR-response-a.jpg

2 CIELE DIPLOMOVEJ PRÁCE

Cieľom diplomovej práce je navrhnúť a vytvoriť pomocou dostupných smart zariadení (Smart watch) systém pre identifikovanie a klasifikáciu emocionálneho stavu používateľa pomocou dostupných fyziologických funkcií. Naším cieľom je vytvorenie systému na monitorovanie základných životných funkcií pri zachovaní neinvazívnosti, ktorého súčasťou je používateľské rozhranie na báze IoT. Navrhneme vhodný postup na meranie fyziologických funkcií pomocou dostupných smart zariadení. Súčasťou bude webová, prípadne mobilná aplikácia na vizualizáciu a vyhodnocovanie získaných dát.

Podciele:

- oboznámiť sa s technológiou IoT,
- vybrať správne snímače a moduly potrebné pre splnenie zadania,
- oboznámiť sa s doskami Raspberry Pi a Arduino,
- navrhnúť vhodný postup na meranie fyziologických funkcií človeka pomocou mini počítačov a vybraných senzorov,
- vytvoriť správnu schému zapojenia,
- vytvoriť funkčný systém pre meranie,
- vytvoriť webové rozhranie pre vizualizáciu dát,
- vytvoriť aplikáciu pre mobilný operačný systém Android na vizualizáciu údajov,
- otestovať systém, navrhované aplikácie a vyhodnotiť získané dáta.

3 METODIKA PRÁCE

Pri tvorbe projektu sme sa zamerali na 3 hlavné úlohy. Prvou úlohou je výber vhodných modulov a mini počítačov. Senzory a moduly sú potrebné na presné meranie životných funkcií človeka. Mini počítač je nutný pre správne spracovanie informácií získaných zo senzorov a modulov. Druhou úlohou je správne spojenie a spájkovanie všetkého do jedného fungujúceho systému, ktorý po prijatí údajov ich odošle do databázy. A poslednou treťou úlohou je uložené dáta prijaté zo senzorov a modulov do databázy zobraziť následnou vizualizáciou vo webovej aplikácii a aplikácii vyvinutej pre mobilný operačný systém Android.

Ako mini počítače boli vybrané Raspberry Pi 4B a Arduino Nano. Raspberry Pi 4B bol programovaný prostredníctvom štandardného editora kódu IDLE, v ktorom bol programový kód napísaný v programovacom jazyku Python. Doska Arduino Nano bola naprogramovaná prostredníctvom IDE Arduino, v ktorom bol program vyvinutý v programovacom jazyku, ktorý je navrhnutý špeciálne pre Arduino a je založený na C/C++.

Na ukladanie dát sme použili databázu, ktorá je uložená v cloude - Firebase Realtime od Google. Pri vytváraní webovej aplikácie na písanie kódu bolo použité vývojové prostredie PyCharm. Pri vytváraní mobilnej aplikácie na písanie kódu bol použitý program Android Studio. Kód na vývoj webovej aplikácie bol tvorený kombináciou programovacích jazykov a skriptov ako Python, HTML, CSS, JavaScript. Kód na vývoj mobilnej aplikácie bol tvorený pomocou programovacieho jazyka Java.

Dáta, ktoré doska Arduino Nano prijala z modulov, ktoré sú k nej pripojené, boli zaslané do mini počítača Raspberry Pi 4B cez USB kábel. Ďalej boli údaje spracované a odoslané do databázy Firebase Realtime Database s následnou vizualizáciou vo webovej aplikácii a mobilnej aplikácii.

4 VLASTNÁ PRÁCA

Po analýze požiadaviek na projekt je potrebné zvoliť príslušné komponenty a vývojové prostredie, ktoré tvoria softvérový a hardvérový komplex.

Ďalšia kapitola popisuje komponenty, ktoré boli použité na vytvorenie projektu. Špecifikácia vybranej databázy na ukladanie informácií získaných zo senzorov a popis technológií použitých na vytvorenie webovej aplikácie a mobilnej aplikácie. Diagramy, ktoré popisujú princíp činnosti snímačov, schému zapojenia týchto snímačov a mini počítačov do jedného pracovného systému. Nasleduje popis postupného vytvárania softvérovej časti projektu, konkrétne programovania Arduina, Raspbrerry a vytvárania webovej a mobilnej aplikácie.

4.1 ŠPECIFIKÁCIA HARDWAROVEJ ČASTI SYSTÉMU

Táto podkapitola popisuje, ako sme začali s vytváraním hardvéru pre vyvíjaný projekt. Podkapitola popisuje komponenty, ktoré boli použité a funkčnosť, ktorú tieto komponenty vykonávajú.

Pre úspešnú implementáciu našich cieľov sme potrebovali použiť nasledujúce komponenty a technológie:

- Mini počítač Raspberry Pi 4.
- Modul EKG AD8232.
- Senzor pre snímanie srdcového rytmu a saturácie krvi kyslíkom MAX30102.
- Bezdotykový infračervený snímač teploty GY-906.
- Senzor pre snímanie galvanickej odozvy pokožky (GSR).
- Mini počítač Arduino Nano.

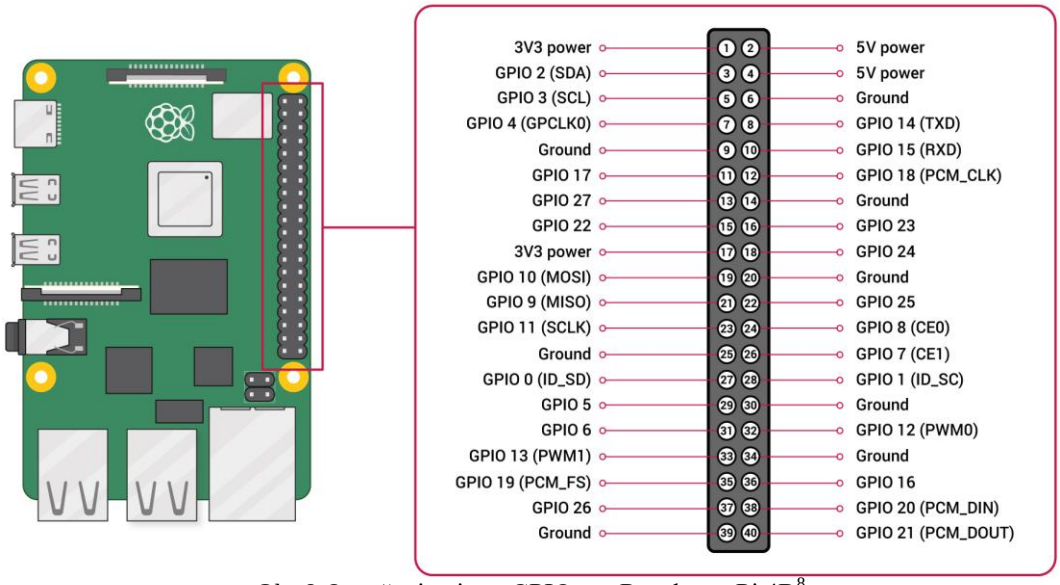
V ďalších podkapitolách sme predstavili hlavné parametre každého prvku, schémy zapojenia a ich charakteristiky.

4.1.1 Mini počítač Raspberry Pi

Raspberry Pi je lacný počítač, ktorý je približne taký veľký ako kreditná karta, ktorý sa pripája k počítačovému monitoru, alebo televízoru a môže používať štandardnú klávesnicu a myš (Abhishek L, Rishi Barath B, 2019). V priebehu času prešiel Raspberry Pi niekoľkými modifikáciami, z ktorých každá sa od predchádzajúcej verzie nejako líšila. Tento prístup umožnil upraviť cenu produktu v závislosti od potrieb používateľa, čo tiež pozitívne ovplyvnilo popularitu zariadenia. Celá rada Raspberry Pi využíva procesory

s architektúrou APM, ktorá sa osvedčila ako najlepšia. Hlavným poznávacím znakom Raspberry Pi od bežných počítačov je prítomnosť programovateľných vstupno-výstupných portov GPIO. Pomocou nich môže používateľ ovládať rôzne zariadenia a prijímať telemetriu z rôznych typov senzorov.

Označenia Pi 1, Pi 2, Pi 3 a Pi 4 predstavujú špecifickú generáciu modelu. Indexy A, A +, B a B + môžu byť spojené s charakteristikami zariadenia. Modely B, v rámci tej istej generácie, sú produktívnejšie a prichádzajú s bohatšou výbavou na rozdiel od modelu A. Model s indexom Plus (+) bol pridaný k modelom, ktoré mali v porovnaní s modelom bez tohto indexu veľmi malé zmeny, napríklad Pi 1B sa líši od Pi 1B+ prítomnosťou funkcie POE (Power over Ethernet).



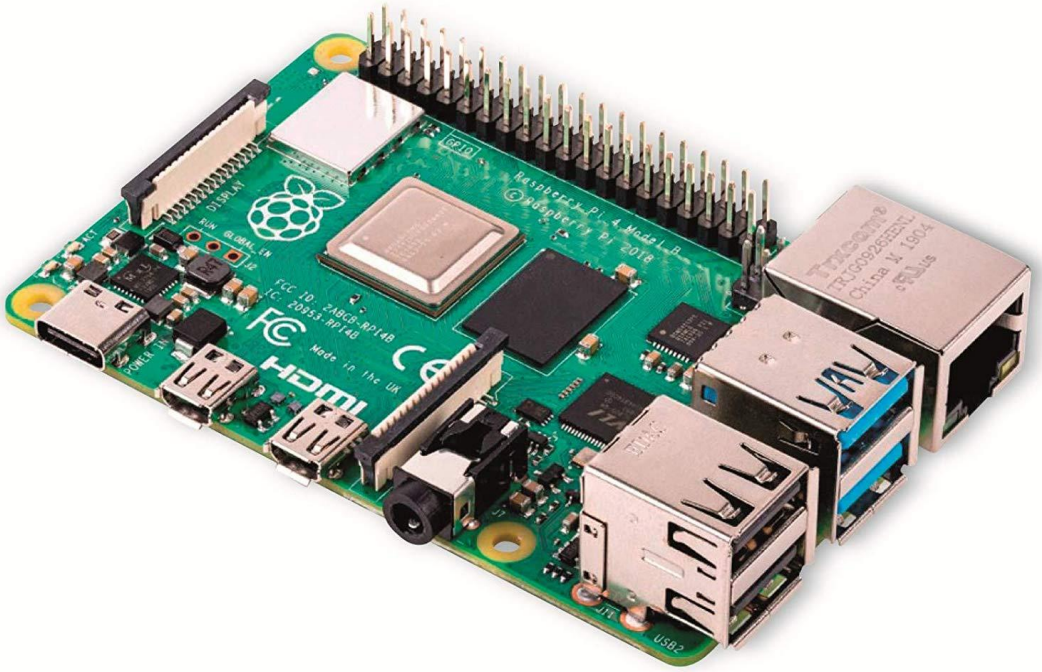
Obr.8 Označenie pinov GPIO pre Raspberry Pi 4B[8]

Na Raspberry Pi má používateľ na začiatku možnosť inštalovať si operačný systém Linux. Odporúčaným operačným systémom je Raspbian, ktorý je založený na Debiane. Prípadne Pidora, ktorý je založený na Fedore.

Názov Raspberry bol zvolený analogicky s názvami iných spoločností, ktoré používajú tiež názvy jedál. V preklade z angličtiny to znamená názov maliny. Časť mena Pi pochádza z pôvodného nápadu vyrobiť malý počítač pre prácu s jazykom Python.

Ako počítač na vytvorenie projektu, ktorý bude spracovávať údaje a odosielať ich do databázy, sme si vybrali model Pi 4B, ktorý má 4 gigabajty pamäte RAM. Generácia 4B sa začala predávať v júni 2019.

[8] Zdroj: https://www.raspberrypi.org/documentation/usage/gpio/images/GPIO-Pinout-Diagram-2.png



Obr.9 Vzhľad dosky Raspberry Pi 4B[9]

Existuje množstvo distribúcií Linuxu, ktoré podporujú Raspberry Pi. Každá distribúcia má svoje vlastné charakteristiky. Medzi nimi je vždy na prvom mieste distribúcia Raspbian. Raspbian je distribúcia Linuxu, ktorá je založená na Debiane. Raspbian je vyvíjaný samotnou Raspberry Pi Foundation a práve tento systém sa odporúča používať. Práve tento systém sa bude používať ako hlavný operačný systém na doske Raspberry Pi 4B. Proces inštalácie OS je veľmi jednoduchý a nevyžaduje žiadnu špeciálnu prípravu. Inštalácia vyžaduje iba kartu microSD s minimálnou kapacitou 8 GB.

Jedným z kľúčových rozdielov medzi Raspberry Pi 4 Model B a ostatnými modelmi je ten, že 4B má výkonnejší procesor - štvorjadrový Broadcom BCM2711 SoC. Pri 1,5 GHz je tento čip iba o 100 MHz rýchlejší ako štvorjadrový procesor Broadcom BCM2837 s frekvenciou 1,4 GHz používaný v modeli Pi 3 Model B+, rozdiel je však oveľa väčší. To možno vysvetliť tým, že model Pi 4 používa ARM V8 Cortex-A72 jadrá a nie starší Cortex-A53 ako používa model Pi 3 B+. Pi 4 používa jadrá ARM v8 Cortex-A72 a nie starší Cortex-A53, ktoré používa Pi 3 Model B+. Model Pi 4B je vybavený dodatočným grafickým procesorom VideoCore VI (OpenGL ES 3.x) a hardvérovým dekodérom 4Kp60 na prehrávanie videa HEVC. Dva porty microHDMI so schopnosťou prenášať signál až do 4K umožňujú pripojenie dvoch monitorov súčasne. Nasledujúca tabuľka popisuje hlavné charakteristiky modelu 4B:

[9] Zdroj: https://d5hu1uk9q8r1p.cloudfront.net/computer/skrar/vorur/9729-1.jpg

Procesor	Broadcom BCM2711 4 jadrá Cortex-A72 (ARMv8) 64-bit SoC @ 1.5 GHz
RAM	Existujú modely (LPDDR4-2400 SDRAM): • 1Gb. • 2Gb. • 4Gb. • 8Gb.
Napájanie	USB Type-C (5V, minimum 3A).
Porty a konektory	• 2 porty Micro-HDMI (simultánna podpora dvoch monitorov s rozlíšením 4K). • 2 porty USB 2.0. • 2 porty USB 3.0. • 1 port Gigabit Ethernet (RJ45, 1000Base-T). • 1 port microSD (pre operačný systém a ukladanie dát). Odporúča sa používať karty s kapacitou najmenej 8Gb. • 40-pinový GPIO (univerzálne vstupno/výstupné rozhranie). Porty sú plne kompatibilné s predchádzajúcimi doskami. • 3.5mm Audio a Composite Output Jack. • CSI Display Connector. • CSI Camera Connector. • Power over Ethernet (PoE).
Bezdrôtové rozhrania	Dvojpásmové WiFi (2.4 GHz a 5.0 GHz IEEE 802.11ac/n) Bluetooth 5.0, BLE (Bluetooth Low Energy)
Indikátory	Vstavané LED diódy: indikácia napájania, práca s kartou microSD, režim Ethernet.
Rozmery (šírka x výška x hĺbka)	87x21x56mm.

Tabuľka 2: Hlavné charakteristiky mini počítača Raspberry Pi 4 Model B

4.1.2 Senzor pre monitorovanie srdcového tepu a saturácie krvi kyslíkom MAX30102

Senzor MAX30102 je vysoko citlivý snímač srdcového rytmu a nasýtenie krvi kyslíkom. Zahŕňa interné LED diódy, fotodetektory, optické prvky a nízkošumovú elektroniku s potlačením okolitého svetla. Model MAX30102 poskytuje kompletné systémové riešenie na uľahčenie procesu navrhovania pre mobilné a nositeľné zariadenia. Model MAX30102 pracuje s jedným zdrojom napájania 1,8 V a samostatným zdrojom napájania 3,3 V pre interné LED diódy. Komunikácia sa uskutočňuje prostredníctvom štandardného rozhrania kompatibilného s I2C. Je široko používaný v prenosných zariadeniach, zariadeniach fitness asistentov a smartfónoch.

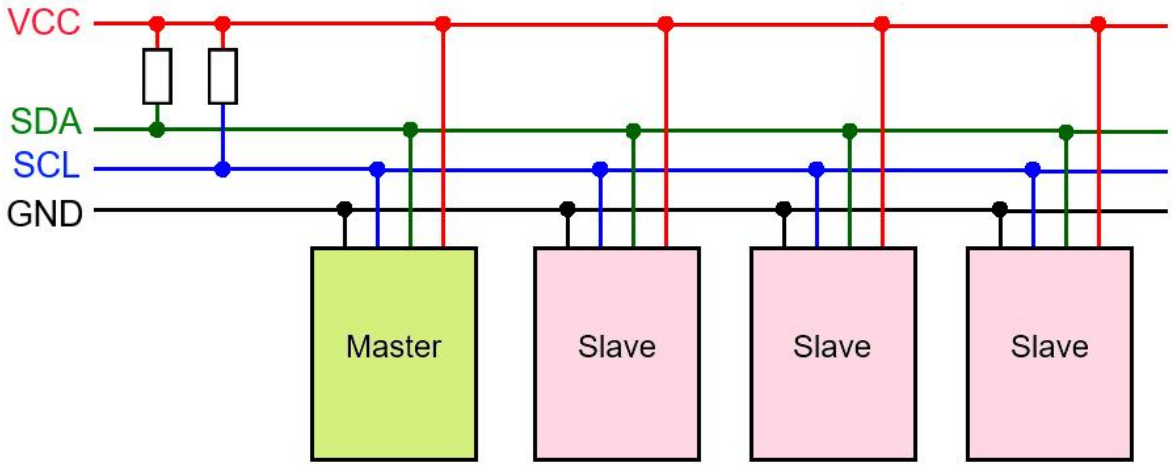
I2C je asymetrická sériová zbernica na komunikáciu medzi integrovanými obvodmi v elektronických zariadeniach. To znamená, že tento komunikačný protokol bol vyvinutý pre internú komunikáciu vo vnútri skrinky zariadenia. Sieť I2C sa skladá z hlavnej

a podradenej jednotky, ktoré sú spojené zbernicou. V sieti I2C môže byť niekoľko zariadení typu master a slave - majstri a nástupcovia:

1. Slave zariadenie (nástupca). Všetky slave zariadenia majú adresu I2C, ktorá sa používa na identifikáciu zariadenia v sieti. Adresa I2C umožňuje hlavnému zariadeniu prenášať údaje na konkrétneho slave po zbernici.
2. Hlavné zariadenie (majster). Majstri môžu odosielať a prijímať údaje. Slave zariadenia reagujú na to, čo pošle master. Pri odosielaní údajov cez zbernicu môže údaje odosielať naraz iba jedno zariadenie.

Zbernica I2C poskytuje viacokruhovú komunikáciu pomocou synchrónnej komunikácie, ktorá využíva piny sériovej synchronizačnej linky (SCL) a sériovej dátovej linky (SDA). Toto je bežný spôsob komunikácie s integrovanými obvodmi (Hoffman, 2018).

Každé podradené (slave) zariadenie má jedinečnú adresu, podľa ktorej majster komunikuje s podriadeným zariadením. Na jednu zbernicu I2C je možné pripojiť až 127 zariadení, vrátane niekoľkých masterov. K zbernici je možnosť pripojiť zariadenia počas práce, t.j. podporuje pripojenie "hot plug".



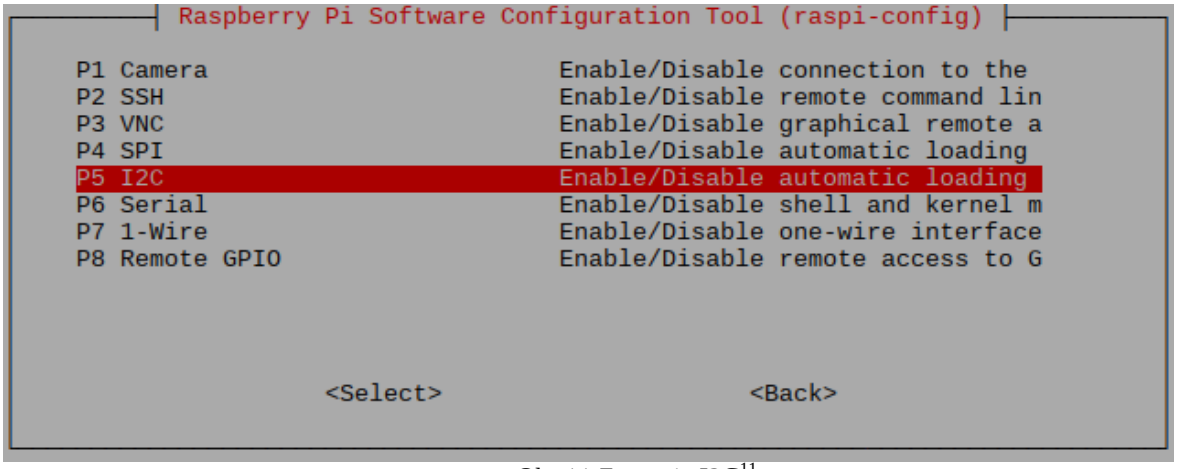
Obr.10 Popis rozhrania I2C[10]

Zbernica v I2C je jednoducho dva drôty, ktoré spájajú všetky I2C zariadenia v sieti. Tieto dva drôty sa nazývajú SDA a SCL. Drôt SDA sa používa na komunikáciu medzi zariadeniami typu master a slave. SCL prenáša taktovací signál používaný pre správnu synchronizáciu komunikácie. Udržiavať dva drôty v stave HIGH sú potrebné impulzné alebo pull-up odpory.

[10] Zdroj: https://soltau.ru/images/ardu-i2c/1.png

Pre prácu so zbernicami I2C a SPI na Raspberry je potrebné ich zapnúť (štandardne sú zakázané). Je to možné urobiť nasledovne:

1. Je potrebné otvoriť terminál a zadať príkaz z klávesnice *sudo raspi-config*.
2. Je nutné zvoliť piatu položku Interfacing Options a stlačiť klávesu Enter.
3. Otvorí sa menu pre výber rozhrania, je potrebné vybrať bod P5 I2C a stlačiť klávesu Enter.
4. Je nutné potvrdiť svoju voľbu odpovedaním na otázku vo vyskakovacom okne Yes.



Obr.11 Zapnutie I2C[11]

Zariadenie má miniatúrne rozmery, ktoré sa dosiahli bez straty optického alebo elektrického výkonu. Integrácia do plne funkčného nositeľného meracieho systému vyžaduje minimálne ďalšie externé komponenty.

Obr.12 Vzhľad snímača MAX30102[12]

Technické údaje snímača MAX30102:

- Špičková vlnová dĺžka LED: 660nm/880nm.
- LED napätie: 3,3-5V.
- Rozhranie: I2C rozhranie.

[11] Zdroj: Vlastný zdroj
[12] Zdroj: https://storage.googleapis.com/stateless-www-faranux-com/2019/09/a22388b2-max30102-module-500x500.jpg

- Rozsah pracovných teplôt: -40°C až +85°C.
- Veľkosť: 20.3x15.2mm.
- Hmotnosť: 1.1g.

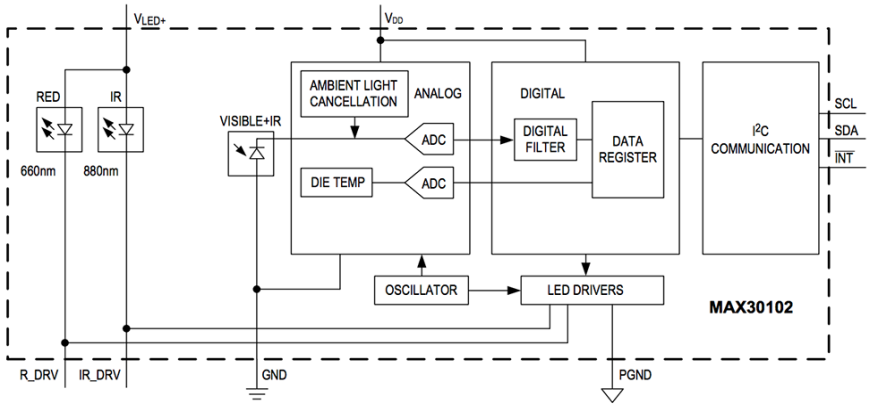
Pre pripojenie na doske senzora je 7 kontaktov, z ktorých je použitých iba 5 kontaktov:

Konektor	**Konektor na Raspberry Pi**	**Popis**
VIN	3v3 power (1)	Vstupný napájací kontakt
GND	Ground (6)	Uzemnenie
SDA	GPIO 2 SDA (3)	Kontakt pre prácu s rozhraním I2C ale ich účel sa môže líšiť na rôznych doskách.
SCL	GPIO 3 SDA (5)	Kontakt pre prácu s rozhraním I2C
INT	GPIO 4 GPCLK0 (7)	Konektor prerušenia

Tabuľka 3: Pripojenie snímača MAX30102

Model MAX30102 obsahuje červené a IR LED diódy poháňané internými ovládačmi, ktoré modulujú šírku impulzu a prúd pre meranie impulzu. Prúd je možné meniť softvérovo v rozsahu 0-50 mA a dobu impulzu je možné programovať v rozmedzí 69-411 µs. Pritom je možné optimalizovať presnosť merania a spotrebu energie pre konkrétnu situáciu.

Percento kyslíka v krvi sa v tomto prípade určuje neinvazívnou metódou cez kožu, ako percento okysličeného hemoglobínu (HbO2) k celkovému hemoglobínu (HbO2 + RHb), stanovené pomocou fotodetektora, IR a červenej LED MAX30102. Merací subsystém SpO2 obsahuje obvod kompenzácie okolitého svetla, analógovo-digitálny prevodník sigma-delta (ADC) a patentovaný digitálny filter. Kompenzácia okolitého svetla má vnútorný obvod blokujúci signál, ktorý eliminuje okolité svetlo a rozširuje efektívny dynamický rozsah. ADC je programovateľný v celom rozsahu merania 2 - 16 µA. Kompenzácia okolitého svetla umožňuje blokovať signál okolitého svetla až do 200 µA.

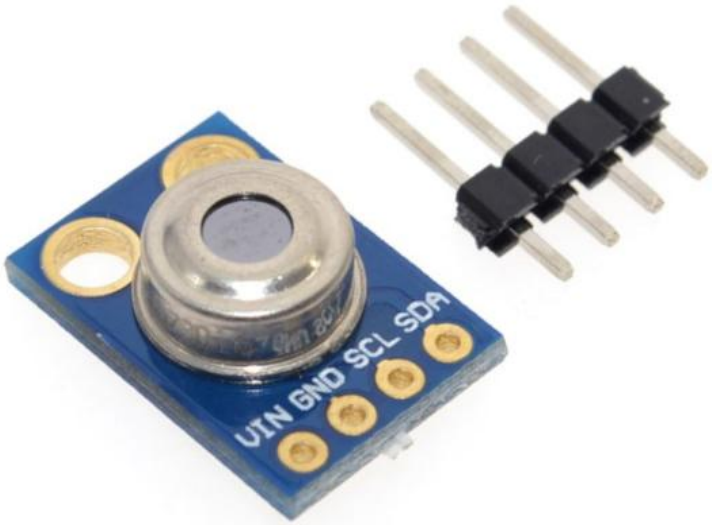


Obr.13 Funkčná bloková schéma snímača MAX30102[13]

[13] Zdroj: https://ru.mouser.com/images/marketingid/2016/microsites/149131390/Maxim-Max30102-Functional%20Diagram.png

4.1.3 Bezkontaktný Snímač Teploty GY-906

Modul GY-906 je bezkontaktný infračervený teplomer s rozsahom merania od -70 až do 380 stupňov Celzia na základe čipu Melexis MLX90614ESF-BAA. Modul má v podstate dva teplotné senzory zabudované v jednom kryte - prvý, v skutočnosti bezkontaktný infračervený lúč, meria teplotu objektov umiestnených oproti oknu v kryte a druhý meria teplotu samotného krytu.

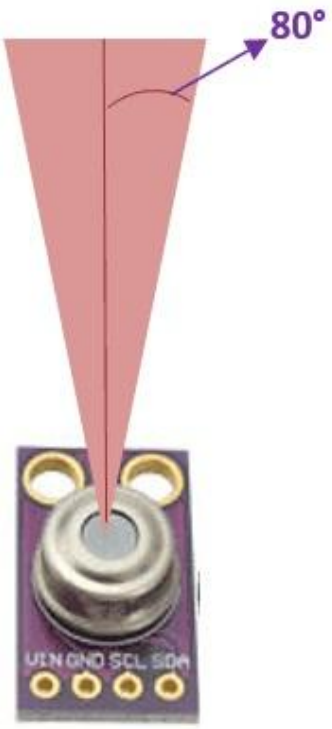


Obr.14 Vzhľad snímača GY-906[14]

Bezkontaktný snímač teploty je v mnohých prípadoch oveľa pohodlnejší ako bežný snímač, a to aj pri meraní teploty kvapalín a plynov, pohyblivých prvkov, pri meraní telesnej teploty v medicíne, pri meraní teploty predmetov, na ktoré je ťažké získať fyzický prístup atď. Okrem bezkontaktného merania ďalšou výhodou tohto snímača v porovnaní so štandardnými teplomermi je rýchlosť a niekde ešte vyššia presnosť merania.

Jedným z dôležitých faktorov merania je vzdialenosť počas merania medzi snímačom a objektom. Hodnota tejto vzdialenosti je daná výrazom Zorné pole (FOV), pre náš snímač je zorné pole približne 80°.

[14] Zdroj: https://www.mini-tech.com.ua/image/cache/catalog/sensors/gy-906_1-550x550.jpg



Obr.15 Zorné pole snímača GY-906[15]

Rozsah je v kónickom tvaru od bodu snímača, ako je to znázornené vyššie na obrázku číslo 15. Takým spôsobom, s rastúcou vzdialenosťou od meraného objektu sa zóna citlivosti zdvojnásobuje. To znamená, že na každý jeden centimeter vzdialený od objektu sa zóna citlivosti zvýši o 2 centimetre.

Princíp fungovania tohto modulu je veľmi jednoduchý. Spočiatku vysiela infračervený signál, ktorý keď narazí na objekt, umožňuje vypočítať množstvo elektromagnetickej energie z jeho povrchu (na základe emitovaného infračerveného žiarenia). Teplomer sa ovláda a sníma cez rozhranie kompatibilné s I2C, ktorého princíp bol opísaný v bode 4.1.2.

Hlavné vlastnosti bezkontaktného infračerveného snímača teploty GY-906 sú popísané nižšie v tabuľke číslo 4.

Použitý čip	MLX90614ESF-BAA
Napájacie napätie	3 alebo 5.5V
Výstupné rozhranie	I2C alebo PWM
Rozsah merania teploty (objekt)	-70...+380 °C
Rozsah merania teploty (prostredie)	-40...+85 °C
Presnosť snímania teploty	±0.5 °C
Rozmery dosky	17 x 13 x 6,3 mm

Tabuľka 4: Hlavné parametre infračerveného snímača teploty GY-906

[15] Zdroj: http://digitrode.ru/uploads/posts/2020-04/1587951089_arduino3.jpg

Pripojenie snímača GY-906 k mini počítaču Raspberry Pi 4B. Na doske senzora sú 4 kontakty, ktoré musia byť pripojené k mini počítaču, ako je to opísané nižšie v tabuľke číslo 5.

Konektor	**Konektor na Raspberry Pi**	**Popis**
VIN	5V power (2)	Vstupný napájací kontakt (3-5,5V)
GND	Ground (9)	Uzemnenie
SDA	GPIO 2 SDA (3)	Kontakt pre prácu s rozhraním I2C ale ich účel sa môže líšiť na rôznych doskách
SCL	GPIO 3 SDA (5)	Kontakt pre prácu s rozhraním I2C

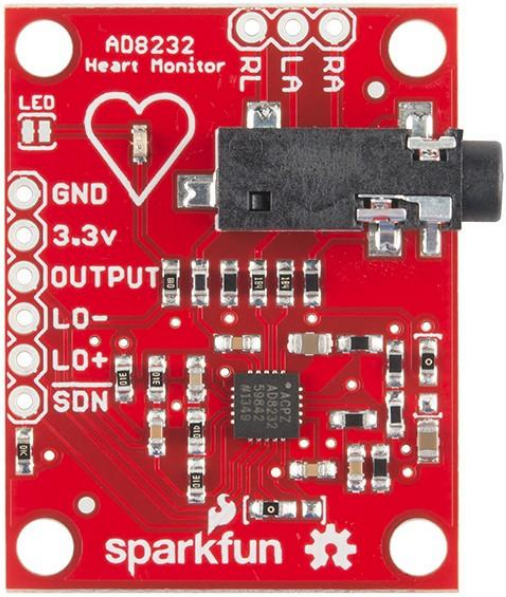
Tabuľka 5: Pripojenie infračerveného snímača teploty GY-906

4.1.4 Modul EKG AD8232

Modul EKG AD8232 je to ekonomický modul, ktorý sa používa na meranie elektrickej aktivity srdca. Táto elektrická aktivita môže byť identifikovaná ako elektrokardiogram (EKG) a zobrazená ako analógové hodnoty. AD8232 obsahuje bipolárny filter vysokých tónov a nespínaný operačný zosilňovač, ktorý umožňuje používať technológiu viacpólového dolnopriepustného filtrovania na odstránenie šumu a iného rušenia.

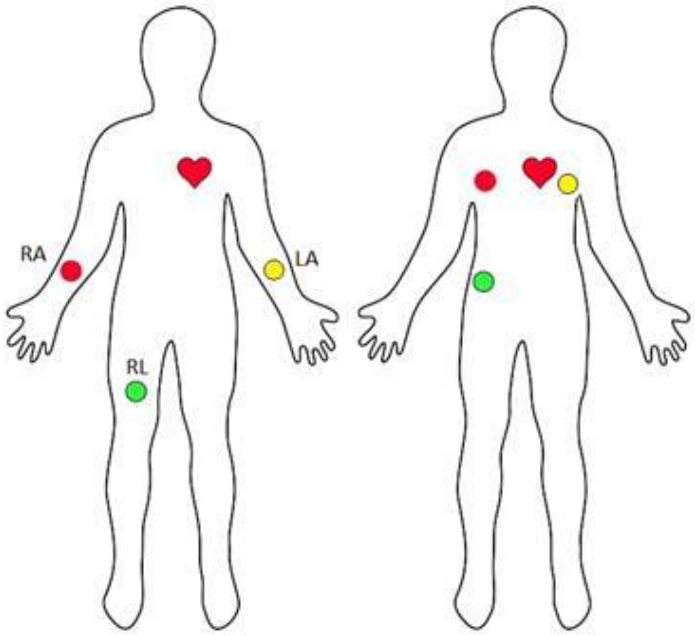
Tento modul sníma elektrickú aktivitu srdca pomocou metódy bipolárneho vedenia, v ktorej sa zaznamenáva potenciálny rozdiel medzi dvoma bodmi elektrického poľa. Tieto údaje možno použiť na sledovanie srdcových rytmov počas cvičenia a športu, ako aj na všeobecné sledovanie činnosti srdcového systému. Kľúčové špecifikácie snímača srdcovej frekvencie AD8232:

- Napájacie napätie: 3,3V.
- Typ výstupného signálu: analógový výstup.
- Výstupné rozhranie: 2,54 pinový alebo konektor pre slúchadlá.
- Veľkosť modulu: 36mm*31mm*18mm.
- Nominálny teplotný rozsah: 0°C až +70°C.
- Rozsah pracovných teplôt: -40°C až + 85°C.



Obr.16 Vzhľad snímača srdcovej frekvencie AD8232[16]

Na získanie kardiogramu sú elektródy pripevnené k hrudníku a končatinám, z ktorých sa odoberajú signály o elektrickej aktivite srdca. Elektródy umiestnené na tele pacienta detekujú malé zmeny potenciálu na koži, ktoré vznikajú v dôsledku depolarizácie srdcového svalu pri každej kontrakcii. Depolarizácia je prudká zmena elektrického stavu bunky, keď sa záporný vnútorný náboj bunky stane na krátky čas pozitívnym.



Obr.17 Schéma aplikácie elektród[17]

Táto doska tiež poskytuje linky RA (pravé rameno), LA (ľavé rameno) a RL (pravé rameno) pre pripojenie a použitie vlastných senzorov. Okrem toho má doska aj LED

[16] Zdroj: https://cdn.antratek.nl/media/product/157/single-lead-heart-rate-monitor-ad8232-sen-12650-31d.jpg
[17] Zdroj: https://cxem.net/medic/images/medic38-5.jpg

indikátor, ktorý zobrazuje pulzovú frekvenciu počas merania. Na pripojenie biomedicínskej senzorovej podložky je možné použiť 3,5 mm konektor alebo 3-pinový konektor.

Okrem toho táto doska je navyše vhodná aj na použitie a na získanie údajov o kontrakciách iných svalov, čo potenciálne umožňuje jej použitie v bionike a protetike. V tomto prípade je potrebné pripojiť elektródy k svalom, činnosť ktorých musí byť sledovaná.

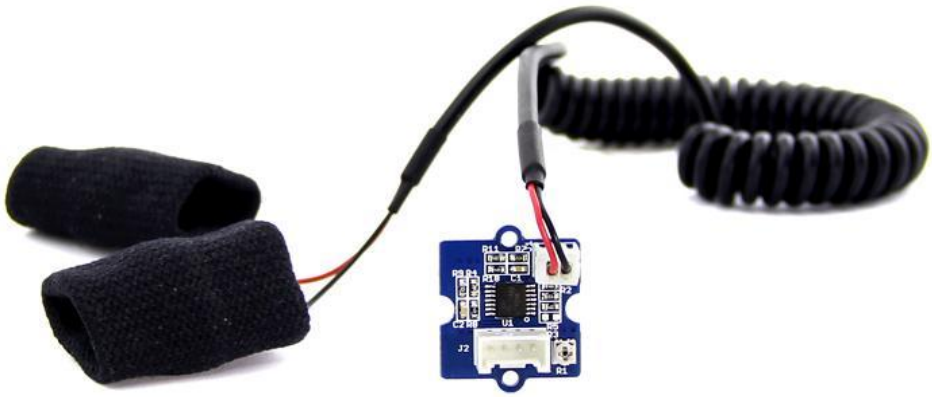
Pripojenie snímača srdcovej frekvencie AD8232 k mini počítaču Arduino Nano. Na doske modulu je 6 pinov, ale použitých je iba 5 z nich:

Konektor	**Konektor na Arduino Nano**	**Popis**
GND	GND	Uzemnenie
3,3V	3V3	Vstupný napájací kontakt (3-5,5V)
OUTPUT	A0	Zodpovedá za signál prijatý z elektród
L0-	D11	Detektor kontaktov elektród (-)
L0+	D10	Detektor kontaktov elektród (+)
SDN	Tento kontakt sa nepoužíva	-

Tabuľka 6: Pripojenie snímača srdcovej frekvencie AD8232

4.1.5 Modul GSR

Modul GSR poskytuje schopnosť detekovať také silné emócie jednoduchým pripojením dvoch elektród k dvom prstom jednej ruky, čo je zaujímavý mechanizmus na vytváranie projektov súvisiacich s emóciami, ako je napríklad monitor kvality spánku.



Obr.18 Vzhľad modula GSR[18]

[18] Zdroj: https://seeeddoc.github.io/Grove-GSR_Sensor/img/GSR.jpg

Hlavné charakteristiky modulu GSR:

- Napájacie napätie: 3.3V/5V.
- Citlivosť: Nastaviteľná pomocou potenciometra.
- Vstupný signál: Odpor, nie vodivosť.
- Výstupný signál: napätie, analógové čítanie.
- Materiál pre kontakt prstov: Nikel.

Pripojenie modulu GSR k mini počítaču Arduino Nano je možné vykonať pomocou kábla, ktorý je dodávaný s modulom. Na doske senzora sú 4 kontakty, ale k mini počítaču je potrebné pripojiť iba 3 kontakty, ako je to opísané nižšie v tabuľke číslo 7.

Konektor	**Konektor na Arduino Nano**	**Popis**
GND	GND	Uzemnenie
VCC	3V3	Vstupný napájací kontakt
TX	A1	Zodpovedný za príjem signálu

Tabuľka 7: Pripojenie modulu GSR

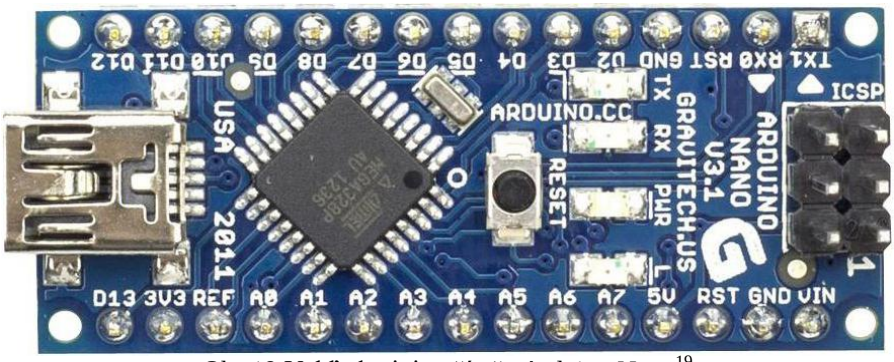
4.1.6 Mini počítač Arduino Nano

Arduino je platforma mikrokontroléra, ktorá zaujala predstavivosť nadšencov elektroniky. Vďaka ľahkému použitiu a otvorenému zdrojovému kódu je skvelou voľbou pre každého, kto chce vytvárať elektronické projekty (Monk, 2012).

Arduino je mikrokontrolér na navrhovanie elektronických zariadení, ktoré užšie spolupracujú s fyzickým prostredím ako štandardné osobné počítače, ktoré v skutočnosti neprekračujú rámec virtuality založený na mikroprocesore ATMega od spoločnosti Atmel.

Vývoj prvého Arduino začal v roku 2005, keď sa ľudia z talianskeho Interaction Design Institute v meste Ivrea rozhodli vytvoriť jednoduchý a lacný vývojový set pre študentov, ktorí si nechceli alebo nemohli zaobstarať vtedy veľmi drahé dosky, ako napríklad BASIC Stam (Zbyšek Voda a tým HW Kitchen, 2015).

Arduino Nano je malá ladiaca doska, ktorá je jedným z troch najpopulárnejších dosiek medzi rádioamatérmi a programátormi. Napriek svojej skromnej veľkosti sa prakticky nelíši od Arduino Uno vo funkčnosti a dá sa použiť v projektoch, kde rozmery hrajú významnú úlohu. Od verzie 3.0 je doska Arduio Nano vybavená pokročilejšími procesormi ATmega328, so zvýšeným počtom pamätí FLASH a EEPROM, ako aj s vyššou taktovacou frekvenciou. Softvér, ktorý používateľ používa na programovanie Arduina, sa nazýva Arduino IDE (Monk, 2014).



Obr.19 Vzhľad mini počítača Arduino Nano[19]

Konektory Digital/PWM/Analog in/Power/Reset sú súhrnne známe ako kolíkové konektory. Kolíky v týchto hlavičkách umožňujú Arduinu komunikovať s externými senzormi a inými zariadeniami (Hoffman, 2018). Nasledujúca tabuľka popisuje hlavné charakteristiky mini počítača Arduino Nano:

Mikrokontrolér	ATmega328P
EEPROM kb	1
SRAM kb	2
Flash kb	32
Digitálne vstupy a výstupy	14
Piny PWM	6
Analógové vstupy	8
Rozmery	45.0mm x 18.0mm

Tabuľka 7: Hlavné parametre mini počítača Arduino Nano

4.2 ŠPECIFIKÁCIA SOFTVÉROVEJ ČASTI

4.2.1 Databázový server

Na ukladanie a synchronizáciu údajov pre vyvinuté webové a mobilné aplikácie bola zvolena cloudová databáza Firebase Realtime Database od Google. Firebase poskytuje cloudovú NoSQL databázu pre aplikácie, ktoré bežia v reálnom čase ako služba. Táto služba poskytuje vývojárom rozhranie API na synchronizáciu údajov aplikácií medzi klientmi a ich ukladanie do cloudu Firebase. Po získaní spoločnosťou Google v roku 2014 sa Firebase rýchlo stal multifunkčným gigantom pre vývoj mobilných a webových aplikácií.

Všetky zmeny v databáze sa okamžite synchronizujú medzi všetkými klientmi alebo zariadeniami, ktoré používajú rovnakú databázu. Inými slovami, aktualizácie údajov vo Firebase sú okamžité. Údaje uložené v databáze môžu byť uložené v akejkoľvek podobe bez toho, aby boli viazané na konkrétny údajový typ. V prípade potreby stanoviť nejaké obmedzenia je možné nastaviť bezpečnostné pravidlá.

[19] Zdroj: https://canada.newark.com/productimages/large/en_US/13T9275-40.jpg

Bezpečným prenosom a úpravami údajov sa zaoberá Firebase Realtime Database Security Rules - toto je jazyk pravidiel, pomocou ktorého je možné určiť, aké práva môže mať používateľ v oblasti databázy. Napríklad do databázy majú prístup iba registrovaní používatelia. Na overenie môžeme zvoliť ľubovoľnú kombináciu e-mailu a hesla, či už je to Facebook, Twitter, GitHub, Google alebo iná sociálna sieť.
Firebase má tiež nasledujúce schopnosti:

1. Prístup z klientskych zariadení. K databáze Firebase Realtime Database je možné pristupovať priamo z mobilného zariadenia alebo webového prehliadača, pre aplikácie nie je potrebné písať server.
2. Firebase Storage je cloudové úložisko určené pre vývojárov aplikácií, ktorí potrebujú ukladať a obsluhovať akýkoľvek obsah nahraný používateľmi, napríklad obrázky alebo videosúbory.

Aby bolo možné čítať existujúce a zapisovať nové hodnoty do databázy, bude potrebné nakonfigurovať pravidlá. Aby to bolo možné urobiť je potrebné prejsť do konzoly na správu databázy v sekcii Pravidlá (Rules) a nakonfigurovať pravidlá nasledovne, ako je to znázornené na obrázku číslo 20.

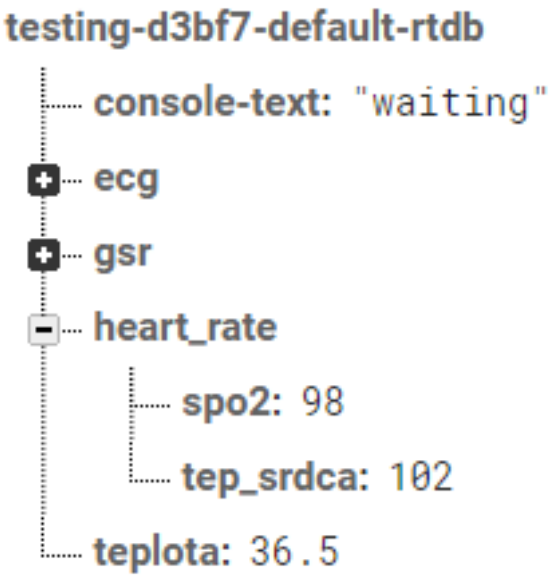
Obr.20 Konfiguračný panel pravidiel databázy Firebase Realtime Database[20]

Databáza, ktorá bola vytvorená na ukladanie a poskytovanie údajov webovej a mobilnej aplikácii, obsahuje jeden dátový uzol. Uzol obsahuje uložené záznamy s názvami senzorov, ktorých sa tieto záznamy týkajú, a záznam, ktorý zobrazuje stav systému. Každé pole s názvom snímača obsahuje prijaté údaje zo snímača, z ktorého boli

[20] Zdroj: Vlastný zdroj

prijaté, a následne sa tieto údaje automaticky zobrazia na webovej stránke alebo v mobilnej aplikácii. Pole, ktoré zobrazuje stav, má 3 možné stavy:

1. Waiting (Čakanie) – tento stav znamená, že žiadny z modulov nie je spustený a podľa toho sa do databázy nezapisujú žiadne údaje.
2. Measuring (Meranie) – tento stav znamená, že meranie prebieha zo zapojeného senzora, ktorý bol zapnutý používateľom.
3. Ready (Pripravený) – tento stav znamená, že merania boli dokončené. Tento stav sa zobrazí po zapísaní nových údajov prijatých zo senzorov do databázy a po ich zobrazení na webe.



Obr.21 Štruktúra vyvinutej databázy Firebase Realtime Database[21]

4.2.2 Webová aplikácia

Základ webovej aplikácie bol vyvinutý pomocou programovacieho jazyka Python a pomocou weboveho aplikačneho frameworku Django. Tiež boli pridané knižnice a funkcie, ktoré boli napísané v programovacom jazyku JavaScript a pre grafické zobrazenie sa používa CSS3 a HTML5, ktoré slúži na úpravu vzhľadu dokumentu a na vytvorenie šablóny aplikácie.

Django je webový framework postavený v Pythone, ktorý umožňuje programátorovi rýchlo vytvárať bezpečné a udržiavateľné webové stránky. Django je vytvorený skúsenými vývojármi a rieši väčšinu ťažkostí s vývojom webových aplikácií, takže programátor sa môže sústrediť na písanie svojich webových aplikácií bez akýchkoľvek problémov. Django je bezplatný a má otvorený zdrojový kód, má rastúcu

[21] Zdroj: Vlastný zdroj

a aktívnu komunitu, vynikajúcu dokumentáciu a veľa možností bezplatnej aj platenej podpory. Django pomáha vývojárovi písať softvér, ktorý je:

- Bezpečný. Django poskytuje bezpečný spôsob správy používateľských účtov a hesiel.
- Prenosný. Django je napísaný v jazyku Python, ktorý funguje na mnohých platformách. To znamená, že je možné spúšťať aplikácie na mnohých verziách Linux, Windows a Mac OS X.
- Univerzálny. Django možno použiť na vytvorenie takmer ľubovoľného typu webových stránok, od systémov na správu obsahu až po sociálne médiá a spravodajské portály.

HTML (Hyper Text Markup Language) je štandardizovaný značkovací jazyk pre WEB dokumenty, ktorý určený na vytváranie hypertextových dokumentov. Dokumenty, ktoré sú napísané vo formáte HTML, sa nazývajú hypertextové dokumenty. Tieto dokumenty sú prevádzané internetovým prehliadačom do podoby čitateľnej pre človeka. Pomocou formátovačov HTML je možné zostaviť zložitú hierarchickú štruktúru zo stránok HTML. Dokument vytvorený v jazyku HTML je textový súbor vo formáte ASCII, to znamená, že na vytvorenie stránky stačí vytvoriť textový dokument a zmeniť príponu na *.html*. HTML stránku je možné zobraziť vo webových prehliadačoch na rôznych platformách a môžu obsahovať mierne rozdiely v závislosti od samotného prehliadača.

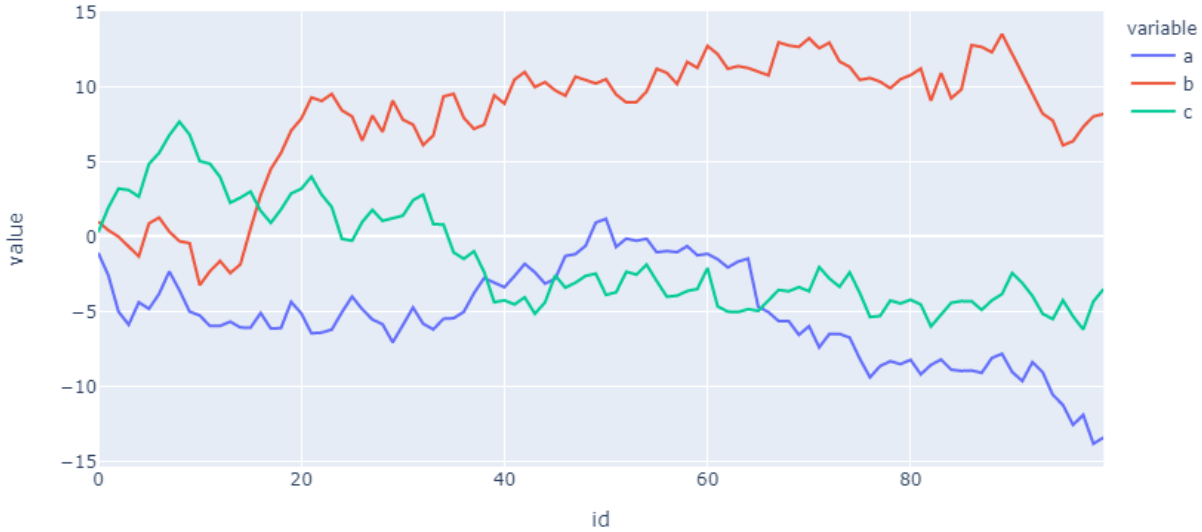
CSS (Cascading Style Sheets) je kolekcia popisov štýlov, rôznych značiek HTML, ktoré môžu odkazovať buď na jednu značku (prvok), alebo okamžite na všetky rovnaké prvky na absolútne všetkých stránkach. CSS je do istej miery pokračovaním HTML, ktoré rozširuje jeho možnosti. Jedným z cieľov CSS je oddeliť prezentáciu dokumentu od štruktúry a obsahu. Oddelenie štýlu a štruktúry uľahčuje prácu programátorovi a zvyšuje prístup pre používateľov. Šablónu štýlov stačí napísať raz pre každé zo zariadení, na ktorých sa stránka otvorí, a zároveň sa zodpovedajúcim spôsobom prispôsobiť parametrom zariadení.

JavaScript je skriptovací jazyk, vďaka ktorému sú webové stránky interaktívne. Bol vyvinutý v roku 1995 spoločnosťou Netcape na základe jazyka Java spoločnosti Sun Microsystems. Pomocou JavaScript je možné meniť webovú stránku, štýl prvkov, pridávať alebo odstraňovať rôzne tagy. Je možné použiť JavaScript na získanie informácií o akýchkoľvek akciách používateľa na stránke, ako napríklad klikanie na tlačidlá, posúvanie stránky, klikanie na tlačidlá myši, zmena pracovnej oblasti obrazovky a ďalšie.

Je tiež možné pristupovať k ľubovoľným prvkom kódu HTML a vykonávať s týmto prvkom rôzne manipulácie. Je tiež možné zobraziť správy, načítať informácie bez opätovného načítania stránky, pracovať s cookies a vykonávať mnoho ďalších postupov. Vzhľadom na to, že kód je spustený na strane klienta, stáva sa bezpečnostný problém veľmi dôležitým, takže pomocou JavaScript nie je možné pracovať s klientskými súbormi, zapisovať informácie na disk a podobne operácie.

Na zostavenie grafov a vizualizácie údajov z modulov GSR a EKG sme si vybrali knižnicu Plotly. Plotly je jeden z najbohatších nástrojov na vizualizáciu dát v Pythone pre Data Science. Plotly uľahčuje vytváranie interaktívnych grafov. Akýkoľvek diagram, ktorý môže vývojár vytvoriť pomocou tejto knižnice, je vybavený funkciami ako zväčšenie, posunutie, automatické zväčšenie atď. Vysoký výkon pri kreslení veľkého počtu bodov robí z Plotly veľkú výhodu, keď vývojár potrebuje vykresliť veľké množstvo dát. Všetky grafy Plotly sú plne prispôsobiteľné. Všetko od farieb a mien až po čiary mriežky a legendy. Knižnica má veľa možností, pomocou ktorých sme prispôsobili všetky aspekty našich grafov. Niektoré z dôležitých funkcií aplikácie Plotly sú:

- Produkuje interaktívne grafy.
- Grafy sú uložené v dátovom formáte JavaScript Object Notation (JSON), takže je možné ich čítať pomocou skriptov z iných programovacích jazykov, ako sú R, Julia, MATLAB atď.
- Grafiku je možné exportovať do rôznych rastrových aj vektorových obrazových formátov.



Obr.22 Príklad lineárneho grafu vytvoreného knižnicou Plotly[22]

[22] Zdroj: https://i.stack.imgur.com/AXJDQ.png

4.2.3 Mobilná aplikácia

Mobilná aplikácia pre operačný systém Android, ktorý je najpopulárnejším mobilným operačným systémom na svete, bola vytvorená pomocou objektovo orientovaného jazyka Java.

Java je objektovo orientovaný programovací jazyk vyvinutý spoločnosťou Sun Microsystems od roku 1991 a oficiálne vydaný 23. mája 1995. Od roku 2010 všetky práva na Javu prešli do Oracle Corporation, ktorá získala spoločnosť Sun Microsystems. Java sa dodáva s pomerne veľkou knižnicou tried. Knižnice triedy Java výrazne zjednodušujú vývoj aplikácií tým, že programátorovi poskytujú výkonné nástroje na riešenie bežných problémov. Programátor preto môže venovať väčšiu pozornosť riešeniu aplikovaných problémov, a nie napríklad organizovaniu dynamických polí, interakcii s operačným systémom alebo implementácii prvkov používateľského rozhrania. Programy Java sú preložené do bytecode, ktorý vykonáva program Java Virtual Machine (JVM), program, ktorý spracováva bajtový kód a odosiela pokyny hardvéru. Výhodou tohto spôsobu vykonávania programov je, že bajtový kód je úplne nezávislý od operačného systému a hardvéru, čo umožňuje spúšťať aplikácie Java na akomkoľvek zariadení, pre ktoré existuje zodpovedajúci virtuálny stroj. Vlastnosti programovacieho jazyka Java:

- poskytuje svoje applety na všeobecné použitie - malé, robustné, dynamické, nezávislé na platforme aktívne sieťové aplikácie, proaktívne sieťové aplikácie zabudované do webových stránok.
- uvoľňuje silu objektovo orientovaného vývoja aplikácií.
- poskytuje programátorovi bohatú sadu tried objektov, aby jasne abstrahovala od mnohých systémových funkcií používaných pri vytváraní okien, vytváraní sietí a vo vstupnom a výstupnom systéme.
- podpora automatického zberu odpadu.
- flexibilný bezpečnostný systém z dôvodu, že vykonávanie programu je úplne riadené virtuálnym strojom. Akékoľvek operácie, ktoré prekročia nastavené oprávnenia programu (napríklad pokus o neoprávnený prístup k údajom alebo o pripojenie k inému počítaču), spôsobia okamžité prerušenie.
- prenositeľnosť programov medzi platformami a hardvérom.

V každom projekte pre Android je potrebné zvoliť minimálnu verziu platformy Android podporovanú aplikáciou. Čím je verzia menšia, tým viac zariadení bude možné aplikáciu nainštalovať, ale bude nemožné, alebo obmedzené pracovať s niektorými

funkciami API neskorších verzií platformy. Preto bolo rozhodnuté zvoliť verziu 5 ako minimálnu podporovanú verziu. Pri výbere minimálnej podporovanej verzie systému Android, vývojové prostredie Android Studio navrhne percento podporovaných zariadení. V našom prípade je to 94,1%.

Dizajn aplikácie bol vyvinutý na základe princípov materiálového dizajnu (Material Design). Materiálové prevedenie je komplexný koncept na vytváranie vizuálnych, pohyblivých a interaktívnych prvkov pre rôzne platformy a zariadenia od spoločnosti Google.

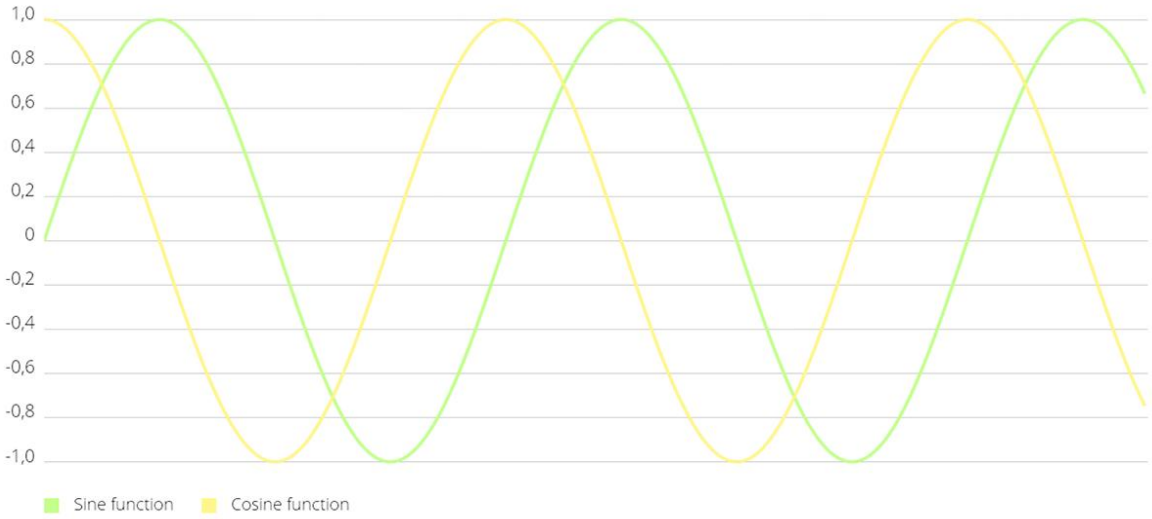
Softvérová aplikácia pre OS Android sa skladá zo sady aktivít, z ktorých každá zodpovedá obrazovke aplikácie. Každú aktivitu v projekte predstavuje trieda implementovaná v jazyku Java uložená v súbore s rovnakým názvom s príponou .java. Každá aktivita má zodpovedajúci súbor s popisom XML. Súbor xml popisuje umiestnenie vykreslených objektov vo forme kódu xml. Po spustení aktivity systém Android automaticky rozpozná veľkosť obrazovky mobilného zariadenia a prinesie zobrazený obsah v súlade so značením popísaným v súbore xml. Rovnaká aktivita teda bude vyzerať rovnako bez ohľadu na uhlopriečku použitého zariadenia. Pre každú aplikáciu Android tiež musí existovať súbor XML, do ktorého sa zapíšu minimálne systémové požiadavky vo forme kódu XML, ako aj aktivita vyvolaná pri spustení aplikácie.

Aplikácia sa funkčne skladá z jednej hlavnej obrazovky (Activity), na ktorej sú umiestnené grafy, obrázky a hodnoty, a všetky vyzerajú rovnako ako na obrazovke webovej aplikácie. Toto bolo urobené preto, aby užívateľ v prípade potreby mohol ľahko a bez nepríjemných pocitov prejsť z mobilnej aplikácie na webovú aplikáciu alebo naopak.

Na vytváranie grafov bolo rozhodnuté použiť knižnicu MPAndroidChart. Je to jednoduchá knižnica grafov pre Android, ktorá podporuje riadkové, čiarové, bodové, sviečkové a výsečové grafy, ako aj škálovanie, presúvanie, výber a animácie. Základné funkcie:

- Mierka na oboch osiach (jednoduchým dotykovým gestom).
- Ťahanie (dotykovým gestom).
- Kreslenie prstom (kreslenie hodnôt do grafu pomocou dotykového gesta).
- Zvýraznenie hodnôt (s prispôsobiteľnými kontextovými oknami).
- Ukladanie grafov na SD kartu (ako obrázok alebo ako súbor .txt)
- Čítanie údajov grafu zo súboru .txt.
- Prednastavené farebné šablóny.

- Legendy (vytvorené automaticky, prispôsobené).
- Štítky (osi X a Y, prispôsobiteľné).
- Animácia (vytváranie animácií pozdĺž osí X a Y).
- Limitné čiary (poskytujúce ďalšie informácie, maximálne hodnoty atď.).
- Plne prispôsobiteľné (farby, písma, legendy, pozadie, gestá, prerušované čiary, atď.).



Obr.23 Príklad lineárneho grafu vytvoreného knižnicou MPAndroidChart[23]

4.3 REALIZÁCIA ZAPOJENIA

4.3.1 Diagramy

Na vytváranie diagramov bola zvolená aplikácia Draw.io. Je to nástroj na vytváranie diagramov, blokových schém, myšlienkových máp, obchodných rozložení, vzťahov entít, programovacích blokov a ďalších. Program je distribuovaný bezplatne aj s otvoreným zdrojovým kódom. Tento nástroj umožňuje vytvárať nasledujúce položky:

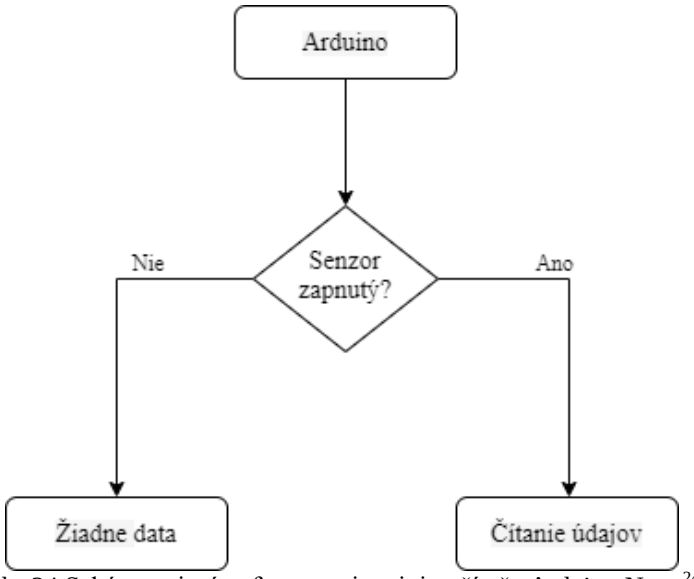
- Grafy.
- Diagramy.
- Tabuľky.
- Prezentácie.
- Blokové diagramy.
- Modely UML.
- Mapy stránok.

[23] Zdroj: https://github.com/PhilJay/MPAndroidChart/blob/master/screenshots/simpledesign_linechart5.png

Draw.io má nasledujúce funkcie:

- viac ako 500 šablón prvkov a tvarov.
- ľahké rozhranie, v ktorom je možné vytvoriť hotový projekt v krátkom časovom období.
- podpora klávesových skratiek používaných vo väčšine grafických editorov.
- export do formátov: JPG, PNG, SVG, VDSX.
- schopnosť pracovať spoločne.
- dostupnosť rôznych základných tém.
- existuje možnosť integrácie s dokumentmi Google, Dropbox, OneDrive, JIRA, Confluence, Chrome a GitHub.
- viacjazyčné rozhranie.

Spôsob fungovania dosky Arduino Nano spočíva v tom, že doska kontroluje, či sú zapnuté moduly EKG alebo GSR. V prípade, že moduly nefungujú, systém na port prenosu údajov odošle správu ziadne data. Prijatie tejto správy z portu znamená, že uplynie čakanie na 2 druhy príkazov. Prvý príkaz je správa, ktorá sa zobrazí, keď je zapnutý modul EKG, a druhý príkaz je, keď je zapnutý modul GSR. Ak je niektorý z modulov v prevádzke, vykoná sa kód zodpovedný za príjem a odosielanie údajov.

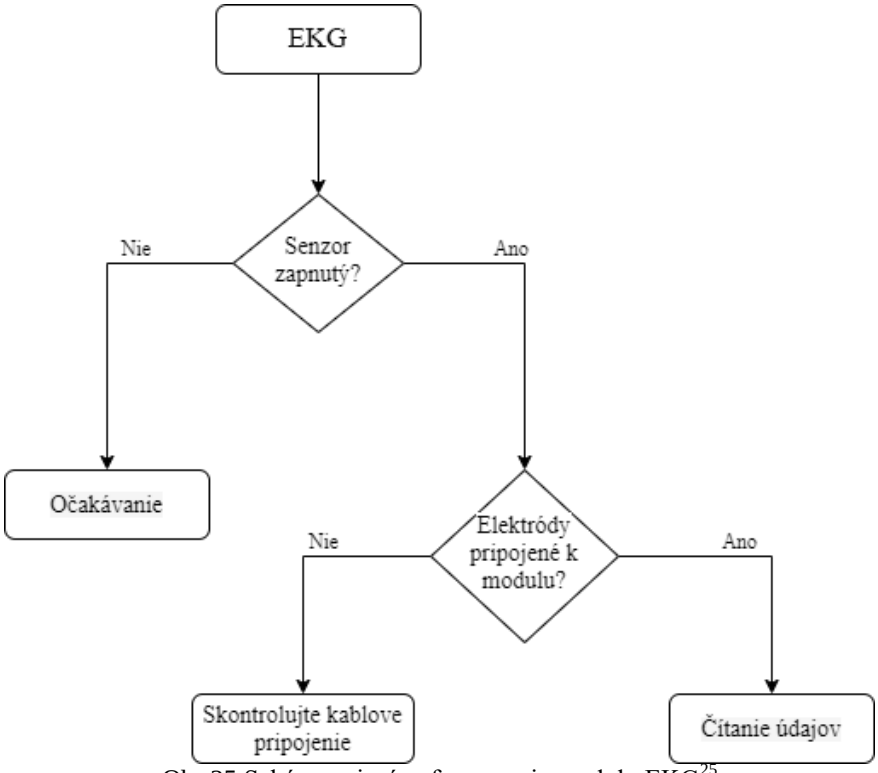


Obr.24 Schéma princípu fungovania mini počítača Arduino Nano[24]

Nižšie je uvedený diagram činnosti modulu EKG, ktorý je pripojený k mini počítaču Arduino Nano. Ak je modul vypnutý, potom doska Arduino pokračuje v ďalšom monitorovaní údajov a čaká na zapnutie modulu. Ak je modul zapnutý, potom sa ďalej kontroluje, či je pripojený kábel s elektródami alebo nie. Ak elektródy nie sú pripojené,

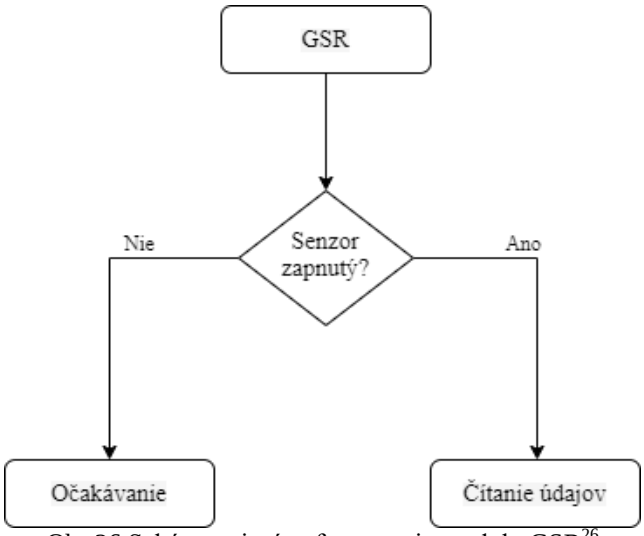
[24] Zdroj: Vlastný zdroj

potom sa do portu prenesie správa Skontrolujte kablove pripojenie. V takom prípade sa do databázy neposielajú žiadne údaje. Ak sú pripojené elektródy, dáta sa načítajú a odošlú sa do portu. Na načítanie údajov bol nastavený čas 5 minút.



Obr.25 Schéma princípu fungovania modulu EKG[25]

Nasleduje schéma činnosti modulu GSR. Ak je modul vypnutý, doska Arduino pokračuje v ďalšom monitorovaní údajov. Ak je modul zapnutý, data sa načítajú zo senzorov na prstoch a ďalej nasleduje výpočet a odoslanie dát na port.



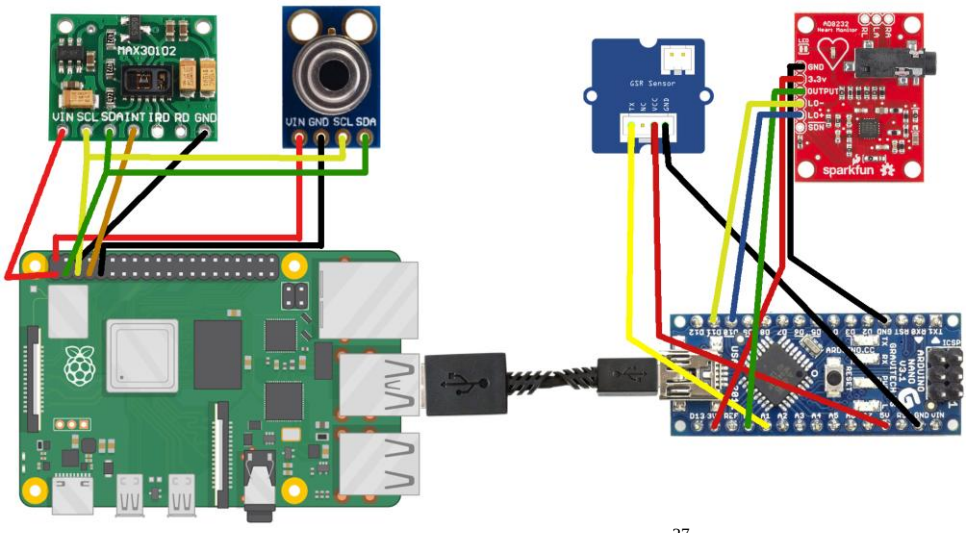
Obr.26 Schéma princípu fungovania modulu GSR[26]

[25] Zdroj: Vlastný zdroj
[26] Zdroj: Vlastný zdroj

4.3.2 Schéma zapojenia

Samotná realizácia spočíva v pospájaní a zapojení jednotlivých modulov a prvkov do plne funkčného systému. Korektné zapojenie hardvéru a vytvorenie správneho softvéru nám zabezpečí správny zber dát senzormi, odosielanie týchto údajov do databázy a vo všeobecnosti zabezpečí správne fungovanie nášho projektu. Bezkontaktné moduly na meranie teploty a srdcového rytmu sú pripojené k mini počítaču Raspberry Pi 4B. Moduly EKG a GSR sú pripojené k mini počítaču Arduino Nano. Samotná doska Arduino Nano je pripojená pomocou USB kábla k doske Raspberry.

Vzhľadom na to, že Raspberry Pi 4B nemá analógové porty ani hardvérový analógovo-digitálny prevodník, na pripojenie modulu EKG a GSR by bolo potrebné použiť doplnkový modul prevodníka. Preto namiesto prevodníka používame dosku Arduino Nano, ktorá prenáša dáta prijaté z modulov pomocou USB kábla.



Obr.27 Ilustrovaná schéma zapojenia[27]

4.4 VYTVORENIE SOFTVÉROVEJ ČASTI PROJEKTU

4.4.1 Programovanie Arduino Nano

Arduino Nano bolo programované v programovacom jazyku založenom na C/C ++ s preddefinovanými funkciami. Ako vývojové prostredie padla voľba na Arduino IDE, čo výrazne zjednodušuje písanie programov pre túto platformu a vytváranie zariadení na Arduine sa stáva oveľa prístupnejším pre ľudí, ktorí nemajú veľa znalostí jazyka C / C ++. Rovnako ako v iných programovacích jazykoch založených na C / C ++, existuje niekoľko pravidiel pre písanie kódu. Tu sú tie najzákladnejšie:

- Po každej inštrukcii musí nasledovať bodkočiarka (;).

[27] Zdroj: Vlastný zdroj

- Pred vyhlásením funkcie musíte určiť dátový typ vrátený funkciou alebo (void), ak funkcia nevráti hodnotu.
- Pred vyhlásením premennej musíte určiť typ údajov.
- Komentáre sú označené: // toto je pre vložené komentáre a /* pre komentáre blokového typu.

Program vyvinutý na Arduine, keď je spustený na zariadení, implementuje rôzne projektové funkcie, napríklad čítanie dát zo senzorov GSR a EKG, ich prevod na reťazce a ich prenos do konzoly, odkiaľ ich vezme Raspberry Pi pre ďalšie spracovanie.

Prvým krokom pri vytváraní programu je pridanie všetkých knižníc, ktoré sú potrebné na to, aby program správne fungoval na doske Arduino Nano. Jediná knižnica, ktorú sme zahrnuli, je knižnica pre správu času GyverTimer.h. Toto je plnohodnotný časovač založený na systéme *millis ()* / *micros ()*, ktorý poskytuje pohodlný multitasking a správu času pomocou iba jedného natívneho prerušenia časovača. Jednotka času v milisekundách bola nastavená ako jediné nastavenie časovača.

Ďalším krokom je identifikácia premenných, ktoré sú potrebné na výpočet a inicializáciu analógových vstupov pre použitie modulov GSR a EKG.

V metóde *setup()* sa inicializuje sériové pripojenie a nastaví sa prenosová rýchlosť, čo je nevyhnutné pre následný príjem dát z Raspberry Pi. Tiež dôjde k priradeniu portov na používanie EKG a pridelí sa čas pre prácu modulu. V tomto prípade bol nastavený čas na 5 minút, čo sa v milisekundách rovná 300000.

```
#include "GyverTimer.h"

const int ECG=A0;
const int GSR=A1;
int sensorValue=0;
int gsr_average=0;
int start_value=0;

GTimer myTimer(MS);

void setup() {
  Serial.begin(9600);
  myTimer.setInterval(300000);
  pinMode(10, INPUT); // Setup for leads off detection LO +
  pinMode(11, INPUT); // Setup for leads off detection LO -
}
```

Obr.28 Kód definovania knižníc a pinov[28]

Vďaka neustálemu opakovaniu tela funkcie *loop()* nie je potrebné opakovanie vykonávať programovým kódom. Funkcia *loop()* je miesto, kde musíme zadávať príkazy,

[28] Zdroj: Vlastný zdroj

ktoré budú bežať, pokiaľ je Arduino zapnuté. Od prvého príkazu, pôjde mini počítač až do konca a okamžite skočií na začiatok, aby opakoval rovnakú postupnosť.

Táto časť funkcie *loop()* obsahuje kód, ktorý je zodpovedný za funkčnosť modulu EKG. Prebieha kontrola či je modul zapnutý, či sú pripojené elektródy. Ak je všetko v poriadku, potom je časovač zapnutý. Počas doby kedy je časovač zapnutý prebieha meranie s následným výstupom dát do konzoly. Dôležitým prvkom vo funkcii *loop()* je výstup fráz *start ecg* a *koniec* do konzoly. Pretože tieto frázy sú signálom pre mini počítač Raspberry Pi o tom, že boli spustené merania, z ktorého senzora sa merania uskutočňujú a kedy sa skončia. Sú potrebné pre správne sledovanie senzorov a správne spracovanie údajov v budúcnosti.

```
void loop() {
  long sum=0;
  if (analogRead(ECG) > 100){
    if((digitalRead(10) == 1)||(digitalRead(11) == 1)){
      Serial.println("Skontrolujte kablove pripojenie");
    }
    else{
      if (myTimer.isEnabled()){
        if (start_value == 0){
          Serial.println("start ecg");
          start_value += 1;
          }
        Serial.println(analogRead(ECG));
      }
      if (myTimer.isReady()){
        Serial.println("koniec");
        start_value = 0;
        myTimer.stop();
      }
    }
  }
```

Obr.29 Kód pre prvú časť metódy loop()[29]

Táto časť funkcie *loop()* obsahuje kód, ktorý je zodpovedný za funkčnosť modulu GSR pripojeného k doske Arduino Nano. Tu sa kontroluje, či je modul zapnutý alebo nie. Ak je všetko v poriadku, potom sa automaticky zapne časovač, počas ktorého prebieha meranie a zároveň sa dáta odošlú na konzolu. Rovnako ako modul AD8232, ktorý sa používa na meranie elektrickej aktivity srdca, počas meraní galvanickej odozvy pokožky (GSR) modul odosiela aj počiatočné a koncové hodnoty pre dosku Raspberry Pi.

Po dokončení práce každého z modulov a následnom uplynutí časovača prebieha čakanie na vypnutie modulu, ktorý bol v praci. Je to nevyhnutné, ak užívateľ po meraniach zabudol modul vypnúť. Ak jeden modul nie je vypnutý, druhý nebude fungovať.

[29] Zdroj: Vlastný zdroj

Pokiaľ nie je zapnutý žiadny modul, na USB porte sa zobrazí správa „ziadne data“, čo pre Raspberry Pi znamená, že je potrebné pokračovať v načítaní USB portu a čakať na zapnutie a následný príjem dát zo zapnutého jedného z modulov.

```
else if(analogRead(GSR) > 100){
  if (myTimer.isEnabled()){
    if (start_value == 0){
      Serial.println("start gsr");
      start_value += 1;
    }
    for (int i=0;i<10;i++){
      sensorValue=analogRead(GSR);
      sum += sensorValue;
      delay(5);
    }
    gsr_average = sum/10;
    Serial.println(gsr_average);
  }
  if(myTimer.isReady()){
    Serial.println("koniec");
    start_value = 0;
    myTimer.stop();
  }
 }
 else{
   Serial.println("ziadne data");
   start_value = 0;
   myTimer.resume();
   myTimer.reset();
 }
 delay(50);
}
```

Obr.30 Kód pre druhú časť metódy loop()[30]

4.4.2 Programovanie Raspberry Pi 4B

Prvým krokom pri vytváraní systému bolo potrebné najskôr všetky moduly, senzory a dosku Arduino Nano pripojiť k mikropočítaču Raspberry Pi 4B ako sme popisovali v predchádzajúcich kapitolách. Ďalším krokom je vytvorenie programu, ktorý pomocou pripojených knižníc a pod kontrolou Rasberry Pi bude spravovať prijaté dáta a odosielať ich do databázy Firebase Realtime Database.

Ako programovací jazyk pre vyvíjaný program bol zvolený Python. Jazyk Python je jedným z najpopulárnejších objektovo-orientovaných programovacích jazykov súčasnosti, ktorý kombinuje jasný a jednoduchý syntax a výkonné schopnosti pomocou vstavaných a zásuvných knižníc. Najčastejšie sa Python používa pri práci s big data a vývojom webových stránok. Je vhodný aj na vytváranie desktopových a mobilných aplikácií.

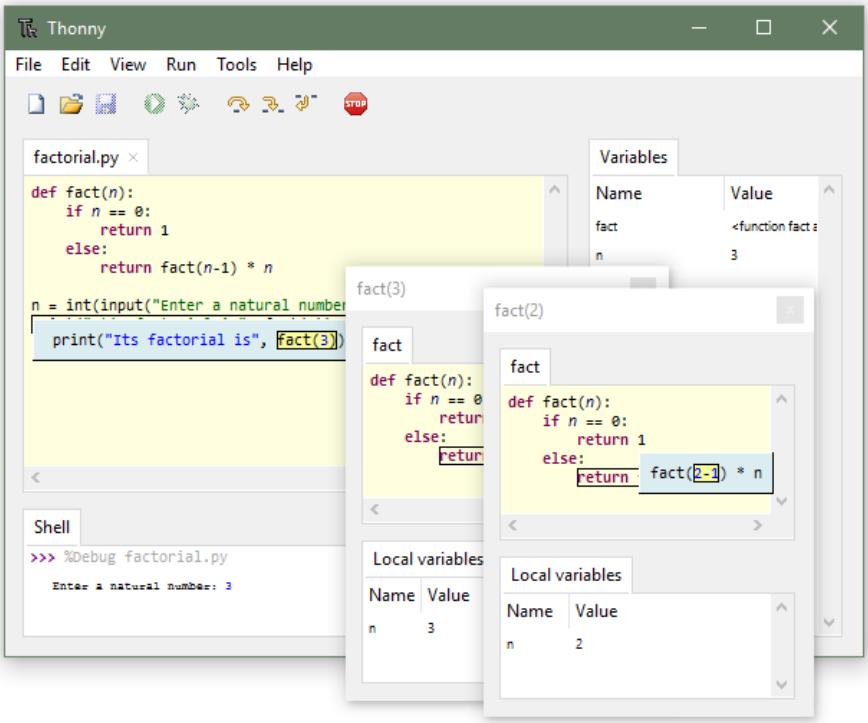
[30] Zdroj: Vlastný zdroj

Výhody programovacieho jazyka Python:

1. Python je interpretovaný programovací jazyk. To znamená, že zdrojový kód sa po jeho načítaní špeciálnym interpretačným programom transformuje po častiach do strojového kódu. Toto výrazne uľahčuje ladenie programov.
2. Python je dynamicky typovaný jazyk. To znamená, že nie je potrebné vopred definovať typ premennej, čo je počas vývoja veľmi výhodné.
3. Podpora objektovo orientovaného programovania.
4. Automatický zber odpadkov, žiadne úniky pamäte.
5. Jasný a lakonický syntax, ktorý prispieva k prehľadnému zobrazeniu kódu. Pohodlný systém funkcií umožňuje kompetentným prístupom vytvoriť kód, ktorý v prípade potreby ľahko pochopí iná osoba.
6. Obrovské množstvo modulov, ktoré sú súčasťou štandardnej distribúcie Python 3. V niektorých prípadoch stačí na napísanie programu nájsť vhodné moduly a správne ich skombinovať.
7. Schopnosť softvéru pracovať s viacerými hardvérovými platformami alebo operačnými systémami. Program napísaný v Pythone bude fungovať úplne rovnako bez ohľadu na to, na akom operačnom systéme beží.

V mini počítači Raspberry Pi sa Python používa na prácu so senzormi cez GPIO pomocou špeciálnej knižnice. Balík potrebný na použitie Pythonu je už nainštalovaný na operačnom systéme Raspbian v dvoch verziách - 2 a 3. Na vývoj programu bola použitá tretia verzia Pythonu, pretože niektoré knižnice už nepodporujú predchádzajúcu, druhú verziu. Na vývoj programu pre dosku Raspberry Pi v programovacom jazyku Python sme si vybrali vývojové prostredie Thonny IDE. Je to integrované vývojové prostredie pre Python, ktoré určené pre začiatočníkov. Podporuje rôzne spôsoby krokovania kódu, vyhodnotenie výrazov, podrobnú vizualizáciu zásobníka vyzváňania. Zvláštnosti Thonny IDE:

- Postupné vykonávanie výrazov bez zastavenia.
- Samostatné okno pre volanie funkcií na vysvetlenie miestnych premenných a zásobníka.
- K dispozícii je zobrazenie čísel riadkov.
- Jednoduché grafické rozhranie správcu balíkov pip.



Obr.31 Vzhľad hlavného okna vývojového prostredia Thonny IDE[31]

Na zabezpečenie správnej práceschopnosti každého snímača a určitej časti kódu je potrebné pripojiť určité moduly, alebo inak povedané, knižnice. Spočiatku počas vývoja boli pripojené nasledujúce moduly:

- time - modul pre prácu s časom v Pythone. Konkrétne sa v skripte, ktorý sme vyvinuli, používa na volanie funkcie time.sleep(), ktorá slúži na pridanie oneskorenia pri vykonávaní programu.
- max30102 - obsahuje triedu, ktorá má niektoré konfiguračné a čítacie funkcie pre senzory pulzného oxymetra a monitor srdcového tepu.
- hrcalc - poskytuje funkciu výpočtu srdcového rytmu a SpO2.
- smbus – modul, ktorý poskytuje možnosť používať rozhranie I2C. Tento modul je potrebný na prácu so snímačom teploty a snímačom srdcového tepu, ktoré sú pripojené priamo k doske Raspberra Pi.
- errno – modul, pomocou ktorého je možné okamžite zistiť, čo znamená každý prijatý chybový kód.
- serial - tento modul zapuzdruje prístup k sériovému portu. Je potrebný na príjem dát z dosky Arduino.
- mlx90614 – modul, ktorý riadi činnosť bezkontaktného infračerveného snímača teploty GY-906.

[31] Zdroj: https://thonny.org/img/screenshot.png

- pyrebase – modul, ktorý je potrebný na správne pripojenie a prácu s databázou Firebase Realtime Database.
- datetime – modul, ktorý slúži na manipuláciu s dátumom a časom.

Na samom začiatku programu prebieha pripojenie k databáze. Pripojenie sa vykonáva pomocou prijatých parametrov pri registrácii v konzole Firebase Realtime Database a autorizácii pomocou prihlasovacieho mena a hesla používateľa.

Metóda *get_date()* je navrhnutá tak, aby získala aktuálny čas, keď dôjde k meraniu, a vrátila textovú šablónu názvu. Tento názov sa používa pre názvy súborov snímačov, v ktorých sú uložené namerané hodnoty. Šablóna názvu pozostáva z mesiaca, dňa, hodiny, minúty a sekundy merania.

Metóda *get_time()* používa knižnicu datetime na získanie aktuálneho času, keď dôjde k meraniu, a vráti čas začiatku merania v mikrosekundách. To je potrebné na výpočet času merania srdcovej frekvencie (EKG) a galvanickej odozvy pokožky (GSR). Ďalej sa tento čas spolu s hodnotami modulov zapíše do súboru na uloženie a potom sa hodnoty spolu s časom odošlú do databázy.

```
config = {
    "apiKey": "AIzaSyArqn_uKggNucpDsHbWIreoRdP-T11Lpuc",
    "authDomain": "testing-d3bf7.firebaseapp.com",
    "databaseURL": "https://testing-d3bf7-default-rtdb.firebaseio.com",
    "storageBucket": "testing-d3bf7.appspot.com",
    "serviceAccount": "/home/pi/testing.json"
}
firebase = pyrebase.initialize_app(config)

auth = firebase.auth()

user = auth.sign_in_with_email_and_password("maximus2933@gmail.com", "testtest2020")
db = firebase.database()


def get_date():
    current_datetime = datetime.now()
    month = current_datetime.month
    day = current_datetime.day
    hour = current_datetime.hour
    minute = current_datetime.minute
    second = current_datetime.second
    name = "data{0}_{1}_{2}_{3}_{4}".format(month, day, hour, minute, second)
    return name


def get_time():
    time_now = datetime.now()
    return (time_now.hour * 3600000000) + (time_now.minute * 60000000) + (time_now.second * 1000000) + time_now.microsecond
```

Obr.32 Kód pripojenia k database a definovania metód get_date() a get_time()[32]

Metóda *check_raspberry()* bola vytvorená na kontrolu spustených senzorov, ktoré sú priamo pripojené k doske Raspberry Pi 4B cez rozhranie I2C. Senzory, ktoré sú monitorované touto metódou, sú snímač srdcového tepu a snímač teploty. Každý z nich má svoju vlastnú adresu, vďaka ktorej je ľahké pochopiť, ktorý z nich je momentálne zapnutý.

[32] Zdroj: Vlastný zdroj

Ďalej program skenuje zbernicu I2C na prítomnosť zariadení a ak jeden zo senzorov beží, potom sa volá určitá metóda, ktorá je priamo zodpovedná za funkčnosť tohto senzora. Pokiaľ v danom okamihu nie je spustený žiadny zo senzorov, potom sa zobrazí určitý stav systému a kontrola sa vykonáva ďalej.

Metóda *arduino_connect()* je potrebná na spojenie a následné načítanie informácií z dosky Arduino Nano a pripojenie k doske modulov pre záznam elektrickej aktivity srdca (EKG) a galvanickej odozvy kože (GSR). Prijaté informácie sa vrátia ako premenná alebo sa vráti 1, ak dôjde k chybe alebo ak nie je pripojená doska Arduino, táto hodnota bude spracovaná v ďalších metódach.

```
def check_raspberry():
    bus_number = 1
    bus = smbus.SMBus(bus_number)
    code_device = ""

    for device in range(3, 128):
        try:
            bus.write_byte(device, 0)
            print("Found {0}".format(hex(device)))
            code_device += hex(device)
            print(code_device)
        except IOError as e:
            if e.errno != errno.EREMOTEIO:
                print("Error: {0} on address {1}".format(e, hex(address)))
                countdown(int(20), 1)
        except Exception as e:
            print("Error unk: {0} on address {1}".format(e, hex(address)))
    bus.close()
    bus = None

    if code_device == "0x5a" or code_device == "0x570x5a":
        print("Teplota")
        temperature()
    elif code_device == "0x57":
        print("Pulz")
        heart_rate()
    else:
        print("Ziaden sensor nie pripojeny")

def arduino_connect():
    try:
        ser = serial.Serial('/dev/ttyUSB0', 9600)
        return ser
    except:
        return 1
```

Obr.33 Kód definovania metód check_raspberry() a arduino_connect()[33]

Metóda *arduino_module()* berie názov modulu ako parameter, ktorý bol spustený a z ktorého sa budú čítať údaje. Spočiatku sa do databázy Firebase Realtime Database zapíše hodnota vo vetve, ktorá je zodpovedná za zobrazenie stavu systému, čo znamená, že meranie teraz prebieha. Ďalej sa aktuálny čas načíta do premennej, ktorá bude v budúcnosti potrebná na výpočet prevádzkovej doby zahrnutého modulu. Potom sa staré záznamy

[33] Zdroj: Vlastný zdroj

odstránia z databázy Firebase Realtime Database. Je to nevyhnutné, pretože do databázy je možné uložiť iba posledné meranie a tie minulé sa ukladajú do pamäte dosky Raspberry Pi. Ďalej sa vytvorí objekt metódy *arduino_connect()* na príjem údajov z dosky Arudino Nano, vytvorí sa objekt metódy *get_date()* na získanie názvu súboru, do ktorého sa budú ukladať údaje z aktuálneho merania a vytvorí sa premenná module_mode, ktorá poskytuje informácie o tom, či je stále dovolené čítať údaje z modulov alebo nie. Potom sa informácie načítajú z modulu prijatého z dosky Arduino Nano cez port a zapíšu sa do súboru, ktorý má prijatú šablónu názvu. Pred začatím zápisu hodnôt do databázy sa systémový stav aktualizuje na stav, ktorý znamená, že meranie je dokončené a hodnoty sa odosielajú. Zápis sa vykonáva v dvoch premenných, prvá premenná je zodpovedná za uloženie hodnoty a druhá za čas, po ktorom bola táto hodnota prijatá. Čítanie a zápis sa končí, keď doska Arduino dokončí prácu s modulom a odošle konečný stav merania. Nakoniec sa po odoslaní všetkých hodnôt do databázy odošle nový stav systému, čo znamená, že čítanie dát zo senzora skončilo. Potom sa spustí časovač na 30 sekúnd. Časovač na 30 sekúnd je potrebný, aby mohol používateľ modul vypnúť a odstrániť senzory alebo elektródy.

```
def arduino_module(name):
    db.child("console-text").set("measuring", user['idToken'])
    start_time = get_time()
    db.child("{0}".format(name)).remove()
    arduino = arduino_connect()
    module_mode = 0
    file_name = get_date()
    f = open("/home/pi/Diplom/Data/{0}/{1}.log".format(name, file_name), "w")
    while module_mode != 1:
        rawserial = arduino.readline()
        cookedserial = rawserial.decode('utf-8').strip('\r\n')
        if cookedserial != "koniec":
            if cookedserial != "ziadne data" and cookedserial != "start {0}".format(name):
                print(cookedserial)
                finish_time = get_time()
                f.write("{0} {1}\n".format(cookedserial, (finish_time - start_time)))
                print("End write")
        else:
            print("Close write")
            f.close()
            module_mode = 1
    db.child("console-text").set("loading", user['idToken'])
    with open("/home/pi/Diplom/Data/{0}/{1}.log".format(name, file_name), "r") as t:
        for line in t:
            split_data = line.split()
            db.child("/{0}".format(name)).push({'value': split_data[0], 'time': split_data[1]}, user['idToken'])
    db.child("console-text").set("ready", user['idToken'])
    countdown(int(30), 1)
```

Obr.34 Kód definovania metódy arduino_module()[34]

Metóda *heart_rate()* je navrhnutá na prácu so snímačom srdcovej frekvencie a saturácie kyslíka MAX30102. Od začiatku metódy sa stav zapíše do databázy Firebase Realtime Database, čo znamená, že v danom čase prebieha meranie. Ďalej sa vytvorí objekt metódy *get_date()* na získanie názvu súboru, do ktorého sa budú ukladať údaje

[34] Zdroj: Vlastný zdroj

z aktuálneho merania. Potom prebieha pripojenie k senzorom, čítanie údajov a ich zápis do súboru. Vzhľadom na to, že hodnoty tepu a hodnoty saturácie krvi kyslíkom sú úplne odlišné údaje a nemôžu byť zmiešané, bolo rozhodnuté zaznamenať údaje do rôznych súborov. Až potom priebeha vypínanie snímača a po matematickom výpočte nasleduje zápis do databázy Firebase Realtime Database.

Zápis do databázy sa vykonáva metódou *db.child().set()* a na začiatku neexistuje žiadna metóda, ktorá by bola zodpovedná za vymazanie predchádzajúcich hodnôt v databáze tak, ako to bolo urobené napríklad v časti kódu ktorý je zodpovedný za čítanie a zápis hodnôt z modulu galvanickej odozvy kože (GSR) a z modulu pre záznam elektrickej aktivity srdca (EKG) ktoré sú pripojené k doske Arduino Nano. Je to všetko preto, že tu sú hodnoty v databáze vždy uložené v jednom riadku a pre pridanie nového nie je potrebné mazať starý riadok. Namiesto toho sa použije metóda *db.child().set()*, ktorá automaticky nahradí starú hodnotu novou. A v metóde *arduino_module()*, ktorá je zodpovedná za čítanie a odosielanie hodnôt prijatých z modulov, ktoré sú pripojené k doske Arduino Nano, ukladanie dát je usporiadané trochu inak. Vzhľadom na to, že existuje veľké množstvo hodnôt prijatých z modulov EKG a GSR, ktoré sa nedajú automaticky nahradiť, staré sa najskôr vymažú a potom sa postupne načítajú nové hodnoty.

Po ukončení merania a zaznamenaní týchto meraní do databázy sa stav systému aktualizuje na stav, ktorý znamená dokončenie meraní. Potom sa spustí časovač na 30 sekúnd. V prípade, že používateľ odpojí senzor, automaticky sa zavolá metóda *find()* na sledovanie zapojenia jedného z dostupných senzorov alebo modulov.

```
def heart_rate():
    db.child("console-text").set("measuring", user['idToken'])
    try:
        file_name = get_date()
        m = max30102.MAX30102()
        red, ir = m.read_sequential(1000)
        with open("/home/pi/Diplom/Data/pulse/red_{0}.log".format(file_name), "w") as f:
            for r in red:
                f.write("{0}\n".format(r))
        with open("/home/pi/Diplom/Data/pulse/ir_{0}.log".format(file_name), "w") as f:
            for r in ir:
                f.write("{0}\n".format(r))

        m.shutdown()

        irs = []
        with open("/home/pi/Diplom/Data/pulse/ir_{0}.log".format(file_name), "r") as f:
            for line in f:
                irs.append(int(line))

        reds = []
        with open("/home/pi/Diplom/Data/pulse/red_{0}.log".format(file_name), "r") as f:
            for line in f:
                reds.append(int(line))

        value_count = 0
        hr_value = 0
        spo2_value = 0
        db.child("console-text").set("loading", user['idToken'])
        for i in range(37):
            hr, spo2 = hrcalc.calc_hr_and_spo2(irs[25 * i:25 * i + 100], reds[25 * i:25 * i + 100])
            if hr != -999 and spo2 != -999:
                value_count += 1
                hr_value += hr
                spo2_value += spo2
        db.child("heart_rate/spo2").set(round(spo2_value / value_count), user['idToken'])
        db.child("heart_rate/tep_srdca").set(round(hr_value / value_count), user['idToken'])
        db.child("console-text").set("ready", user['idToken'])
        countdown(int(30), 1)
    except:
        find()
```

Obr.35 Kód metódy heart_rate()[35]

Metóda *temperature()* je navrhnutá na prácu so snímačom teploty GY-906 MLX90614. Od začiatku metódy sa stav práce systému odosiela do databázy Firebase Realtime Database, čo znamená, že v čase práce tohto modulu prebieha meranie. Ďalej sa vytvorí objekt metódy *get_date()* na získanie názvu súboru, do ktorého sa budú ukladať údaje z aktuálneho merania. Ďalším krokom je pripojenie k senzoru pomocou jeho adresy, ďalej sa údaje načítajú a zapisujú do súboru. Čas, po ktorý meria snímač teploty, je nastavený na 10 sekúnd. Tento čas je ideálny na získanie presných údajov o telesnej teplote. Potom sa snímač vypne, hodnoty sa vypočítajú a zapíšu do databázy Firebase Realtime Database. Po ukončení merania a aktualizácii prijatých údajov v databáze sa tiež aktualizuje stav systému, čo znamená, že došlo k dokončeniu meraní. Potom sa spustí časovač na 30 sekúnd.

Metóda *find()* je navrhnutá na čakanie a nepretržité sledovanie zapojenia ktoréhokoľvek z modulov. Spočiatku je sledovaná doska Arduino. Ak je na doske zapnutý niektorý z modulov, zavolá sa metóda *arduino_module()*. Ako hlavný parameter metóda berie názov modulu, z ktorého bude musieť načítať údaje. Pokiaľ nie je zapnutý žiadny z modulov, je ďalším krokom sledovanie senzorov, ktoré sú pripojené k Raspberry Pi. Pokiaľ nie je zapnutý žiadny zo senzorov a modulov pripojených k doskám Arduino

[35] Zdroj: Vlastný zdroj

a Raspberry, sledovanie pokračuje, kým sa neobjaví signál na zapnutie jedného zo senzorov a modulov.

```
def temperature():
    db.child("console-text").set("measuring", user['idToken'])
    file_name = get_date()
    bus = smbus.SMBus(1)
    sensor = MLX90614(bus, address=0x5A)
    i = 1500
    with open("/home/pi/Diplom/Data/teplota/{0}.log".format(file_name), "w") as f:
        while i:
            print("Object Temperature :", sensor.get_object_1())
            f.write("{0}\n".format(sensor.get_object_1()))
            i -= 1
    bus.close()
    bus = None

    temp_value = 0
    temp_count = 0
    db.child("console-text").set("loading", user['idToken'])
    with open("/home/pi/Diplom/Data/teplota/{0}.log".format(file_name), "r") as f:
        for line in f:
            temp_value += float(line)
            temp_count += 1
    print(temp_count)
    db.child("teplota").set(round((temp_value / temp_count), 1), user['idToken'])
    print(temp_value)
    db.child("console-text").set("ready", user['idToken'])
    countdown(int(30), 1)
def find():
    db.child("console-text").set("waiting", user['idToken'])
    while True:
        arduino = arduino_connect()
        if arduino != 1:
            rawserial = arduino.readline()
            cookedserial = rawserial.decode('utf-8').strip('\r\n')
            if cookedserial != "ziadne data":
                if cookedserial == "start ecg":
                    arduino_module("ecg")
                elif cookedserial == "start gsr":
                    arduino_module("gsr")
            else:
                check_raspberry()
        else:
            check_raspberry()
```

Obr.36 Kód metódy temperature() a find()[36]

Metóda *countdown()* implementuje funkciu časovača. Metóda prijíma dva parametre, prvý je čas v sekundách, druhý je režim práce. To znamená, že ak dôjde k volaniu metódy *countdown(30,1)*, potom po 30 sekundách bude vyvolaná metóda sledovania *find()*. Ak je druhá hodnota, ktorá je v príklade uvedená ako jednotka (1), nahradená iným číslom, potom po uplynutí času program zastaví svoju prácu.

Po spustení skriptu sa stav systému automaticky aktualizuje na stav, čo znamená, že momentálne neprebieha žiadne meranie. Potom začne pracovať časovač. Po dokončení sa automaticky zavolá metóda sledovania *find()*. Je to nevyhnutné, aby po spustení používateľ nemusel nič konfigurovať. Namiesto toho program automaticky začne sledovať senzory a moduly.

[36] Zdroj: Vlastný zdroj

```
def countdown(t, value):
    db.child("console-text").set("waiting", user['idToken'])
    while t:
        mins, secs = divmod(t, 60)
        timer = '{:02d}:{:02d}'.format(mins, secs)
        print(timer)
        time.sleep(1)
        t -= 1
    if value == 1:
        find()


find()
```

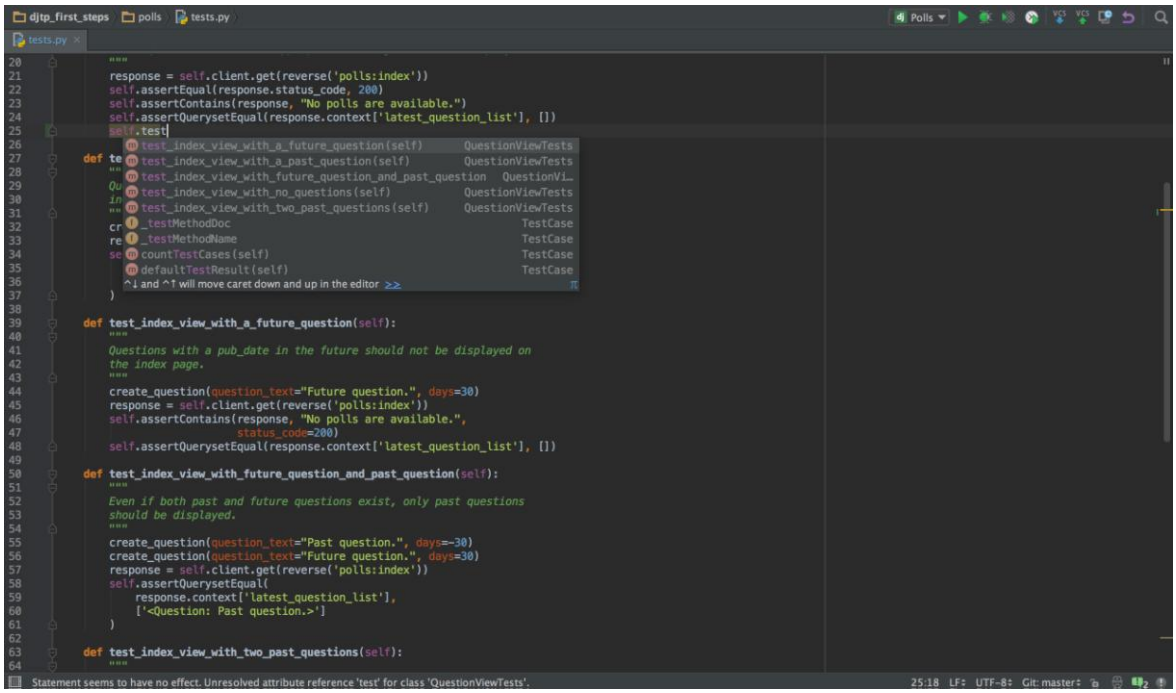
Obr.37 Kód metódy countdown()[37]

4.4.3 Vývoj webovej aplikácie

Ako editor kódu na vytvorenie webovej aplikácie bol vybraný program PyCharm. PyCharm je integrované vývojové prostredie pre programovací jazyk Python. Poskytuje nástroje na analýzu kódu, grafický debugger a podporuje vývoj webových aplikácií Django. PyCharm je vyvíjaný českou spoločnosťou JetBrains založenou na IntelliJ IDEA. PyCharm môže bežať na operačných systémoch Windows, Mac OS X a Linux. Vďaka PyCharm je vývoj čo najproduktívnejší vďaka funkciám na dokončenie a analýzu kódu, okamžité zvýraznenie chyby a rýchle opravy. Automatické refaktoringy pomáhajú vývojárovi efektívne upravovať kód a pohodlná navigácia umožňuje okamžitú navigáciu v projekte. Výhody PyCharm:

- Rýchle a bezpečné refaktorovanie kódu. Refaktorovanie je proces údržby, pri ktorom dochádza k zmene štruktúry a návrhu existujúceho kódu, bez zmeny vo funkcionalite či správaní programu resp. systému.
- Prehľadné rozhranie pre prácu s GIT.
- Vytváranie a otváranie existujúcich projektov je veľmi jednoduché.
- Vysoká rýchlosť práce. Program PyCharm sa spustí niekoľkonásobne rýchlejšie a pri otváraní veľkých súborov nespomalí.

[37] Zdroj: Vlastný zdroj

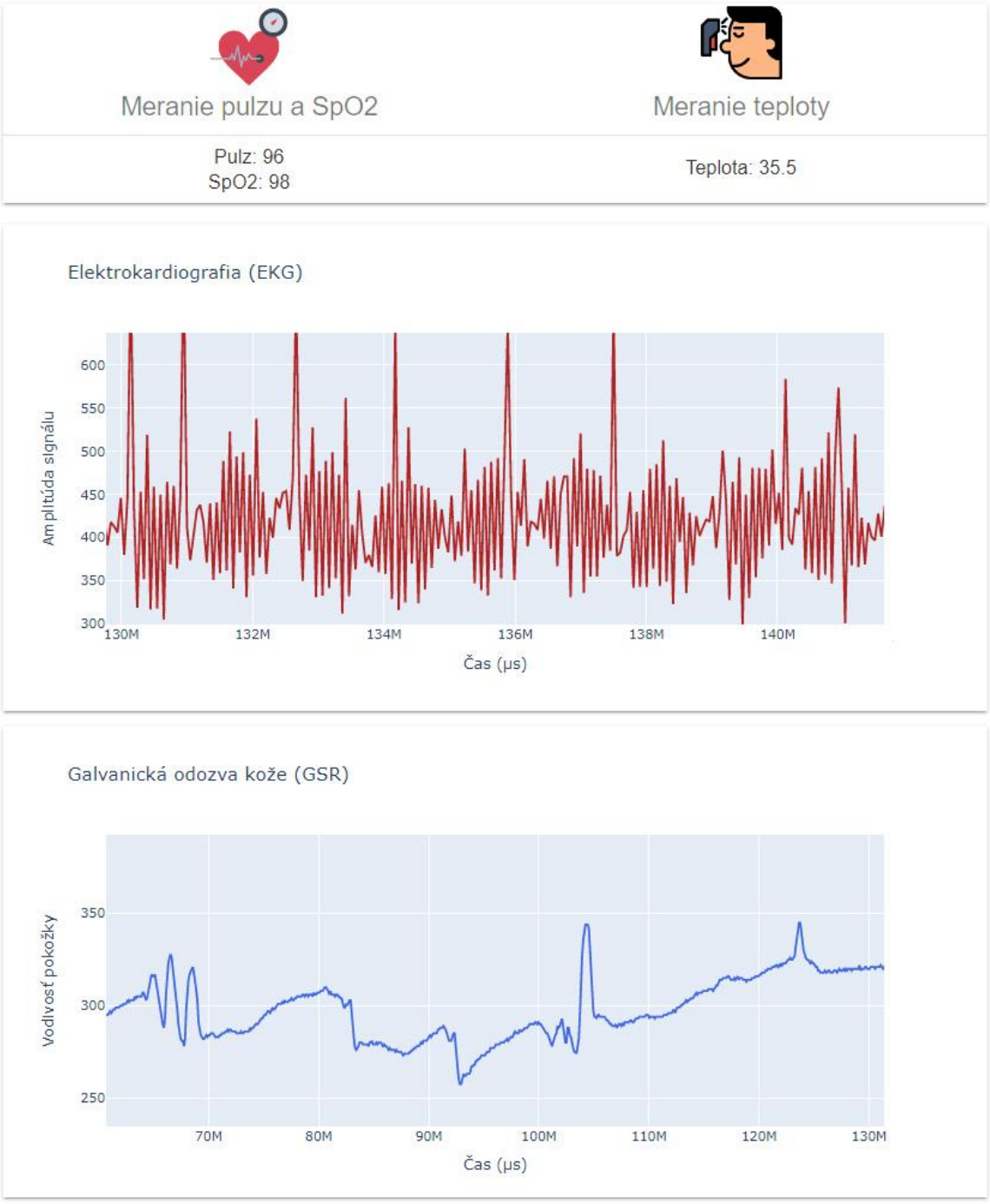


Obr.38 Hlavná obrazovka PyCharm IDE[38]

Základ webovej aplikácie bol vytvorený pomocou Python a Django. Pre vytvorenie a spracovanie vonkajšieho vzhľadu boli použité HTML a CSS. JavaScript bol použitý na získanie hodnoty stavu systému z databázy a potom na vytvorenie vyskakovacieho okna pre konkrétny stav systému. Ďalej na obrázku číslo 39 je zobrazený vonkajší vzhľad aplikácie. Webová aplikácia je rozdelená do 3 blokov. Prvý zobrazuje informácie týkajúce sa teploty, srdcového rytmu a SpO2. Druhý blok obsahuje graf, ktorý je zodpovedný za vykreslenie EKG. Tretí blok obsahuje graf, ktorý je zodpovedný za vykreslenie GSR.

Informácie zo senzorov v prvom bloku sú uložené v značke textového bloku <*p*> a obrázky sú spojené pomocou tagu <*img*>, ktorý je určený na zobrazovanie obrázkov v grafickom formáte GIF, JPEG alebo PNG na webovej stránke.

[38] Zdroj: https://www.jetbrains.com/pycharm/img/screenshots/simpleLook@2x.jpg



Obr.39 Hlavná obrazovka webovej aplikácie[39]

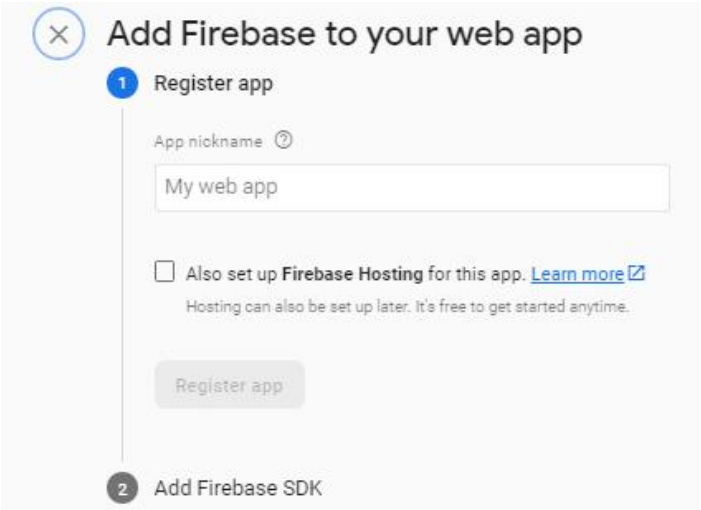
Vyvinutá webová aplikácia má tiež pomocné vyskakovacie okno, ktoré obsahuje aktuálny stav systému. Celkovo existujú 4 stavy, ktoré popisujú, či sa momentálne vykonávajú nejaké merania, či sú zapnuté snímače, či sa odosielajú údaje do databázy a či sa meranie skončilo. Každý stav má svoju vlastnú farebnú schému pre lepšiu viditeľnosť. Keď teda prebieha čakanie na meranie, okno je modré. V okamihu, keď prebieha meranie,

[39] Zdroj: Vlastný zdroj

okno je oranžové. Keď sa údaje odosielajú do databázy, okno je červené. A keď je meranie ukončené, okno sa zmení na zelené.

Obr.40 Vzhľad vyskakovacích okien so stavom systému[40]

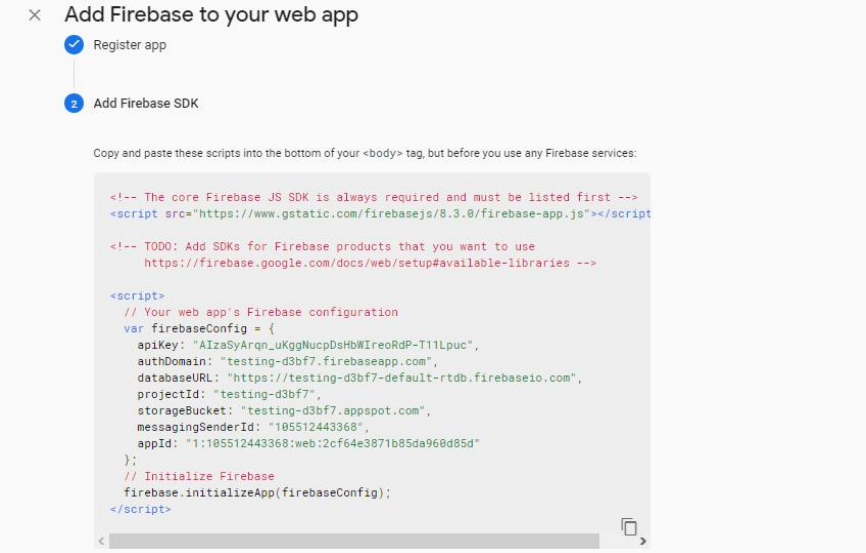
Na pripojenie k databáze je nutné v konzole pre správu databázy Firebase Realtime Database spočiatku zaregistrovať svoju novú webovú aplikáciu. Je to nevyhnutné na to, aby získala parametre, ktoré sú potrebné na pripojenie. Na registráciu aplikácie je nutné zadať názov webovej aplikácie a registráciu potvrdiť kliknutím na tlačidlo.



Obr.41 Okno registrácie webových aplikácií v konzole Firebase Realtime Database[41]

Ďalej dostaneme inštrukciu, ktorá popisuje postupnosť, ktorú je potrebné dodržať, aby bolo možné prepojiť vyvinutú webovú aplikáciu a databázu. Všetko, čo je potrebné urobiť, je iba skopírovať výsledný kód do vyvinutej aplikácie, ako je to znázornené na obrázku číslo 42.

[40] Zdroj: Vlastný zdroj
[41] Zdroj: Vlastný zdroj



Obr.42 Výsledný kód pre pripojenie k databáze[42]

Ďalej na obrázku číslo 43 je zobrazená časť HTML kódu, ktorá je zodpovedná za vytvorenie vyskakovacieho okna, ktoré zobrazuje aktuálny stav systému a je zodpovedná za vytvorenie sekcie, v ktorej sú vytvorené grafy. Na vytváranie grafov sa používa značka <*section*>, ktorá definuje časť dokumentu. Túto značku možno použiť aj pre blok správ, kontaktné informácie, textové kapitoly, karty v dialógovom okne atď.

```
<div id="myModal" class="modal">
    <div class="modal-content">
        <div class="modal-header" id="modal-header">
            <span class="close">×</span>
            <h2 id="banner_h2"></h2>
        </div>
        <div class="modal-body">
            <p id="banner_p"></p>
        </div>
    </div>
</div>
<br/>

<div class="container">
    <section class="section--center mdl-grid mdl-grid--no-spacing mdl-shadow--2dp">
        <div class="col-md-12" style="...">{{ plot_ecg|safe }}</div>
    </section>
</div>
```

Obr.43 Kód pre vytvorenie vyskakovacích okien a grafu[43]

Po umiestnení prijatého kódu počas registrácie do skriptu nami vyvinutej webovej aplikácie bolo potrebné vytvoriť a inicializovať exemplár aplikácie Firebase. Na obrázku číslo 44 sa nachádza časť kódu, ktorá je zodpovedná za pripojenie a zobrazenie informácií týkajúcich sa srdcového rytmu, SpO2 a teploty v značkách <*p*>. Vzhľadom na to, že

[42] Zdroj: Vlastný zdroj
[43] Zdroj: Vlastný zdroj

rovnaký programovací jazyk sa používa na vytvorenie programu pre dosku Raspberry Pi a pre vývoj webovej aplikácie, bola použitá rovnaká knižnica na pripojenie k databáze, ktorá je zobrazená na obrázku 32. Informácie sa aktualizujú automaticky v reálnom čase. To znamená, že na získanie najnovších údajov používateľ nepotrebuje obnovovať stránku po každom novom meraní. Namiesto toho vyvinutý kód po každom meraní automaticky aktualizuje webovú aplikáciu.

```
def get_firebase_pulse(self):
    get_pulse = "Pulz: " + str(self.db.child("heart_rate/tep_srdca").get().val())
    return get_pulse
```

Obr.44 Kód pre príjem a zobrazenie hodnoty srdcového tepu[44]

Ďalšia celkom dôležitá časť kódu je zodpovedná za zobrazenie stavu systému. Od začiatku je potrebné identifikovať vetvu databázy Firebase Realtime a premennú, v ktorej je nutné kontrolovať aktuálnu hodnotu. Keďže máme 4 systémové pracovné stavy, bolo potrebné použiť podmienený operátor if-else na identifikáciu prijatého stavu. Keď bol zistený stav systému, potom sa spustí konkrétna metóda, ktorá je zodpovedná za fungovanie konkrétneho stavu. V tejto metóde dôjde k zatvoreniu vyskakovacieho okna predchádzajúceho stavu v prípade, že ho používateľ nezavrel. Každý stav okrem toho, že má svoju vlastnú farbu, má aj svoj vlastný text, preto ďalším krokom je vloženie textu do značiek *<h2>* a *<p>*, ktoré sú v bloku vyskakovacieho okna. Premenná *reloadCount* je potrebná na sčítanie počtu odpracovaných stavov systému počas merania. Po každom stave sa hodnota premennej zvýši o 1. Keďže systém má celkovo 4 stavy, pri spustení funkcie zodpovednej za prevádzku posledného stavu skontroluje, či boli správne spracované predchádzajúce stavy. Ak áno, zobrazí sa posledný stav a stránka sa automaticky obnoví. Na zatvorenie vyskakovacieho okna boli vytvorené dve možnosti. Prvá možnosť je pomocou tlačidla zatvorenia v pravom hornom rohu okna a druhou možnosťou je kliknutie myšou kdekoľvek vo webovej aplikácii.

[44] Zdroj: Vlastný zdroj

```
firebase.initializeApp(firebaseConfig);
var firebaseRef = firebase.database();
var consoleRef = firebaseRef.ref().child('console-text');

consoleRef.on("value", function (snapshot) {
    if (snapshot.val() == "waiting") {
        waiting_banner();
    } else if (snapshot.val() == "measuring") {
        measuring_banner();
    } else if (snapshot.val() == "loading") {
        loading_banner();
    } else if (snapshot.val() == "ready") {
        ready_banner();
    }
});
span.onclick = function () {
    modal.style.display = "none";
}

window.onclick = function (event) {
    if (event.target == modal) {
        modal.style.display = "none";
    }
}

function waiting_banner() {
    reloadCount += 1;
    modal.style.display = "none";
    document.getElementById("modal-header").style["background-color"] = "#1E88E5";
    document.querySelector('#banner_h2').innerHTML = "Čakanie na meranie";
    document.querySelector('#banner_p').innerHTML = "Pre pokračovanie zapnite jeden z modulov";
    modal.style.display = "block";
}
```

Obr.45 Kód pre príjem a zobrazenie stavu systému[45]

Nasledujúci kód je zodpovedný za kreslenie grafov GSR a EKG. Prvým krokom je vytvorenie dvoch zoznamov, ktoré sú potrebné na uloženie prijatého času merania a hodnôt merania. Ďalej sa hodnoty načítajú z databázy, nasleduje filtrovanie v slučke a zápis do predtým vytvorených zoznamov. Potom sa každej z osí priradia hodnoty, nastaví sa typ grafu, nastaví sa jeho farba, nastavia sa minimálne a maximálne hodnoty pre os Y, zaznamená sa názov každej osi a názov celého grafu. A ako návratova hodnota je pripravený graf.

[45] Zdroj: Vlastný zdroj

```
def get_firebase_gsr(self):
    gsr_time = []
    gsr_value = []
    try:
        all_users = self.db.child("gsr").get()
        for user in all_users.each():
            gsr_time.append(user.val()["time"])
            gsr_value.append(user.val()["value"])

    except:
        gsr_time.append(0)
        gsr_value.append(0)

    trace1 = go.Scatter(
        x=gsr_time,
        y=gsr_value,
        mode='lines',
        line=dict(color='royalblue')
    )

    data = [trace1]
    fig = go.Figure(data=data, layout_yaxis_range=[0, 500])
    fig.update_layout(title='Galvanická odozva kože (GSR)',
                      xaxis_title='Čas (µs)',
                      yaxis_title='Vodivosť pokožky',
                      )
    plot_div = plot(fig, output_type='div', include_plotlyjs=False)
    return plot_div
```

Obr.46 Kód pre príjem a zobrazenie hodnôt GSR[46]

Metóda *get_context_data()* je zodpovedná za zobrazovanie hodnôt v HTML šablóne. Šablóna obsahuje kľúče a pomocou nich je možné zobraziť graf a hodnoty z databázy. Takže každý z prvkov v šablóne je pomocou kľúča prirovnávaný k metóde, ktorá sa vracia buď ako hodnota z databázy, alebo pripravený graf.

```
def get_context_data(self, **kwargs):
    context = super(Plot1DView, self).get_context_data(**kwargs)
    context['plot_ecg'] = self.get_firebase_ecg()
    context['plot_gsr'] = self.get_firebase_gsr()
    context['spo2'] = self.get_firebase_spo2()
    context['pulse'] = self.get_firebase_pulse()
    context['temperature'] = self.get_firebase_temp()
    return context
```

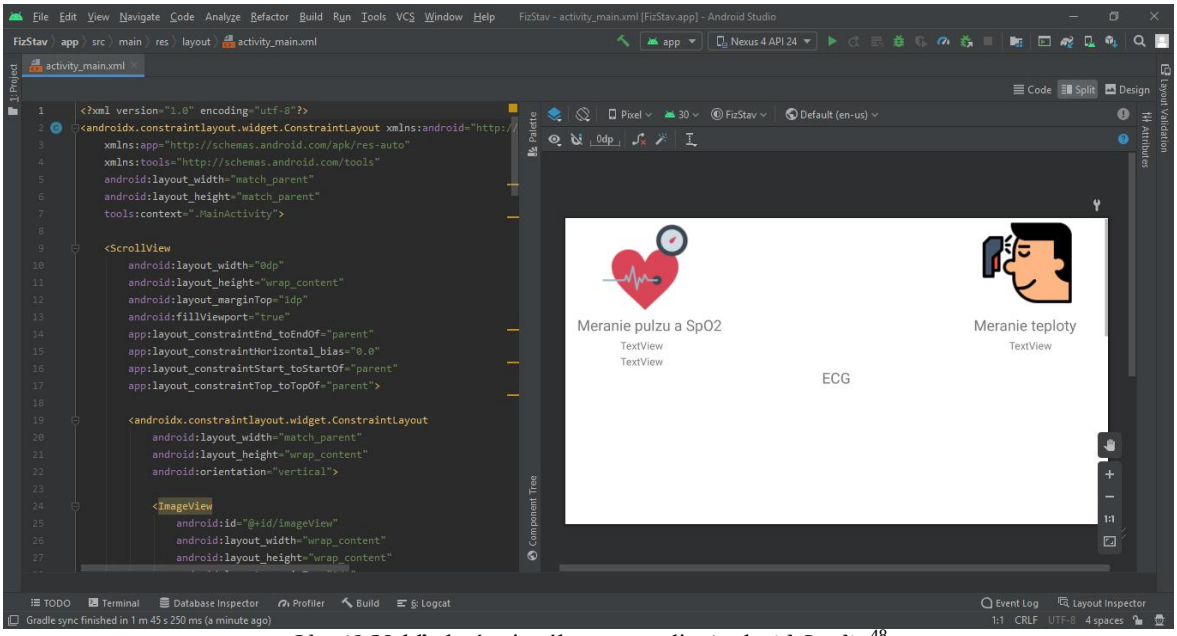
Obr.47 Kód metódy get_context_data() [47]

[46] Zdroj: Vlastný zdroj

4.4.4 Vývoj mobilnej aplikácie

Pre vývoj mobilnej aplikácie pre operačný systém Android, je potrebné spočiatku zvoliť vývojové prostredie. Pre vývoj našej mobilnej aplikácie bolo rozhodnuté použiť o ficiálny nástroj na vývoj aplikácií Android Studio.

Android Studio je nové a plne integrované vývojové prostredie (IDE) vydané spoločnosťou Google pre operačný systém Android. Vývojári vzali pomerne populárny Intellij IDEA ako základ pre svoje nové IDE, zdokonalili ho a predstavili značný zoznam inovácií. Obsahuje Software Development Kit (SDK), vývojovú súpravu, ktorá umožňuje softvérovým inžinierom vytvárať aplikácie pre konkrétny softvérový balík. Účelom tohto produktu je poskytnúť vývojárom nové nástroje na vytváranie aplikácií. Súčasťou aplikácie Android Studio je tiež optimalizovaný emulátor. Emulátor je program, ktorý umožňuje emulovať inú konzolu na osobnom počítači alebo hernej konzole. Emulátor vytvorí na pracovnej ploche digitálny analóg smartfónu, v ktorom môže vývojár program otestovať pomocou kurzora myši namiesto prsta. Medzi emulátorovými programami na spúšťanie a testovanie aplikácií pre Android je najbežnejší program Android Virtual Device (AVD), ktorý je dodávaný s balíkom SDK.



Obr.48 Vzhľad vývojového prostredia Android Studio[48]

Android Studio umožňuje vývojárovi vidieť všetky vizuálne zmeny aplikácie v reálnom čase. Umožňuje tiež vývojárovi zistiť, ako bude vyvíjaná aplikácia súčasne

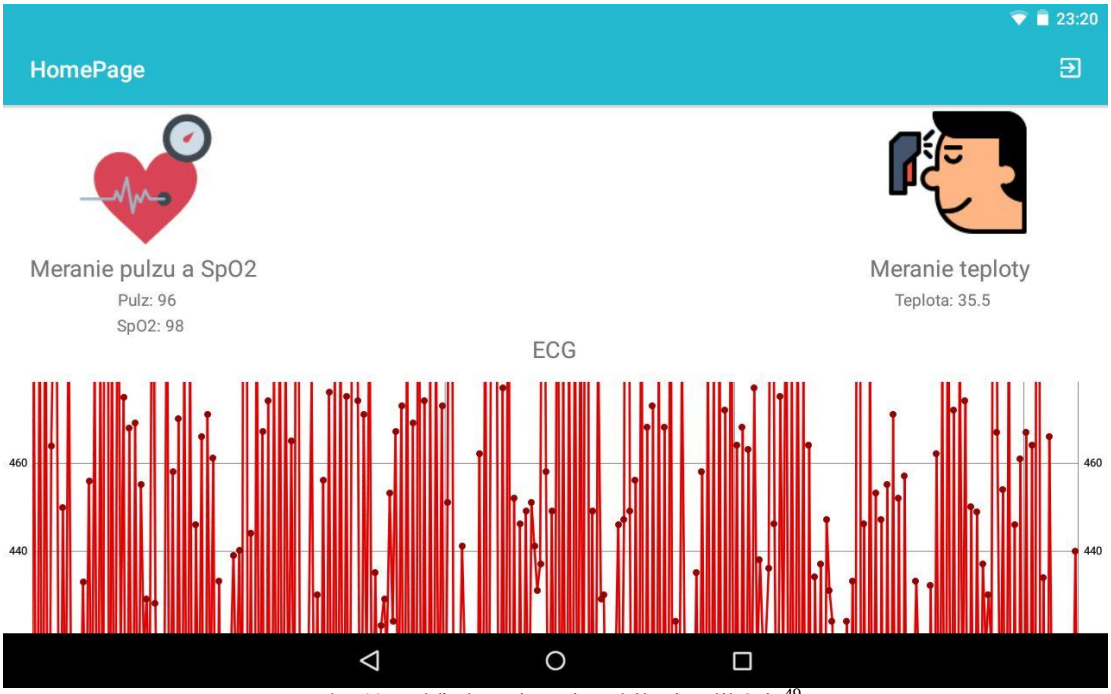
[47] Zdroj: Vlastný zdroj
[48] Zdroj: Vlastný zdroj

vyzerať na rôznych zariadeniach, s rôznymi verziami operačného systému Android, s rôznymi nastaveniami a rozlíšením obrazovky.

Vývojové prostredie má množstvo pozitívnych vlastností. Nižšie sú uvedené niektoré z nich:

- Android Studio je zadarmo na stiahnutie a použitie v systémoch Windows, Mac a Linux.
- Android Studio obsahuje nástroje na vývoj riešení pre smartphony a tablety, ako aj nové technologické riešenia pre Android TV, Android Wear, Android Auto, Glass a ďalšie kontextové moduly.
- V centre pracovného postupu Android Studio je koncepcia nepretržitej integrácie, ktorá umožňuje okamžite odhaliť existujúce problémy.
- Vo vývojovom prostredí je možné pripojiť rôzne API ako Google Play, Android Pay a Health.
- K dispozícii je podpora pre všetky platformy Android od verzie Android 1.6.

Vyvinutá mobilná aplikácia má podobný vzhľad ako vyvinutá webová aplikácia, s výnimkou vzhľadu grafov. Aplikácia má iba jednu hlavnú obrazovku, ktorá obsahuje všetky prvky.



Obr.49 Vzhľad vyvinutej mobilnej aplikácie[49]

[49] Zdroj: Vlastný zdroj

Hlavné použité rozloženie je ConstraintLayout. ConstraintLayout je kontajner, ktorý obsahuje rôzne zobrazenia. Prvým takýmto prvkom je ImageView, ktorý je navrhnutý na zobrazovanie obrázkov. V aplikácii sa nachádzajú iba 2 ImageView, ktoré obsahujú obrázky meraní srdcového rytmu a teploty. Ďalším prvkom je komponent TextView, ktorý je navrhnutý tak, aby zobrazoval text bez možnosti úpravy používateľom. Tento komponent je ideálny na zobrazenie prijatých hodnôt srdcovej frekvencie, teploty a pre zobrazenie jednoduchých nadpisov, ktoré používateľ nemusí upravovať. Ďalším prvkom je LineChart z pripojenej knižnice MPAndroidChart. Vo vyvinutej aplikácii sú iba dva také prvky, jeden je zodpovedný za vykreslenie lineárneho grafu EKG a druhý za vykreslenie lineárneho grafu GSR.

Aby sa na obrazovku aplikácie zmestilo toľko prvkov, bolo potrebné použiť ScrollView. ScrollView je zobrazenie rozloženia, ktoré je obdĺžnikovým kontajnerom s vertikálnym posuvníkom a môže obsahovať ďalší komponent väčší ako on sám. Kontajner ScrollView je navrhnutý tak, aby vytváral rolovanie pre také rozhranie, ktorého všetky prvky sa nezmestia súčasne na obrazovku zariadenia. ScrollView môže obsahovať iba jeden prvok, takže ak chce vývojár umiestniť niekoľko prvkov, musia byť umiestnené v nejakom kontajneri.

Keďže do ScrollView je možné umiestniť iba jeden prvok, všetky ImageView, TextView, LineChart sú uzavreté v LinearLayout. Kontajner LinearLayout je objektom ViewGroup, ktorý usporiada všetky svoje podriadené prvky v jednom smere: horizontálne alebo vertikálne. Všetky prvky LinearLayout sú zoradené jeden za druhým, takže vertikálny zoznam bude mať iba jeden prvok na riadok.

V hornom paneli aplikácie sa nachádza tlačidlo, po kliknutí na ktoré sa používateľovi otvorí webová aplikácia. K tomu bolo spočiatku potrebné vytvoriť rozloženie, pridať obrázok tlačidla, farbu a umiestnenie. Ďalším krokom bolo pridanie tlačidla do kódu programu a nastavenie sledovania kliknutí.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.my_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.item1:
            startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(
                    "127.0.0.1:8000/webpage/plot1d")));
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Obr.50 Kód, ktorý je zodpovedný za funkčnosť tlačidla [50]

Ďalšou dôležitou časťou je získavanie hodnôt z databázy. Na rozdiel od vývoja webovej aplikácie, pri vývoji mobilnej aplikácie vo vývojovom prostredí Android Studio je registrácia a pripojenie k databáze Firebase Realtime Database oveľa jednoduchšie. Pre registráciu a pripojenie k databáze je potrebné prejsť do ponuky Nástroje (Tools) a vybrať položku Firebase a potom zvoliť položku Realtime Database a stlačiť tlačidlo Connect to Firebase. Po kliknutí v prehliadači sa otvorí konzola Firebase, kde je potrebné vybrať existujúci projekt. Ďalej bude všetko automaticky nakonfigurované systémom. Konkrétne v konzole Firebase bude mobilná aplikácia automaticky zaregistrovaná a do kódu aplikácie budú pridané všetky potrebné knižnice na pripojenie k databáze. Jedinou vecou, ktorú treba nakonfigurovať samostatne, sú názvy premenných, z ktorých bude potrebne čítať hodnoty telesnej teploty, pulzu, EKG a GSR. Dole na obrázku číslo 51 je kód, ktorý je zodpovedný za čítanie údajov na zostavenie grafu GSR.

```
private void getGSR() {
    databaseReference4.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            gsrValue.clear();
            gsrTime.clear();
            for (DataSnapshot ds : dataSnapshot.getChildren()) {
                gsrValue.add(ds.child("value").getValue().toString());
                gsrTime.add(ds.child("time").getValue().toString());
            }
            setDataGSR();
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            System.out.println(databaseError.getMessage());
        }
    });
}
```

Obr.51 Kód metódy getGSR()[51]

Po prijatí údajov sa automaticky zavolá metóda, ktorá je zodpovedná za zobrazenie údajov na hlavnej obrazovke aplikácie. V prípade, že sa jedná o meranie teploty alebo

[50] Zdroj: Vlastný zdroj
[51] Zdroj: Vlastný zdroj

pulzu, potom sa výsledná hodnota z databázy vloží do konkrétneho TextView. Ak sa jedná o EKG a GSR, tak proces od prijatia údajov po ich zobrazenie vo forme grafov je spočiatku o 1 metódu viac. Táto metóda zodpovedá za štýl grafu, ktorý TextView na zobrazovanie srdcovej frekvencie a telesnej teploty nemá. A práve táto metóda sa vykonáva ako prvá, na rozdiel od merania pulzu a telesnej teploty, kde sa najskôr ako prvá vykonáva metóda pre získavanie dát.

Metóda, ktorá je zodpovedná za kreslenie grafov obsahuje kód, ktorý je určený na pripojenie k prvku pomocou identifikátora, ktorého úlohou je zobrazenie grafu a je zodpovedný za nastavenie veľkostí, farieb, animácií, nadpisov. Tiež v tejto metóde prebieha pripojenie vyskakovacieho okna, ktoré sa zobrazí, keď používateľ klikne na určitú značku v grafe. Toto okno má špecifický štýl a zobrazuje názov hodnoty a hodnotu tejto značky, ktorú používateľ stlačil. Ďalej sa volá nasledujúca metóda, ktorá je zodpovedná za načítanie údajov z databázy a ich zápis do poľa. Táto metóda je zobrazená na obrázku číslo 51. Posledným krokom je spustenie metódy, ktorá je zodpovedná za zobrazovanie údajov a ďalších nastavení vo forme nastavenia veľkosti, farieb pre čiary a body.

```
private void setDataGSR() {
    ArrayList<Entry> values1 = new ArrayList<>();

    for (int i = 0; i < gsrTime.size(); i++) {
        values1.add(new Entry(i, Float.parseFloat(gsrValue.get(i))));
    }
    LineDataSet set1;
    if (chartGSR.getData() != null &&
            chartGSR.getData().getDataSetCount() > 0) {
        set1 = (LineDataSet) chartGSR.getData().getDataSetByIndex(0);
        set1.setValues(values1);
        chartGSR.getData().notifyDataChanged();
        chartGSR.notifyDataSetChanged();
    } else {
        set1 = new LineDataSet(values1,  label: "GSR");
        set1.setAxisDependency(YAxis.AxisDependency.LEFT);
        set1.setColor(Color.rgb( red: 28,  green: 190,  blue: 207));
        set1.setCircleColor(Color.rgb( red: 25,  green: 145,  blue: 158));
        set1.setLineWidth(2f);
        set1.setCircleRadius(3f);
        set1.setFillAlpha(65);
        set1.setDrawCircleHole(false);

        LineData data = new LineData(set1);
        data.setValueTextColor(Color.WHITE);
        data.setValueTextSize(9f);
        chartGSR.setData(data);
    }
}
```

Obr.52 Kód metódy setDataGSR()[52]

[52] Zdroj: Vlastný zdroj

4.5 VÝSLEDKY PRÁCE A DISKUSIA

Výsledkom spojenia hardvérovej a softvérovej časti bol funkčný systém na monitorovanie základných životných funkcií pri zachovaní neinvazívnosti. Na demonštráciu funkčnosti systému, sme vykonali niekoľko rôznych testov. Výsledky testov sme preskúmali a vyhodnotili. Po vyhodnotení systému nasleduje podkapitola, v ktorej sme diskutovali o zdokonalení vyvinutého systému v budúcnosti.

4.5.1 Zhodnotenie systému

Aby sme mohli plnohodnotne využívať vytvorený systém na meranie základných fyziologických funkcií, musíme najprv splniť určité podmienky. V prvom rade sme museli zabezpečiť pripojenie mini počítača Raspberry Pi 4B k internetu cez WiFi aby systém mohol správne spolupracovať s databázou nameraných hodnôt a webovou aplikáciou. Ďalej je potrebné správne umiestniť senzory alebo elektródy, aby meranie prebehlo správne. Ďalej sa údaje odoslali do databázy s následnou vizualizáciou vo webovej a mobilnej aplikácii.

Po spustení vyvinutého programu pre Raspberry Pi automaticky prebieha sledovanie zapojenia jedného zo snímačov. Ihneď po zapnutí snímača sa spustí metóda, ktorá je zodpovedná za jeho činnosť. Po ukončení merania sa prijaté dáta odošlú do databázy Firebase Realtime Database. V intervale od začiatku merania až po zobrazenie údajov vo webovej aplikácii je používateľ doprevádzaný vyskakovacím oknom, ktoré obsahuje informácie o aktuálnom stave systému. Po odoslaní údajov metóda končí svoju prácu. Potom sa časovač zapne na 30 sekúnd a po jeho odpočítaní začne doska znova monitorovať zapojenie ktoréhokoľvek zo senzorov. Vyvinutý kód pre webové a mobilné aplikácie automaticky načíta údaje z databázy pri spustení a potom ich zobrazí ako hodnoty alebo grafy. Pre lepšie pochopenie je možné každý graf zväčšiť alebo zmenšiť a po kliknutí na ľubovoľný bod v grafe sa zobrazia jeho hodnoty. V prípade webovej aplikácie je možné výsledný graf alebo fragment uložiť ako obrázok.

Aby sme skontrolovali, či systém správne funguje, boli vykonané testy na overenie správneho fungovania. Konkrétne boli simulované situácie, ktoré sa môžu stať pri bežnom a nie celkom vhodnom používaní systému. Boli to nasledujúce situácie:

- Zapnúť systém a na dlhú dobu nezapínať senzory.
- Ak po meraní nevypneme senzor.
- Ak zapneme modul EKG bez pripojených k modulu elektród.

- Ak zapneme senzor a počas merania ho vypneme. Bolo dôležité pozrieť sa na správanie webovej aplikácie, mobilnéj aplikácie a systému zvlášť.
- Ak nepretržite robíme merania bez vypnutia.

Bolo dôležité vziať do úvahy situáciu, keď počas merania došlo k vypnutiu snímača alebo modulu. Konkrétne modul elektrokardiogramu (EKG) a galvanickej odozvy pokožky (GSR). Dôvodom je to, že počas spustenia funkcie zodpovednej za ich funkčnosť, údaje starého merania sa automaticky vymažú. Preto bolo dôležité skontrolovať fungovanie webovej a mobilnej aplikácie. Keď údaje, ktoré sú potrebné na vykreslenie grafov nie sú uložené v databáze.

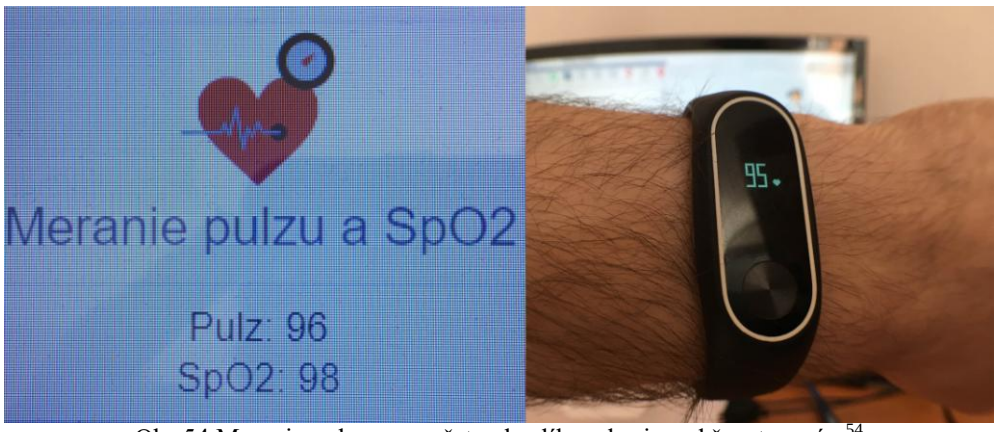
Zo všetkých testovaných situácií sa objavili iba 2 chyby, ktoré boli okamžite opravené. Prvá chyba bola v odosielaní údajov z dosky Raspberry Pi do databázy. Preto sme sa rozhodli vymeniť starú knižnicu firebase za novú pyrebase. Po výmene knižnice bolo urobených veľa testov a problém s pripojením zmizol. Ďalšou chybou bolo zobrazenie grafov EKG a GSR. Na zobrazovanie grafov sa pôvodne používala JavaScript knižnica Chart.js. Problém bol v tom, že knižnica nie je vhodná na zobrazovanie veľkého množstva údajov naraz. Napríklad iba pre jeden GSR graf knižnica načítavala údaje približne 10 minút, čo je neprijateľne dlhé čakanie. Taktiež sa po dlhom načítaní stratila schopnosť zväčšovať a zmenšovať graf. Podobný problém nebol v mobilnej aplikácii. Z tohto dôvodu bolo na vyriešenie tohto problému potrebné vymeniť knižnicu. Využili sme novú knižnicu plotly, ktorá má bohatšiu funkčnosť a ktorá dokáže vytvárať grafy z veľkého súboru údajov. Po výmene starej knižnice, bola nová knižnica dôkladne otestovaná a problém s dlhým načítaním údajov zmizol.

Tiež bol vykonaný test na porovnanie údajov získaných z nášho vyvinutého systému a ďalších meracích prístrojov. Test je potrebný na pochopenie správnosti merania fyziologických parametrov nami vyvinutým systémom. V rámci porovnávacieho testu bolo vykonané meranie pulzu, množstva kyslíka v krvi a telesnej teploty. Merania boli vykonané pred športovaním a po športových aktivitách.

Spočiatku, pred športovaním, sa teplota tela merala pomocou vyvinutého systému a elektronického teplomeru. Rozdiel medzi týmito dvoma typmi merania je tolerancia merania. Preto môžeme povedať, že merania dávali rovnakú hodnotu.

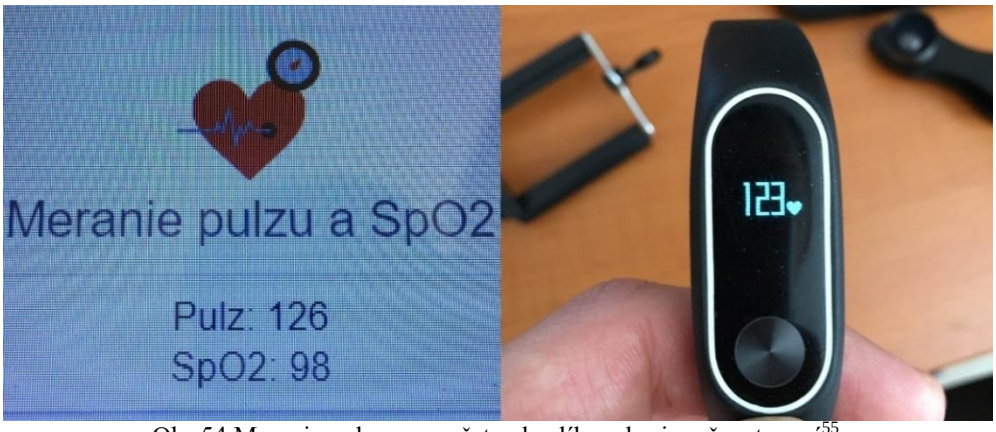
Obr.53 Meranie teploty pred športovaním[53]

To isté možno povedať o meraní srdcového rytmu pred športovaním. Spočiatku pred športovaním pulz bol meraný pomocou vytvoreného systému a fitnes náramku na zápästí. Aj tu je rozdiel medzi týmito dvoma typmi merania, čo je prípustná chyba merania.



Obr.54 Meranie pulzu a množstva kyslíka v krvi pred športovaním[54]

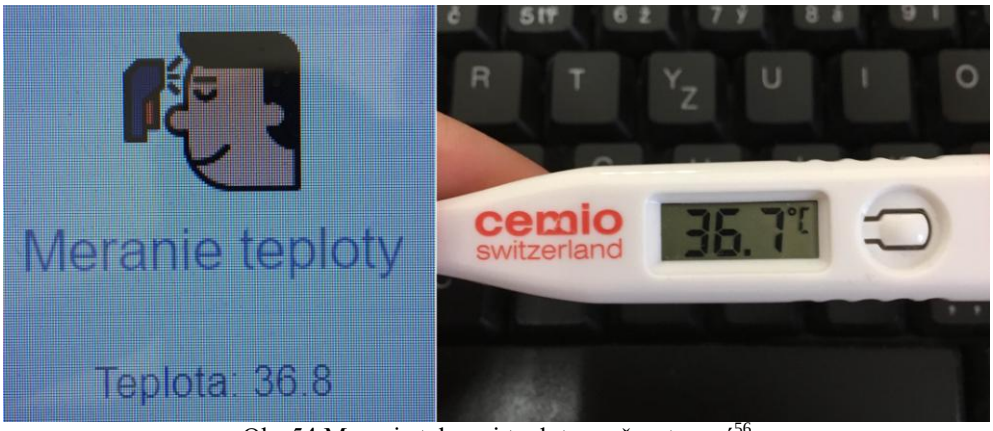
Po športovaní zostávajú hodnoty na rovnakej úrovni. Pokiaľ ide o meranie srdcového rytmu, medzi týmito dvoma meraniami je tiež minimálny rozdiel, ktorým je prípustná chyba merania.



Obr.54 Meranie pulzu a množstva kyslíka v krvi po športovaní[55]

[53] Zdroj: Vlastný zdroj
[54] Zdroj: Vlastný zdroj

Meranie telesnej teploty po športovaní tiež ukázalo malý rozdiel, ktorým je prípustná chyba merania. Preto môžeme povedať, že merania dávali rovnakú hodnotu.



Obr.54 Meranie telesnej teploty po športovaní[56]

4.5.2 Vylepšenie systému v budúcnosti

Náš vyvinutý systém na monitorovanie základných životných funkcií momentálne spĺňa všetky základné funkcie, ktoré boli deklarované v diplomovej práci. Chceli by sme však aspoň teoreticky navrhnúť niektoré funkcie, ktoré by sme v budúcnosti chceli pridať do nášho systému.

Pridanie funkcie diagn ostiky cukrovky, a to vývoj a implementácia modulu glukometra do systému. Táto funkčnosť je dôležitá pre pacientov diagnostikovaným diabetom, ktorí musia kontrolovať hladinu glukózy v krvi kvôli kontrole glykémie. Hlavnou úlohou pri vývoji tohto modulu je uchovanie neinvazívneho merania.

Ďalším možným vylepšením je pridanie monitorovania spánku. Monitorovanie spánku poskytuje cenné informácie o kvalite odpočinku človeka a pomáha demonštrovať vplyv faktorov na spánok, ako je napríklad jedlo, denný program atď. Spánok je dôležitá prirodzená činnosť, ktorá pomáha predchádzať únave, stresu a udržuje ľudí zdravými. Nedostatok spánku môže mať vplyv na našu pamäť, imunitný systém, kognitívne funkcie, schopnosť učiť sa a tým viesť k zníženiu produktivity.

[55] Zdroj: Vlastný zdroj
[56] Zdroj: Vlastný zdroj

ZÁVER

Cieľom diplomovej práce bolo navrhnúť a vytvoriť systém pre identifikovanie a klasifikáciu emocionálneho stavu používateľa pomocou dostupných fyziologických funkcií. Hlavnými podcieľmi bolo zoznámenie sa s technológiou IoT, zoznámenie sa s mini počítačmi Raspberry Pi a Arduino Nano, výber senzorov a modulov potrebných pre merania, vytvorenie funkčného systému pre merania, vytvorenie webovej a mobilnej aplikácie, testovanie a vyhodnotenie dát.

V teoretickej časti diplomovej práce sme objasnili pojem internet vecí zo všeobecného hľadiska a pozerali sme sa na históriu vzniku IoT. Ďalej sme sa venovali segmentom, v ktorých sa IoT využíva a popísali sme jeho použitie v týchto jednotlivých segmentoch. Následne sme popísali základné fyziologické funkcie človeka. Konkrétne popis, charakteristika merania a dôvod, prečo je dôležité merať. V tejto práci bolo popísané meranie pulzu a množstva kyslíka v krvi, meranie telesnej teploty, elektrokardiogram a galvanická odozva pokožky.

V praktickej časti diplomovej práce sme sa venovali samotnému meraciemu systému a tvorbe webovej a mobilnej aplikácie. Pri návrhu systému bolo nutné myslieť na to, aby systém dostával správne údaje a mohol spoľahlivo pracovať. Z ponuky dostupných mini počítačov sme si vybrali Raspberry Pi 4B a Arduino Nano. Popísali sme konkrétne mini počítače a vybrané snímače, ktoré boli použité na merania. Následné sme vytvorili schému zapojenia a diagramy funkčnej prevádzky systému. V ďalšej časti sme sa venovali programovaniu dosiek Raspberry Pi a Arduino Nano, ktoré sú zodpovedné za čítanie údajov zo senzorov a za ich odosielanie do databázy. Ako databáza na ukladanie údajov, odkiaľ budú následne odoslané do webovej a mobilnej aplikácie, bolo rozhodnuté použiť riešenie Firebase Realtime Database od spoločnosti Google. Vytvorili sme sme webovú a mobilnú aplikáciu pre operačný systém Android, pomocou ktorých je možné zobraziť prijaté dáta zo senzorov a modulov. Počas písania programu pre dosku Raspberry Pi a vytvárania webovej stránky pomocou webového frameworku Django, sme sa zoznámili s programovacím jazykom Python a naučili používať naprogramované knižnice. Vyššie uvedené podcieľe sa taktiež podarilo splniť.

Výsledkom diplomovej práce je funkčný merací systém, ktorý dokáže merať pulz, množstvo kyslíka v krvi, elektrokardiogram a galvanickú odozvu pokožky. Senzory sa zapínajú pomocou tlačidiel, následne sa vo webovej aplikácii objaví vyskakovacie okno, čo znamená, že meranie začalo. Celkovo sa na prednom paneli nachádzajú 4 tlačidlá,

z ktorých každé je zodpovedné za zapnutie a vypnutie konkrétneho snímača. Pomocou webovej a mobilnej aplikácie, sa používateľ môže pozrieť na výsledok merania. Do databázy sa ukladajú iba posledné merania, predchádzajúce merania sa ukladajú ako logovací súbor na doske Raspberry Pi.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

ABHISHEK, L. RISHI BARATH, B. 2019. Automation in Agriculture Using IOT and Machine Learning. [online]. *IJITEE*. [cit.2021-03-07]. ISSN 2278-3075. Dostupné na internete: <https://www.ijitee.org/wp-content/uploads/papers/v8i8/H6651068819.pdf>.

BEXTON, R. S., VALLIN, H. O. a CAMM, A. J. (1986) "Diurnal variation of the QT interval –influence of the autonomic nervous system". [online]. Dostupné na internete: <https://heart.bmj.com/content/heartjnl/55/3/253.full.pdf>.

BELLO, O. et al. 2016. Network layer inter-operation of Deviceto-Device communication technologies in Internet of Things (IoT). *Ad Hoc Networks*. doi: 10.1016/j.adhoc.2016.06.010.

DZEDZICKIS, A., KAKLAUSKAS, A. a BUCINSKAS, V. 2020. Human emotion recognition: Review of sensors and methods. *Sensors* (Basel, Switzerland). doi: 10.3390/s20030592.

HOFFMAN, J. 2018. *Mastering Arduino: A project-based approach to electronics, circuits, and programming.* Packt Publishing. 372 s. ISBN 978-1-78883-058-4.

MADAKAM, S. et al. 2015. Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*. doi: 10.4236/jcc.2015.35021.

MATTERN, F. FLOERKEMEIER, C. 2010. From the Internet of Computers to the Internet of Things. [online]. Dostupné na internete: <https://vs.inf.ethz.ch/publ/papers/Internet-of-things.pdf>.

MONK, S. 2012. *Programming Arduino: getting started witch sketches*. McGraw-Hill. 176 s. ISBN 978-0-07-178422-1.

MONK, S. 2014. *Programming Arduino Next Steps: Going Further with Sketches*. McGraw-Hill. 288 s. ISBN: 978-0-07-183025-6.

NEMATI, E., DEEN, M. J. a MONDAL, T. (2012) "A wireless wearable ECG sensor for long-term applications". V *IEEE Communications Magazine*, s. 36-43, doi: 10.1109/MCOM.2012.6122530.

PERRONE, G. et al. 2019. The Internet of things: a survey and outlook". *Sensors in the Age of the Internet of Things: Technologies and applications*. doi: 10.1049/PBCE122E_ch1.

RATTANYU, K., & MIZUKAWA, M. (2011) "Emotion Recognition Based on ECG Signals for Service Robots in the Intelligent Space During Daily Life". *J. Adv. Comput. Intell. Intell. Informatics.* 15. 582-591.

SETHI, P. a SARANGI, S. 2017. Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering*. Hindawi Limiteds. doi: 10.1155/2017/9324035.

TSIATSIS, V. et al. 2018. *Internet of Things: Technologies and Applications for a New Age of Intelligence*. Academic Press. [cit.2021-03-05]. ISBN 978-0-12-814435-0.

UDOVIČIĆ, G. et al. 2017. Wearable Emotion Recognition system based on GSR and PPG signals. V MMHealth 2017 - *Proceedings of the 2nd International Workshop on Multimedia for Personal Health and Health Care, co-located with MM 2017*. doi: 10.1145/3132635.3132641.

VERMESAN, O. a FRIESS, P. 2013. Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. [online]. *River Publishers Series in Communication*. ISBN 978-87-92982-73-5.
Dostupné na internete: <http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf>.

VODA, Z. a tým HW Kitchen. 2015. *Sprievodca svetom Arduina*. Martin Stří, Bučovice. 240s. ISBN 978-80-87106-90-7.

XIEFENGG, C. et al. 2019. Heart sound signals can be used for emotion recognition. *Scientific Reports*. doi: 10.1038/s41598-019-42826-2.

ZONG, C. a CHETOUANI, M. 2009. Hilbert-Huang transform based physiological signals analysis for emotion recognition". In: *IEEE International Symposium on Signal Processing and Information Technology. ISSPIT 2009*. doi: 10.1109/ISSPIT.2009.5407547.

ZOZNAM PRÍLOH

Príloha A

```
class Plot1DView(TemplateView):
    template_name = "index.html"

    config = {
        "apiKey": "AIzaSyArqn_uKggNucpDsHbWIreoRdP-T11Lpuc",
        "authDomain": "testing-d3bf7.firebaseapp.com",
        "databaseURL": "https://testing-d3bf7-default-rtdb.firebaseio.com",
        "storageBucket": "testing-d3bf7.appspot.com",
        "serviceAccount": "D://Dima/Projects/django projects/diplomProject/testing.json"
    }
    firebase = pyrebase.initialize_app(config)

    auth = firebase.auth()

    user = auth.sign_in_with_email_and_password("maximus2933@gmail.com",
"testtest2020")
    db = firebase.database()

    def get_firebase_gsr(self):
        gsr_time = []
        gsr_value = []
        try:
            all_users = self.db.child("gsr").get()
            for user in all_users.each():
                gsr_time.append(user.val()["time"])
                gsr_value.append(user.val()["value"])

        except:
            gsr_time.append(0)
            gsr_value.append(0)

        trace1 = go.Scatter(
            x=gsr_time,
            y=gsr_value,
            mode='lines',
            line=dict(color='royalblue')
        )

        data = [trace1]
        fig = go.Figure(data=data, layout_yaxis_range=[0, 500])
        fig.update_layout(title='Galvanická odozva kože (GSR)',
                          xaxis_title='Čas (µs)',
                          yaxis_title='Vodivosť pokožky',
                          )
        plot_div = plot(fig, output_type='div', include_plotlyjs=False)
        return plot_div

    def get_firebase_ecg(self):
```

```
    ecg_time = []
    ecg_value = []

    try:
        all_users = self.db.child("ecg").get()
        for user in all_users.each():
            ecg_time.append(user.val()["time"])
            ecg_value.append(user.val()["value"])

    except:
        ecg_time.append(0)
        ecg_value.append(0)

    trace1 = go.Scatter(
        x=ecg_time,
        y=ecg_value,
        mode='lines',
        line=dict(color='firebrick')
    )

    data = [trace1]

    fig = go.Figure(data=data, layout_yaxis_range=[200, 800])
    fig.update_layout(title='Elektrokardiografia (EKG)',
                      xaxis_title='Čas (µs)',
                      yaxis_title='Amplitúda signálu')
    plot_div = plot(fig, output_type='div', include_plotlyjs=False)
    return plot_div

def get_firebase_pulse(self):
    get_pulse = "Pulz: " + str(self.db.child("heart_rate/tep_srdca").get().val())
    return get_pulse

def get_firebase_spo2(self):
    get_spo2 = "SpO2: " + str(self.db.child("heart_rate/spo2").get().val())
    return get_spo2

def get_firebase_temp(self):
    get_temp = "Teplota: " + str(self.db.child("teplota").get().val())
    return get_temp

def get_context_data(self, **kwargs):
    context = super(Plot1DView, self).get_context_data(**kwargs)
    context['plot_ecg'] = self.get_firebase_ecg()
    context['plot_gsr'] = self.get_firebase_gsr()
    context['spo2'] = self.get_firebase_spo2()
    context['pulse'] = self.get_firebase_pulse()
    context['temperature'] = self.get_firebase_temp()
    return context
```

Index.html

```
{% extends "base.html" %}

{% block head %}
   {{ block.super }}
   <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
   <script src="https://code.getmdl.io/1.3.0/material.min.js"></script>
{% endblock %}

{% block content %}
   <br/>
   <div class="container">
      <div class="mdl-layout__tab-panel is-active" id="overview">
         <section class="section--center mdl-grid mdl-grid--no-spacing mdl-shadow--2dp">
            <div class="mdl-cell mdl-cell--6-col">
               <div class="mdl-card__supporting-text" style="padding: 0px; padding-
bottom: 10px; width: auto">
                  <img width="80" height="80"
src="http://domv.in.ua/test/diplom/pulse.png" class="icon_image">
                  <h4 style="margin-top: 0">Meranie pulzu a SpO2</h4>
               </div>
               <div class="mdl-card__actions" style="border-top: 1px solid rgba(0, 0, 0,
0.12)">
                  <p id="pulse_value">{{ pulse }}</p>
                  <p id="spo2_value">{{ spo2 }}</p>
               </div>
            </div>

            <div class="mdl-cell mdl-cell--6-col">
               <div class="mdl-card__supporting-text" style="padding: 0px; padding-
bottom: 10px; width: auto">
                  <img width="80" height="80"
src="http://domv.in.ua/test/diplom/temp.png" class="icon_image">
                  <h4 style="margin-top: 0">Meranie teploty</h4>
               </div>
               <div class="mdl-card__actions" style="border-top: 1px solid rgba(0, 0, 0,
0.12)">
                  <p id="temp_value">{{ temperature }}</p>
               </div>
            </div>
         </section>
      </div>
   </div>

   <div id="myModal" class="modal">
      <div class="modal-content">
         <div class="modal-header" id="modal-header">
            <span class="close">&times;</span>
            <h2 id="banner_h2"></h2>
         </div>
```

```
        <div class="modal-body">
            <p id="banner_p"></p>
        </div>
    </div>
</div>
<br/>

<div class="container">
    <section class="section--center mdl-grid mdl-grid--no-spacing mdl-shadow--2dp">
        <div class="col-md-12" style="height: 100%">{{ plot_ecg|safe }}</div>
    </section>
</div>
<br/>
<div class="container">
    <section class="section--center mdl-grid mdl-grid--no-spacing mdl-shadow--2dp">
        <div class="col-md-12" style="height: 100%">{{ plot_gsr|safe }}</div>
    </section>
</div>

<script>
    var reloadCount = 0;

    var modal = document.getElementById("myModal");

    var span = document.getElementsByClassName("close")[0];

    var firebaseConfig = {
        apiKey: "AIzaSyArqn_uKggNucpDsHbWIreoRdP-T11Lpuc",
        authDomain: "testing-d3bf7.firebaseapp.com",
        databaseURL: "https://testing-d3bf7-default-rtdb.firebaseio.com",
        projectId: "testing-d3bf7",
        storageBucket: "testing-d3bf7.appspot.com",
        messagingSenderId: "105512443368",
        appId: "1:105512443368:web:64f6ec093b68665760d85d"
    };

    firebase.initializeApp(firebaseConfig);
    var firebaseRef = firebase.database();
    var consoleRef = firebaseRef.ref().child('console-text');

    consoleRef.on("value", function (snapshot) {
        if (snapshot.val() == "waiting") {
            waiting_banner();
        } else if (snapshot.val() == "measuring") {
            measuring_banner();
        } else if (snapshot.val() == "loading") {
            loading_banner();
        } else if (snapshot.val() == "ready") {
            ready_banner();
        }
```

```
});

span.onclick = function () {
    modal.style.display = "none";

}

window.onclick = function (event) {
    if (event.target == modal) {
        modal.style.display = "none";
    }
}

function waiting_banner() {
    reloadCount += 1;
    modal.style.display = "none";
    document.getElementById("modal-header").style["background-color"] =
"#1E88E5";
    document.querySelector('#banner_h2').innerHTML = "Čakanie na meranie";
    document.querySelector('#banner_p').innerHTML = "Pre pokračovanie zapnite
jeden z modulov";
    modal.style.display = "block";
}

function measuring_banner() {
    reloadCount += 1;
    modal.style.display = "none";
    document.getElementById("modal-header").style["background-color"] =
"#FF7043";
    document.querySelector('#banner_h2').innerHTML = "Prebieha meranie";
    document.querySelector('#banner_p').innerHTML = "Prosím Vás, počkáte kým sa
meranie neskončí";
    modal.style.display = "block";
}

function loading_banner() {
    reloadCount += 1;
    modal.style.display = "none";
    document.getElementById("modal-header").style["background-color"] =
"#C62828";
    document.querySelector('#banner_h2').innerHTML = "Načítavajú sa údaje do
databázy";
    document.querySelector('#banner_p').innerHTML = "Prosím Vás, počkáte kým sa
údaje načítajú do databázy";
    modal.style.display = "block";
}

function ready_banner() {
    modal.style.display = "none";
```

```
            document.getElementById("modal-header").style["background-color"] =
"#00C853";
            document.querySelector('#banner_h2').innerHTML = "Meranie ukončené";
            document.querySelector('#banner_p').innerHTML = "Toto okno môžete zavrieť a
pokračovať ďalej";
            modal.style.display = "block";
            if (reloadCount === 3) {
                location.reload();
            }
        }
    </script>

{% endblock %}
```

Príloha B

```
package com.example.fizstav;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.WindowManager;
import android.view.animation.AnimationUtils;
import android.view.animation.RotateAnimation;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;

import com.github.mikephil.charting.charts.LineChart;
import com.github.mikephil.charting.components.Description;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.components.XAxis;
import com.github.mikephil.charting.components.YAxis;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.LineData;
import com.github.mikephil.charting.data.LineDataSet;
import com.github.mikephil.charting.highlight.Highlight;
import com.github.mikephil.charting.listener.OnChartValueSelectedListener;
import com.github.mikephil.charting.utils.ColorTemplate;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity implements
OnChartValueSelectedListener {

    private LineChart chartECG;
    private LineChart chartGSR;
```

```
private TextView retrivePulse;
private TextView retriveSpo2;
private TextView retriveTemp;

FirebaseDatabase firebaseDatabase1;
DatabaseReference databaseReference1;
DatabaseReference databaseReference2;
DatabaseReference databaseReference3;
DatabaseReference databaseReference4;

ArrayList<String> ecgValue = new ArrayList<String>();
ArrayList<String> ecgTime = new ArrayList<String>();

ArrayList<String> gsrValue = new ArrayList<String>();
ArrayList<String> gsrTime = new ArrayList<String>();

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

   ImageView imageView = findViewById(R.id.imageView);
   imageView.setImageResource(R.drawable.pulse);

   ImageView imageView2 = findViewById(R.id.imageView2);
   imageView2.setImageResource(R.drawable.temp);

   retrivePulse = findViewById(R.id.textView5);
   retriveSpo2 = findViewById(R.id.textView7);
   retriveTemp = findViewById(R.id.textView6);

   firebaseDatabase1 = FirebaseDatabase.getInstance();
   databaseReference1 = firebaseDatabase1.getReference("teplota");
   databaseReference2 = firebaseDatabase1.getReference().child("heart_rate");

   databaseReference3 = firebaseDatabase1.getReference().child("ecg");
   databaseReference4 = firebaseDatabase1.getReference().child("gsr");

   chartECG_method();
   chartGSR_method();

   getTeplota();
   getHR();
   getECG();
   getGSR();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
```

```
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.my_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.item1:
                startActivity(new Intent(Intent.ACTION_VIEW,
Uri.parse("127.0.0.1:8000/webpage/plot1d")));
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private void chartECG_method() {

        MyMarkerView mmv = new MyMarkerView(this, R.layout.custom_marker_view,
"Amplitúda signálu ");

        chartECG = findViewById(R.id.chart1);
        chartECG.setOnChartValueSelectedListener(this);

        mmv.setChartView(chartECG);
        chartECG.setMarker(mmv);

        Description description = new Description();
        description.setText("Sample");
        description.setTextSize(12f);
        chartECG.setDescription(description);

        chartECG.setTouchEnabled(true);

        chartECG.setDragDecelerationFrictionCoef(0.9f);

        chartECG.setDragEnabled(true);
        chartECG.setScaleEnabled(true);
        chartECG.setDrawGridBackground(false);
        chartECG.setHighlightPerDragEnabled(true);

        chartECG.setPinchZoom(true);

        chartECG.setBackgroundColor(Color.WHITE);

        chartECG.animateX(1500);

        Legend l = chartECG.getLegend();

        l.setForm(Legend.LegendForm.LINE);
```

```
    l.setTextSize(11f);
    l.setTextColor(Color.BLACK);
    l.setVerticalAlignment(Legend.LegendVerticalAlignment.BOTTOM);
    l.setHorizontalAlignment(Legend.LegendHorizontalAlignment.LEFT);
    l.setOrientation(Legend.LegendOrientation.HORIZONTAL);
    l.setDrawInside(false);

    XAxis xAxis = chartECG.getXAxis();
    xAxis.setTextSize(11f);
    xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
    xAxis.setTextColor(Color.BLACK);
    xAxis.setGranularityEnabled(true);
    xAxis.setDrawGridLines(true);
    xAxis.setDrawAxisLine(true);

    YAxis leftAxis = chartECG.getAxisLeft();
    leftAxis.setTextColor(Color.BLACK);
    leftAxis.setAxisMaximum(800);
    leftAxis.setAxisMinimum(0);
    leftAxis.setDrawGridLines(true);
    leftAxis.setGranularityEnabled(true);

    YAxis rightAxis = chartECG.getAxisRight();
    rightAxis.setTextColor(Color.BLACK);
    rightAxis.setAxisMaximum(800);
    rightAxis.setAxisMinimum(0);
    rightAxis.setDrawGridLines(false);
    rightAxis.setDrawZeroLine(false);
    rightAxis.setGranularityEnabled(true);

    chartECG.invalidate();
}

private void setDataGSR() {
    ArrayList<Entry> values1 = new ArrayList<>();

    for (int i = 0; i < gsrTime.size(); i++) {
        values1.add(new Entry(i, Float.parseFloat(gsrValue.get(i))));
    }
    LineDataSet set1;
    if (chartGSR.getData() != null &&
            chartGSR.getData().getDataSetCount() > 0) {
        set1 = (LineDataSet) chartGSR.getData().getDataSetByIndex(0);
        set1.setValues(values1);
        chartGSR.getData().notifyDataChanged();
        chartGSR.notifyDataSetChanged();
    } else {
        set1 = new LineDataSet(values1, "GSR");
        set1.setAxisDependency(YAxis.AxisDependency.LEFT);
        set1.setColor(Color.rgb(153, 32, 174));
```

```
            set1.setCircleColor(Color.rgb(25, 145, 158));
            set1.setLineWidth(2f);
            set1.setCircleRadius(3f);
            set1.setFillAlpha(65);
            set1.setDrawCircleHole(false);

            LineData data = new LineData(set1);
            data.setValueTextColor(Color.WHITE);
            data.setValueTextSize(9f);
            chartGSR.setData(data);
        }
    }

    private void chartGSR_method() {

        MyMarkerView mv = new MyMarkerView(this, R.layout.custom_marker_view,
"Vodivosť pokožky ");

        chartGSR = findViewById(R.id.chart2);
        chartGSR.setOnChartValueSelectedListener(this);

        mv.setChartView(chartGSR);
        chartGSR.setMarker(mv);

        chartGSR.setTouchEnabled(true);

        chartGSR.setDragDecelerationFrictionCoef(0.9f);

        chartGSR.setDragEnabled(true);
        chartGSR.setScaleEnabled(true);
        chartGSR.setDrawGridBackground(false);
        chartGSR.setHighlightPerDragEnabled(true);

        Description description = new Description();
        description.setText("Sample");
        description.setTextSize(12f);
        chartGSR.setDescription(description);

        chartGSR.setPinchZoom(true);

        chartGSR.setBackgroundColor(Color.WHITE);

        chartGSR.animateX(1500);

        Legend l = chartGSR.getLegend();

        l.setForm(Legend.LegendForm.LINE);
        l.setTextSize(11f);
        l.setTextColor(Color.BLACK);
        l.setVerticalAlignment(Legend.LegendVerticalAlignment.BOTTOM);
```

```
    l.setHorizontalAlignment(Legend.LegendHorizontalAlignment.LEFT);
    l.setOrientation(Legend.LegendOrientation.HORIZONTAL);
    l.setDrawInside(false);

    XAxis xAxis = chartGSR.getXAxis();
    xAxis.setTextSize(11f);
    xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
    xAxis.setTextColor(Color.BLACK);
    xAxis.setDrawGridLines(true);
    xAxis.setDrawAxisLine(true);

    YAxis leftAxis = chartGSR.getAxisLeft();
    leftAxis.setTextColor(Color.BLACK);
    leftAxis.setAxisMaximum(500);
    leftAxis.setAxisMinimum(0);
    leftAxis.setDrawGridLines(true);
    leftAxis.setGranularityEnabled(true);

    YAxis rightAxis = chartGSR.getAxisRight();
    rightAxis.setTextColor(Color.BLACK);
    rightAxis.setAxisMaximum(500);
    rightAxis.setAxisMinimum(0);
    rightAxis.setDrawGridLines(false);
    rightAxis.setDrawZeroLine(false);
    rightAxis.setGranularityEnabled(false);

    chartGSR.invalidate();
}


private void setDataECG() {

    ArrayList<Entry> values1 = new ArrayList<>();

    for (int i = 0; i < ecgTime.size(); i++) {
        values1.add(new Entry(i, Float.parseFloat(ecgValue.get(i))));
    }

    System.out.println(values1);
    System.out.println("-----");
    LineDataSet set1;

    chartECG.getAxisLeft().setGranularity(1f);
    chartECG.getAxisLeft().setGranularityEnabled(true);

    if (chartECG.getData() != null &&
            chartECG.getData().getDataSetCount() > 0) {
        set1 = (LineDataSet) chartECG.getData().getDataSetByIndex(0);
        set1.setValues(values1);
        chartECG.getData().notifyDataChanged();
```

```
            chartECG.notifyDataSetChanged();
        } else {
            set1 = new LineDataSet(values1, "ECG");

            set1.setAxisDependency(YAxis.AxisDependency.LEFT);
            set1.setColor(Color.rgb(225, 0, 0));
            set1.setCircleColor(Color.rgb(150, 0, 0));
            set1.setLineWidth(2f);
            set1.setCircleRadius(3f);
            set1.setFillAlpha(65);
            set1.setDrawCircleHole(false);

            LineData data = new LineData(set1);
            data.setValueTextColor(Color.WHITE);
            data.setValueTextSize(9f);

            chartECG.setData(data);
        }
    }


    @Override
    public void onValueSelected(Entry e, Highlight h) {
        Log.i("Entry selected", e.toString());

        chartECG.centerViewToAnimated(e.getX(), e.getY(),
chartECG.getData().getDataSetByIndex(h.getDataSetIndex())
                .getAxisDependency(), 500);
    }

    @Override
    public void onNothingSelected() {
        Log.i("Nothing selected", "Nothing selected.");
    }

    private void getTeplota() {
        databaseReference1.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                retriveTemp.setText("Teplota: " + snapshot.getValue().toString());
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Toast.makeText(MainActivity.this, error.getMessage(),
Toast.LENGTH_SHORT).show();
            }
        });
    }
```

```
    private void getGSR() {
        databaseReference4.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                gsrValue.clear();
                gsrTime.clear();
                for (DataSnapshot ds : dataSnapshot.getChildren()) {
                    gsrValue.add(ds.child("value").getValue().toString());
                    gsrTime.add(ds.child("time").getValue().toString());
                }
                setDataGSR();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                System.out.println(databaseError.getMessage());
            }
        });
    }

    private void getECG() {
        databaseReference3.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                ecgValue.clear();
                ecgTime.clear();
                for (DataSnapshot ds : dataSnapshot.getChildren()) {
                    ecgValue.add(ds.child("value").getValue().toString());
                    ecgTime.add(ds.child("time").getValue().toString());
                }
                setDataECG();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                System.out.println(databaseError.getMessage());
            }
        });
    }

    private void getHR() {
        databaseReference2.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                retriveSpo2.setText("SpO2: " +
dataSnapshot.child("spo2").getValue().toString());
                retrivePulse.setText("Pulz: " +
dataSnapshot.child("tep_srdca").getValue().toString());
            }
```

```
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            Toast.makeText(MainActivity.this, databaseError.getMessage(),
Toast.LENGTH_SHORT).show();
        }
      });
   }

}
```