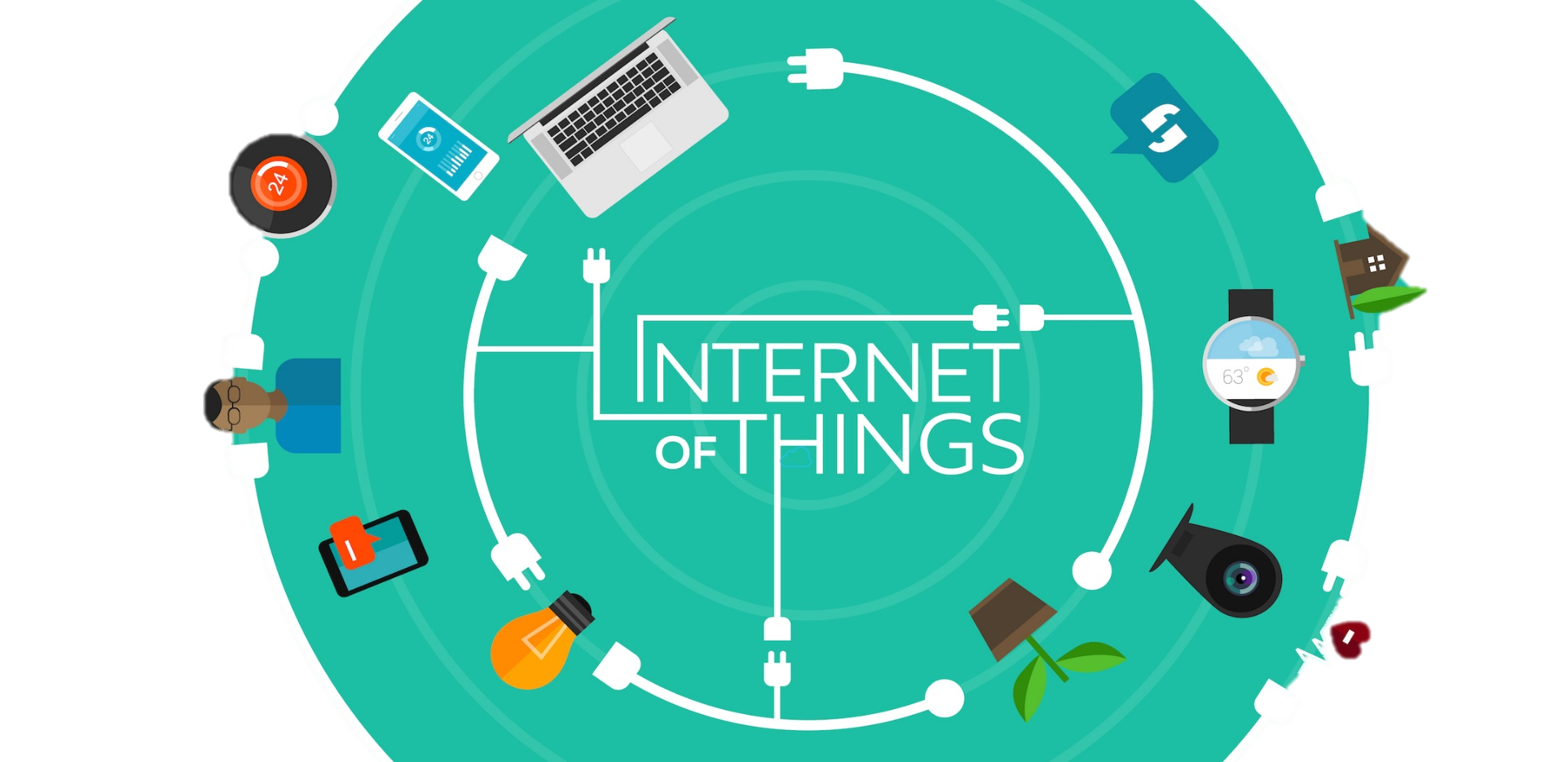
****

**Универзитет Св. Климент   
 Охридски -Битола**

**Факултет за информатички и  
комуникациски  
технологии**

ДИПЛОМСКА РАБОТА









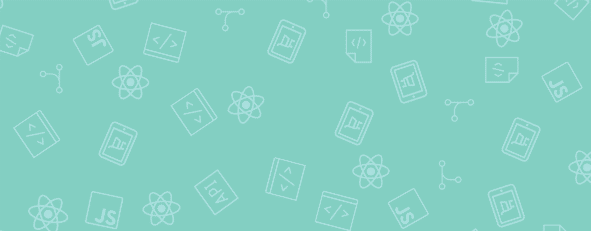


**Кандидат:** Мартина Талевска ИНКИ 53



**Ментор:** проф. Д-р Никола Рендевски

2020, Битола

**СОДРЖИНА**

[Вовед 2](#_Toc50985002)

[Интернет на нештата и Веб на нештата 3](#_Toc50985003)

[Хардверски компоненти 9](#_Toc50985004)

[Други Компоненти 10](#_Toc50985006)

[Пински каблиња (Jumper wires) 11](#_Toc50985008)

[Отпорник (Resistor) 12](#_Toc50985009)

[LED диоди 12](#_Toc50985010)

[Breadboard 13](#_Toc50985011)

[Node.js 14](#_Toc50985012)

[Node JS VS. Apache 18](#_Toc50985017)

[Node.js NPM 19](#_Toc50985018)

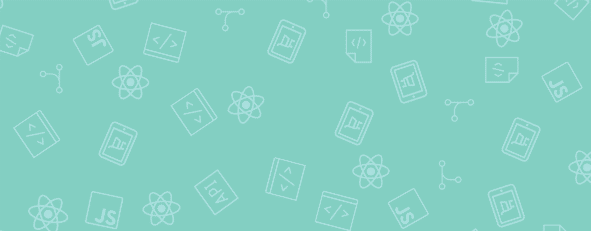
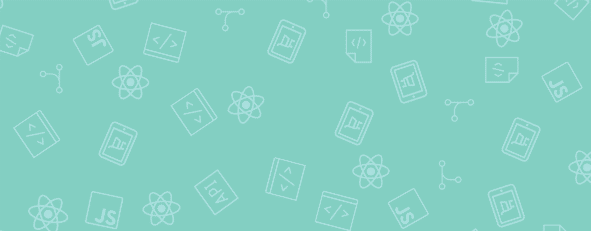
[Програмски дел 20](#_Toc50985019)

[Зошто WebSocket сервер? 20](#_Toc50985020)

[Чекор по чекор 30](#_Toc50985020)

[Заклучок 35](#_Toc50985027)

[Користена литература 37](#_Toc50985035)



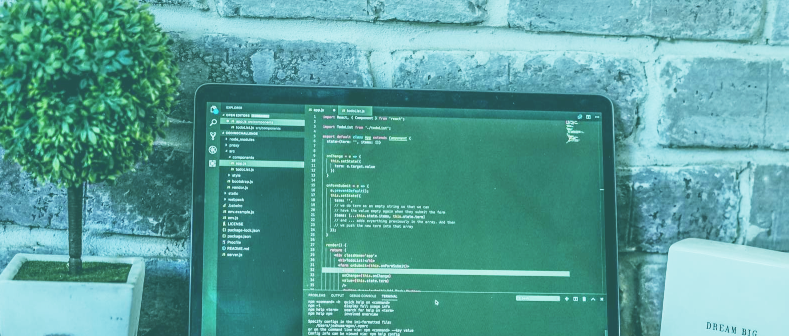
# Вовед

Идејата за изработување на дипломската работа е со цел да прикажаме како би изгледал светот со имплементација на Интернет на нештата во секојдневниот живот. Интернет на нештата (IoT) се појавува како следната голема револуција во светот на технологијата и се надеваме дека ќе ни овозможи разбирање и потребни информации за да напредуваме во свет во кој повеќето физички предмети имаат свои дигитални животи благодарение на веб-протоколите. Секојдневно се користат солидни протоколи за да се изградат уште поголеми, попаметни но поедноставни работи - за да се изгради веб на нештата! Спектарот на ИоТ вклучува низа апликации, стандарди, протоколи, архитектура и техники за анализа на податоци со кои уредите и предметите се поврзани на Интернет, за некоја деловна, индустриска или човечка цел. Комбинацијата на хардверски и софтверски компоненти за контролни апликации претставува основа за еволуција на cyber-физичките системи. Способностите на VLSI (Very Large Scale Integration) играат огромна улога во дизајнот на IoT системите. Специјализираните мрежни протоколи овозможуваат ефикасна комуникација во ова опкружување, вклучувајќи соодветни модели за комуникација машина за машина (M2M). Овие технологии се појавуваат со ограничувања за IoT опкружувањето кои се различни од типичната IT-околина, поради барањата за безбедност, реакции во реално време, работа со мала моќност итн. Безбедноста и приватноста бараат посебно внимание и посебни техники.

Интернетот на нештата (IoT) станува една од најпопуларните теми во технологијата и деловниот свет и следејќи го процесот, фокусот на IoT се префрлува од чист аспект на поврзување на уредите и собирање податоци кон премостување на јазот помеѓу физичката, дигиталната и човечката сфера преку мрежите и поврзаните процеси. Употребата на IoT се случува со различна брзина. Брзината е далеку повисока во производствената индустрија отколку во просторот за потрошувачки интернет на нешта (CIoT). Воглавно, IoT е суштински двигател за иновации за соочување со клиентот, дигитална трансформација, поврзаност на нештата, премостување на поделбата помеѓу физичките и дигиталните, податочна оптимизација и прилив на приходи во различни деловни сектори.

Во следните поглавија ќе ги разгледаме главните развојни чекори и предизвици при дизајнирање на решение базирано на Интернет на нештата.

**1**



## Интернет на нештата и Веб на нештата

Во последните години, Internet of Things го бележи својот развој во технологијата и бизнисот. Визијата на светот каде мини компјутерите со сензори и комуникациски интерфејс се вградени во инфраструктурата на нашите градови, автомобили, канцеларии и домови ја револуционираат секоја област во нашите животи- како играме, како работиме и како живееме. До пред некое време, ИоТ проектите беа фокусирани на градење проекти кои се мали во размер, затворени и изолирани, каде што уредите не беа дизајнирани за лесен пристап или пак за репрограмирање. Спојувањето на уредите со апликациите во даден случај значеше дека секоја промена на веќе постоечкиот проект ќе биде комплексна и скапа. Ова водеше кон поставување на лимит во одржувањето и еволуцијата на ИоТ, бидејќи ресурсите (време, финансиски средства и технички способности) се задолжителни секој пат кога е додадена нова функција.

Во контраст, веб на нештата станува широко успешен во последните две декади поради што е поедноставен да се научи и користи, исто така и лесно го нагласува спојување помеѓу серверите, пребарувачите и апликациите. Web of Things е специјалност на Internet of Things и она што го направи веб толку успешен е што се применува на вградени уреди со цел да го направи последниот развој на Internet of Things колку што е можно подостапен за повеќето девелопери. Со Web of the Things – исто како и со секој веб-текст едитор и со основно ниво на познавање на веб стандардите ( HTML и HTTP) може лесно да поврземе уреди и објекти на веб. Исто така овозможува напредок на следно ниво и води кон ефективно градење на интерактивна и иновативна real-world апликација што ги вклопува физичкиот и дигиталниот свет.

За да ја доловиме есенцијалната дефиниција на Интернет на нештата една реченица не е доволна. Концептот постои повеќе декади и нема дефинирани граници околу тоа што е Internet of things и тоа што не е. Сепак, основната дефиниција за визијата на Интернет на нештата е свет каде што интернетот е многу повеќе од колекција на мултимедиска содржина: туку се проширува во физичкиот, real-time свет користејќи огромен број на мали или микро компјутери. Или накратко дефинирано, Internet of things е систем на физички објекти што можат да бидат истражувани, мониторирани, контролирани или можат да стапат во интеракција со електронски уреди што комуницираат со различни мрежни интерфејси и евентуално можат да бидат конектирани на поголема интернет мрежа.

Пред две декади, каде секојдневните објекти можат да го почуствуваат светот преку сензори и после тоа да анализираат, зачувуваат или разменуваат информации постоеше само во научно-фантастичните новели или the Jetsons. Денес, тие сценарија стануваат реалност благодарение на колосалниот процес во вградените уреди што воведија во светот нова класа на објекти: паметни објекти. A smart thing (паметен објект) ќе го дефинираме секој физички објект што е дигитално надграден или зголемен со едно од наведените:

* Сензори (за температура, светло, движење)
* Активатори (дисплеј, звук, мотор)
* Компјутери (програми и логика)
* Комуникациски интерфејси (жични или безжични)

Паметните објекти го проширија светот во кој живееме со нов размер на апликации  
(сл.1 и сл.2). Со имплементирање на микро компјутери со ниски цени-но сепак моќни насекаде околу нас, станува полесно за интеракција со физичкиот свет со многу поприлагодливи просторни и временски резолуции од порано.

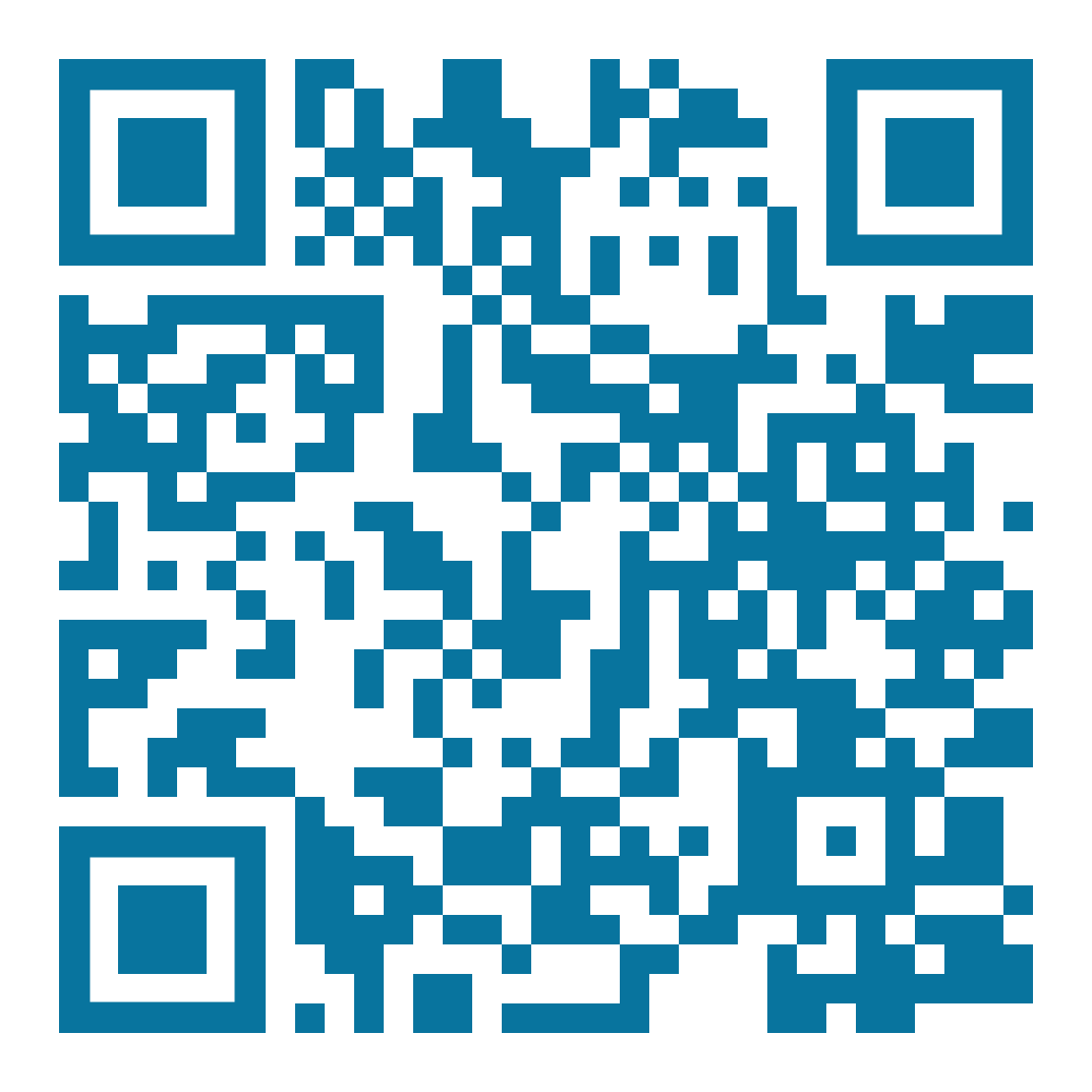
Околини

Машини

Уреди



Ознаки



Сл.1 Internet of things landscape

Компјутерска моќ и комплексност

Smart building

Smart car

Smart city

Arduino

Philips Hue

Raspberry Pi

iBeacon/BLE

QR ознака

NFC/RFID ознака

Со оглед на тоа што секојдневните објекти ќе бидат дигитално надградени, следниот логичен чекор е користење на World Wide Web екосистемот и инфраструктурата за да изградиме апликации за Internet of Things, воедно и ефективно ја прекршуваме шемата: ‘‘ one device, one protocol, one app‘‘ или ‘‘ еден уред, еден протокол, една апликација‘‘. Идејата за максимизирање на постојните и новите алатки и техники кои се користат на веб и применувајќи ги истите во развојот на Internet of things е ултималната цел на Web of things.



Сл.2 Internet of things landscape

Идеологијата на Интернет на нештата (IoT) може да се погледне како високо динамичен и радикално дистрибуиран мрежен систем составен од многу голем број на паметни предмети што можат да се идентификуваат. Овие објекти се во можност да комуницираат меѓу себе, со крајните корисници или други субјекти во мрежата. Влегувајќи во ерата на Интернет на нештата, употребата на мал, ефтин и флексибилен компјутерски хардвер што дозволува програмирање на крајниот корисник станува сегашност. Една од нив, разгледана во овој труд, е Raspberry Пи, целосно прилагодлива и програмабилна мала компјутерска табла. Компаративната анализа на нејзините клучни елементи и изведби со некои тековни постојни платформи на прототип IoT покажаа дека и покрај малкуте недостатоци, Raspberry Pi останува ефтин компјутер со многу успешно користење во разновиден спектар на апликации за истражување во IoT визијата.

IoT е чадор-термин што се користи за широк спектар на основни технологии и услуги, вклучително и:



Брзо растечкиот пазар на IoT технологија ги намалува трошоците за хардвер. Производите опремени со IoT сензори можат да ги предупредат клиентите кога е потребна услуга. Ова создава можности за бизнисите за постојани односи со клиенти и создавање на приливи на приходи за она што претходно беше еднократна продажба.

Распоредувањето на IoT-уредите ги обезбедува податоците и увидот, потребни за автоматизирање на процесите, насочување на работните текови, визуелизирање на моделите на употреба и поефикасно натпреварување во менување на деловната област.

## 1.1 Апликации

IoT системите можеме да ги користеме во широк спектар на апликации: Индустриските системи користат сензори за да ги следат самите индустриски процеси - квалитетот на производот - и состојбата на опремата. Паметните згради користат сензори за да ги идентификуваат локациите на луѓето, како и состојбата на зградата. Тие податоци можат да се користат за контрола на системите за греење/вентилација/ климатизација и системи за осветлување со цел да се намалат трошоците. Паметните градови користат сензори за следење на сообраќајот, пешаците и возилата и можат да интегрираат податоци од паметни згради. Возилата користат мрежни сензори за да ја следат состојбата на возилото и да обезбедат подобрена динамика, намалена потрошувачка на гориво и помали емисии. Медицинските системи поврзуваат широк спектар на сензори за следење на пациентите кои може да се наоѓаат дома, во возила за итни случаи, лекарска канцеларија или болница. Податоците од сензорите се собираат и се анализираат со дополнително подесување дали извештаите да се генерираат периодично - на час, на ден, итн. - или може постојано да се ажурираат. Анализата на податоците од сензорот може да предизвика активирање на активаторите. Податоци што ги добиваме ги користиме за надополнување на алгоритмите за контролирање што генерираат излези за активаторите.

## 1.2 Архитектура

Клучен аспект на IoT е водење на настани или апериодично земање примероци. Традиционалната обработка и контрола на дигиталниот сигнал претпоставува периодични примероци што резултираат во податоци од временска серија. Сепак, временските серии трошат премногу енергија на јазлите и премногу ширина на опсегот на мрежата. Не сите апликации се подложни на апериодично собирање податоци. Ограничувањата на напојувањето и ширината на опсегот, исто така, поттикнуваат дистрибуирани компјутери преку сензорски настани. Релативно малите процесори можат да извршат корисна обработка на многу текови на податоци. Препознавањето на интересни настани со користење на обработка на работ ја намалува количината на потрошен опсег на мрежата; исто така ја намалува потрошувачката на енергија бидејќи безжичната комуникација бара големи количини на енергија. Cloud-computing - (централизирани сервери) или fog-computing (сервери поблиску до работ) може да се користат за извршување на понатамошна обработка на извлечените настани.

## 1.3 Безжични мрежи

Безжичните мрежи се составен дел на IoT системите. Безжичните мрежни врски ја олеснуваат инсталацијата и работењето. Радио комуникацијата бара повеќе енергија отколку жичаната комуникација. Некои од безжичните мрежи што се користат во денешните IoT уреди се дизајнирани за други цели, како што се телефонија и мултимедија. Како резултат, тие не се оптимизирани за комуникација водена од настани и трошат значителни количини на енергија во протоколот за комуникација. // Една од иронијата на IoT е што многу раб уреди и нивните безжични мрежи не работат на Интернет протоколот ( IP). IP воведува значителни трошоци со дополнително ниво на пакетизација и поврзана обработка. Многу IoT уреди избегнуваат IP и се потпираат на возводни јазли за да им обезбедат присуство на Интернет. Мрежите на IoT обично ги водат експерти за не-компјутер. Безжичните мрежи на IoT мора да бидат лесни за распоредување и релативно самостојно управување.

## 1.4 Безбедност и приватност

Безбедноста конечно е препознаена како основно барање за сите видови компјутерски системи, вклучително и IoT системи. Сепак, многу IoT системи се помалку безбедни од типичните Windows / Mac / Linux системи. Безбедносните проблеми на IoT потекнуваат од низа причини: несоодветни безбедносни карактеристики во хардверот, слабо дизајниран софтвер со низа ранливости, стандардни лозинки и други безбедносни грешки во дизајнот. Несигурните IoT јазли создаваат проблеми за безбедноста на целиот IoT систем. The Dyn нападот [Sch16] е еден пример за напад врз основа на IoT врз традиционалната Интернет инфраструктура. Приватноста е поврзана со безбедноста, но бара специфични мерки на ниво на апликација, мрежа и уред. Не само што корисничките податоци мора да бидат заштитени од целосна кражба, туку мрежата треба да биде дизајнирана така што помалку приватните податоци не можат лесно да се користат за да се заклучат повеќе приватни податоци.

## 1.5 Системи управувани од настани

Земајќи во предвид дека настанот е основен тип на податоци во IoT системите и дека системите управувани од настани се важна техника за структурирање на IoT, голем дел од технолошките блокови што се користат за креирање на IoT денес покажуваат одредена моќност за разлика од традиционалните системи ориентирани кон трансакции. Обработката на настани нè тера да го третираме времето како концепт од прва класа и да ја разгледаме врската помеѓу настаните во секвенците на настаните. Симулацијата на системот воден од настани е широко користена за моделирање широк спектар на инженерски системи. Во тој контекст, настан генерално се користи за да се означи промената на состојбата на променливата. Со оглед на децентрализираната природа на IoT системите, ние сме подготвени да го разгледаме- повторување на вредноста на настанот - како дел од моделот на настанот. Сите овие употреби на поимот настан можат да се обединат за да се создадат богати системски структури.



**2**

# Хардверски компоненти

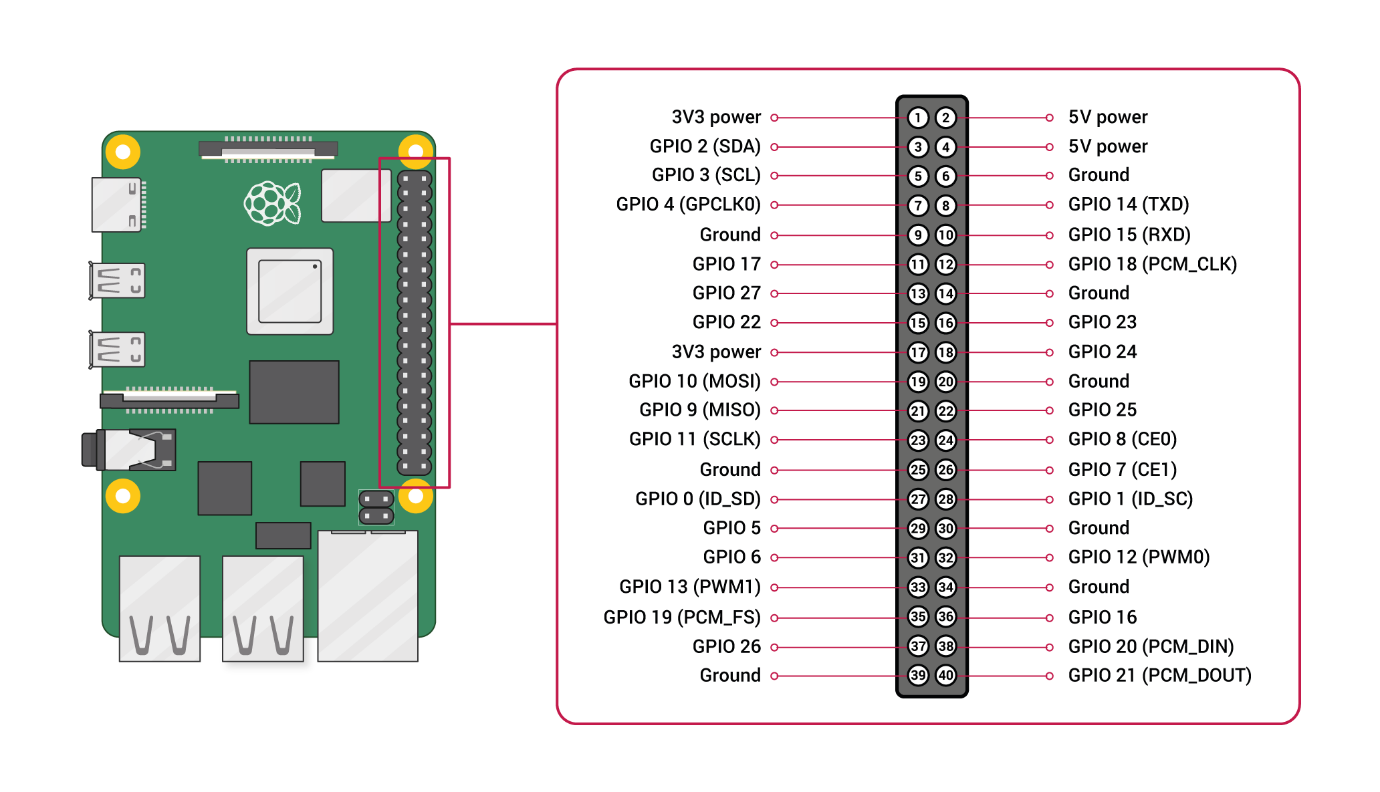
Интернетот на нештата, или IoT, се однесува на милијарди физички уреди ширум светот кои се поврзани на интернет мрежа, и меѓусебно собираат и споделуваат податоци. Благодарејќи на евтините процесори и безжичниот интернет, сега е возможно од авто-пилот на авион до self-driving автомобил како дел од IoT. Со помош на Raspberry Pi and Node Js сега можеме да креираме ИоТ процес. Во продолжение ги опишуваме деловите кои ни се потребни и нивно синхронизирање во процесот на изработката на овај IoT процес.

## **Raspberry Pi**

**Raspberry Pi** е ARM architecture процесор базиран на боард дизајниран за електро-инженери, и исто така е една од најдоверливите платформи за развивање на проекти. Со поголема брзина на процесорот и од само 1 GB RAM, the PI може да се користи за проекти од поголеми размери.

Raspberry Pi е името на серијалот на single-board компјутери направени од Фондацијата Raspberry Pi, добротворна организација во Велика Британија, чија цел е да ги едуцира луѓето за компјутерите и да создаде полесен пристап за учење. Во големина е колку една кредитна картичка и се приклучува на компјутерскиот монитор или ТВ. Користат 700 MHz процесор ARM, а моделите се наоѓаат опремени со 256 MB или 512 MB RAM меморија. Разликуваме два модела на уредот: Model A и Model B.

Единствените вистински разлики се додавањето на Ethernet и дополнителната USB порта на поскапиот Model B. Тие можат да поддржат најмногу пет USB 2.0 порти. Raspberry Pi може да користи MicroSD картичка или флеш меморија (до 4 GB) за нестабилно складирање и користи HDMI за дигитален аудио и видео излез. Нема мрежен интерфејс, но тие поддржуваат мрежно поврзување преку Wi-Fi и Ethernet преку USB. Raspberry Pis обично го работат Linux како нивен оперативен систем. Arch Piux, Fedora, Puppy Linux, Slackware, FreeBSD, NetBSD, OpenSUSE и RISC OS се достапни за Pi. Исто така е достапен и Raspbmc, оперативен систем специјално создаден за користење на Pi како дигитален центар за –медиуми или HTPC.



Сл.3 Raspberry Pi GPIO (general-purpose input/output) pins

## Други Компоненти

Други компоненти кои ни се потребни за комплетирање на проектот се:

* ****Пински каблиња (jumper wires)
* ****3x Отпорници од 3.3Ω
* ****ЛЕД диоди
* ****Breadboard

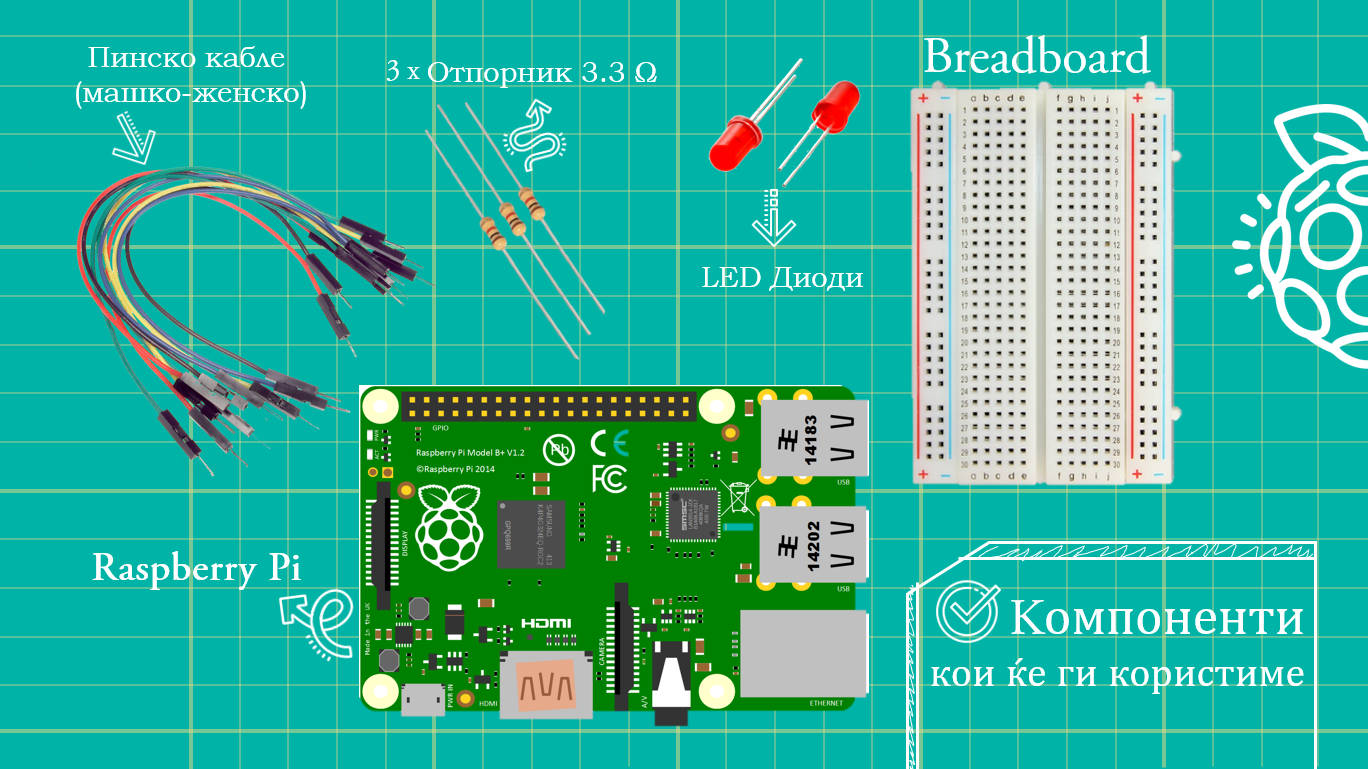
### 

### Пински каблиња (Jumper wires)

Поимот „jumper wire“ едноставно се однесува на спроводна жица што воспоставува електрична врска помеѓу две точки во коло. Пинските каблиња или jumper wires можете да ги користите за да модифицирате коло или да дијагностицирате проблеми во колото.

Пинското кабле е со конектор или игла на секој крај (или понекогаш без нив - едноставно „затегната“), што вообичаено се користи да ги поврзете компонентите на breadboard-от или друг прототип или тест коло, внатрешно или со друга опрема или компоненти, без лемење. Индивидуалните пински каблиња се вградени со вметнување на нивните „крајни конектори“ во слотите што се предвидени на breadboard-от, во header на конекторот на поклопецот или на парче опрема за тестирање.

Пинските каблиња обично доаѓаат во три верзии: машко-машко, машко- женско и женско- женско кабле. Разликата помеѓу секоја е во крајната точка на жицата. Машките краеви имаат испакната игла и можат да се вклучат во нештата, додека женските краеви немаат испакнати игли и се користат за приклучување во работите кои ги користиме со нив.



Сл.4 Компоненти кои ќе ги користиме****

### Отпорник (Resistor)

Отпорникот е пасивна дво-терминална електрична компонента која спроведува електричен отпор како елемент на колото. Во електронските кола, отпорниците се користат за намалување на протокот на струја, прилагодување на нивото на сигналот, за поделување на напони, активни елементи на пристрасност, меѓу другите намени и прекинување на преносни линии. Главната функција на отпорниците во колото е да го контролираат протокот на струја во други компоненти. Доколку земеме една LED диода на пример: ако премногу струја тече низ ЛЕД диодата, таа е уништена. Значи, ги користиме отпорниците за да го ограничуваат протокот на електроните, намалувајќи ја струјата.Напонот се појавува со потенцијалната разлика во енергијата во резисторот. Математичкиот одговор е дека отпорник е двостепен електричен уред за спроведување, законот на Ом: V = IR. Отпорниците влијаат и на струјата и на напонот. Тие го прават тоа во линеарен начин... Напонот низ секој отпор ќе се разликува директно во однос на струјата што тече низ неа. Значи, во првиот отпорник од 10 оми, напонот низ него е 10 х 10.

За да го намалиме напонот на половина, едноставно формираме делител на напон помеѓу 2 отпорници со еднаква вредност (на пример, 2 10KΩ) отпорници. За да го поделиме напонот на половина, сè што треба да сториме е да поставиме какви било 2 отпорници со еднаква вредност во серија и потоа ставаме пинско кабле помеѓу резисторите.Иако отпорниците доаѓаат во различни форми, можеме да ги поделиме на само два основни типов:

„Фиксирани“ отпорници

Варијабилни отпорници (или „потенциометри“)

****

### LED диоди

Светлечка диода, односно ЛЕД (*Light-emitting Diode*) е електронски полуспроводничка компонента која испушта светлина кога низ неа ќе помине електрична струја. Светлината се произведува кога честичките што ја носат струјата (познати како електрони) се комбинираат заедно во полупроводничкиот материјал. Бидејќи светлината се создава во рамките на цврстиот полупроводнички материјал, LED диоди се опишуваат како уреди во цврста состојба. Терминот solid-state lighting, кој исто така опфаќа органски LED диоди (OLED), ја разликува оваа технологија на осветлување од други извори кои користат загреани нишки (лампи за блескаво и волфрам) или гас празнење (флуоресцентни ламби). ЛЕД се користат како индикатори во технички апликации, но сè повеќе и како извор на светло. Користени се како практичен електронски елемент уште од 1962 година, кога првите LED светеа со црвено слабо светло, додека новите модерни верзии се достапни со видливи, ултравиолетови и инфрацрвени бранови должини како и силна светлина.

Главните полупроводнички материјали што се користат за производство на LED диоди се:  
  
     Индиум галиум нитрид (InGaN): сини, зелени и ултравиолетови LED диоди со висока осветленост

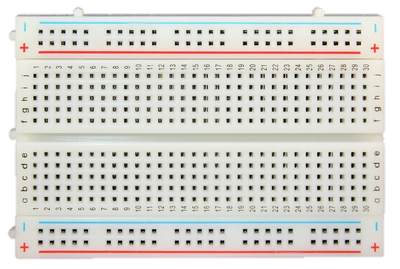
Фосфид на алуминиум галиум индиум (AlGaInP): жолти, портокалови и црвени LED диоди со висока осветленост

 Алуминиум галиум арсенид (AlGaAs): црвени и инфрацрвени LED диоди

Галиум фосфид (GaP): жолти и зелени LED диоди

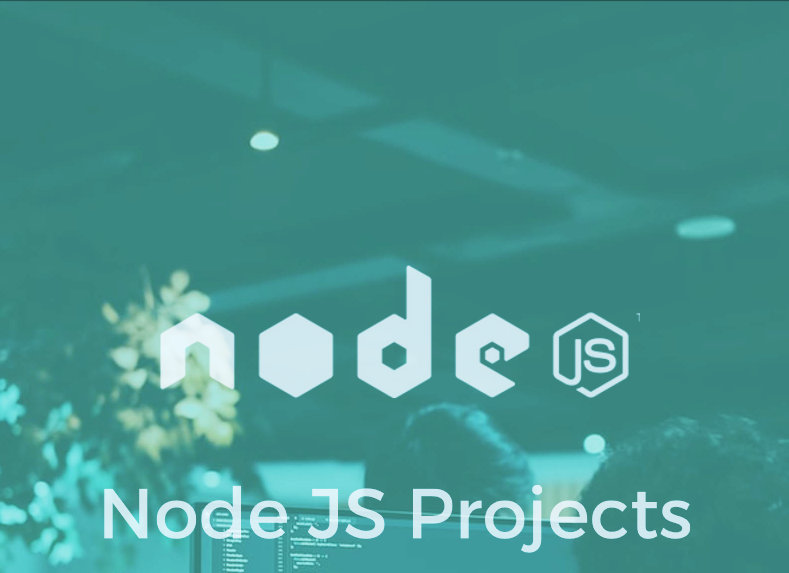
****

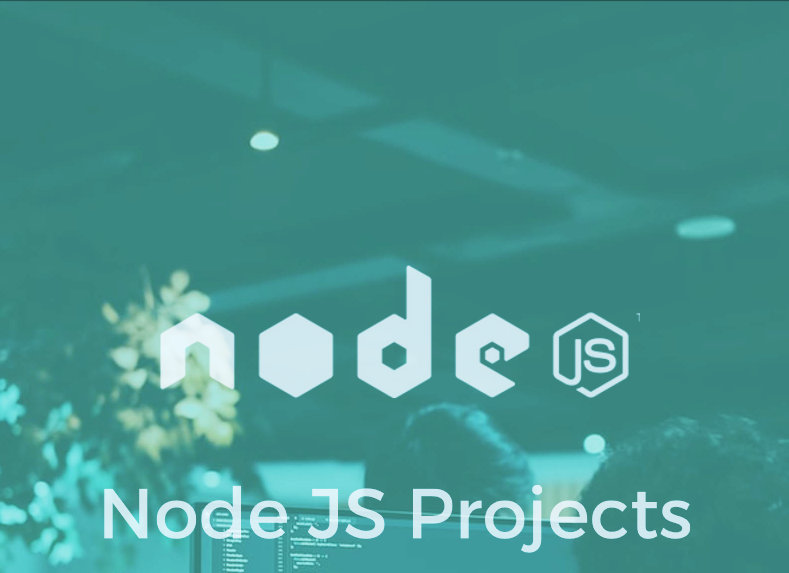
### Breadboard

Breadboard е градежна основа за прототипирање на електроника. Првично, зборот се однесувал буквално на табла за леб, полирано парче дрво кое се користи за сечење леб. Бидејќи solderless breadboard не бара лемење, таа може повторно да се употреби. Ja користимe за да изградеме и тестираме кола брзо пред да финализираме кој било дизајн на колото. Breadboard-от има многу дупки во кои можат да се вметнат компоненти на колото како IC и отпорниците.

Повеќето електронски компоненти во електронските кола можат да бидат меѓусебно поврзани со вметнување на нивните leads или терминали во дупките и потоа да поврзат преку жици каде што е соодветно. Breadboard-от има ленти од метал под таблата и ги поврзува дупките на горниот дел од таблата. Металните ленти се поставени како што е прикажано подолу. Забележете дека горниот и долниот ред на дупките се поврзани хоризонтално и се делат на средина додека преостанатите дупки се поврзани вертикално.

Сл.5 Breadboard





## Node.js

Зголемената популарност на JavaScript донесе многу промени, па затоа денес лицето на веб-развој драматично се разликува од претходно. Работите што можеме да ги направиме со JavaScript на серверот, како и во прелистувачот, беше тешко да се замислат пред неколку години или беа капсулирани во средини како што се Flash или Java Applets.

Пред да започнеме да навлегуваме во решенијата со Node.js, ќе разгледаме неколку дефиниции за тоа што е Node.js. Како што наведува Википедија: „Node.js е спакувана компилација од Google’s V8 JavaScript engine, слојот за апстракција на платформата libuv и основната библиотека, што е главно напишано и во JavaScript“. Покрај тоа, вреди да се напомене дека Рајан Дал, творецот на Node.js, имаше за цел да создаде веб-страници во реално време со можност за притисок, „инспирирани од апликации како Gmail“.

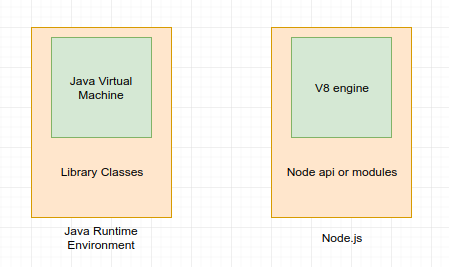
По повеќе од 20 години со stateless-web заснован на stateless барање-одговор парадигмата, конечно, имаме веб-апликации со двонасочна конекција во реално време.

Со една реченица: Node.js се истакнува во веб-апликации во реално време, користејќи push технологија преку web sockets. Сигурно и ќе се запрашате што е толку револуционерно во тоа? Сега во веб-апликациите со двонасочна врска во реално време, и двата, клиентот и серверот можат да иницираат комуникација, овозможувајќи им слободно да разменуваат податоци . Ова е целосен контраст со типичната парадигма каде клиентот секогаш ја започнува комуникација. Дополнително, сето тоа се заснова на отворен веб-stack (HTML, CSS и JS) преку стандардната порта 80.

Некој сепак може да тврди дека ова постоеше со години во форма на Flash и Java Applets - но во реалноста, тие беа само sandboxed околини кои ја користеа мрежата како протокол за транспорт што треба да му се достави на клиентот. Додадено на тоа, беа управувани во изолација и честопати оперираа со нестандардни портови, што можно беше да бараат дополнителни дозволи и слично.

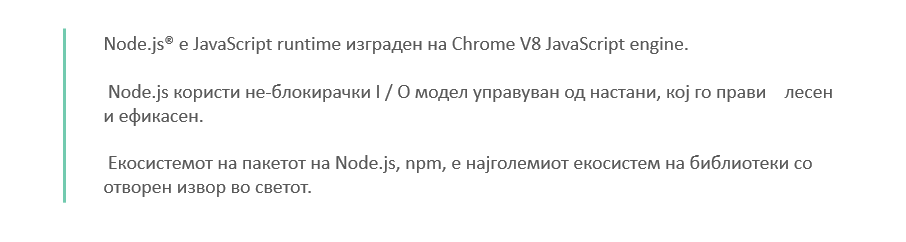
Со сите свои предности, Node.js сега игра клучна улога во технолошкиот куп кај многу компании со висок профил кои зависат од неговите уникатни придобивки. Фондацијата Node.js ги консолидираше сите најдобри размислувања околу тоа зошто претпријатијата треба да размислат за Node.js во кратка презентација што може да се најде на страницата Студии на случај на Фондацијата Node.js.

Доколку знаеме дека JavaScript е широко познат јазик креиран да се погрижи за интеракцијата на корисникот во веб пребарувачот, сега благодарение на node.js можеме да користиме JavaScript надвор од пребарувачот, и исто така во интеракција со хардвер уреди како што е Raspberry Pi. Node.js run-time околината вклучува сè што ни треба за да извршиме програма напишана во JavaScript. Node.js настана кога развивачите на JavaScript го проширија од нешто што можевме да го извршуваме само во прелистувачот во нешто што може да го извршиме на нашата машина како самостојна апликација. JavaScript сега има можност да прави работи што можат да ги прават другите јазици за скрипти како Python. И прелистувачот JavaScript и Node.js работат на V8 JavaScript runtime engine, со тоа што го зема нашиот JavaScript код и го претвора во побрз машински код. Машинскиот код е код на ниско ниво што компјутерот може да го извршува без претходно да го интерпретира.



Сл.6 Аналогија на Node JS со Java Runtime Environment

**Зошто Node.js?**

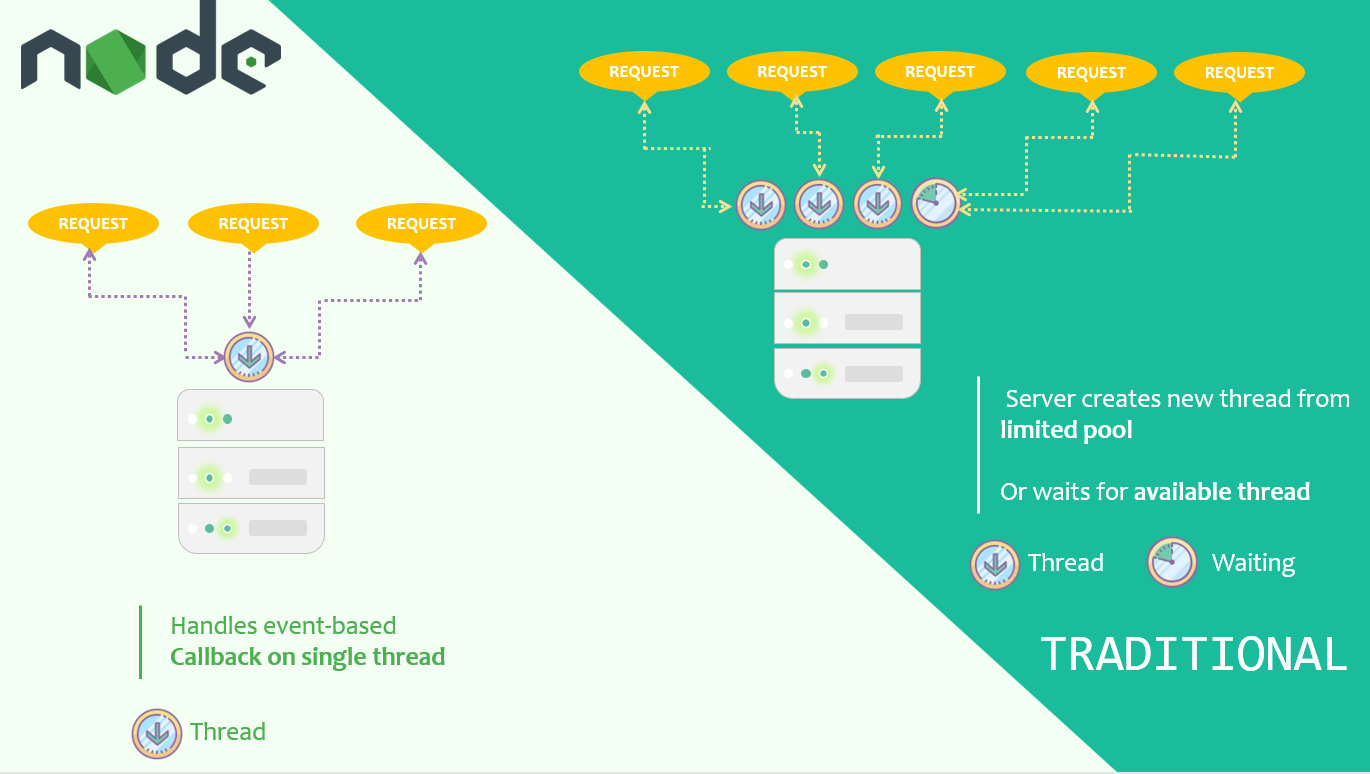
Во продолжение ја наведуваме дефиницијата како што е дадена на официјалната веб-страница на Node.js:

Input/Output или I/O се однесува на влез / излез, во спектар почнувајќи од читање / пишување локални датотеки до испраќање HTTP барање до API. I/O одзема време и може да блокира други функции. Како пример ќе земеме сценарио каде бараме backend база на податоци за деталите на user1 и user2, а потоа ги отпечатиме на екранот/ конзолата. Одговорот на ова барање одзема време, но и двете барања за кориснички податоци може да се извршат самостојно и истовремено.

**Како работи?**

Идејата позади Node.js е да користи не-блокирачки, настан-базиран (event-driven) I/O со цел да биди лесен и ефикасен пред real-time апликациите кои работат на дистрибуирани уреди. Покрај тоа, Node.js не е нова платформа што ќе доминира во светот за развој на веб-страници, наместо, тоа е платформа што исполнува одредена потреба. Доколку го разбираме ова, нема да сакаме да го користиме Node.js за CPU- интензивни операции (кои го преоптоваруваат процесорот), всушност користејќи го за тешки пресметки ќе ги поништиме сите негови предности. Креирањето брзи, скалабилни мрежни апликации е местото каде Node.js го зазема своето место, бидејќи е способен за справување со голем број на истовремени врски со голема пропустливост (throughput) што е еднакво на висока приспособливост.

Споредено со традиционалните web-serving техники каде што секоја конекција (request) поткрева нова нишка (thread), зафаќајќи повеќе RAM, а некогаш и целосната меморија достапна, Node.js оперира со една нишка (single-thread) користејќи non-blocking I/O повици, со што дозволува и поддржува десетици илјади истовремени врски што се случуваат во циклусот на настани.

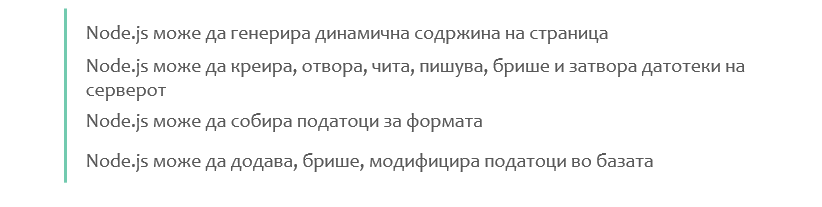


Сл.7 Node.js VS Traditional Web server

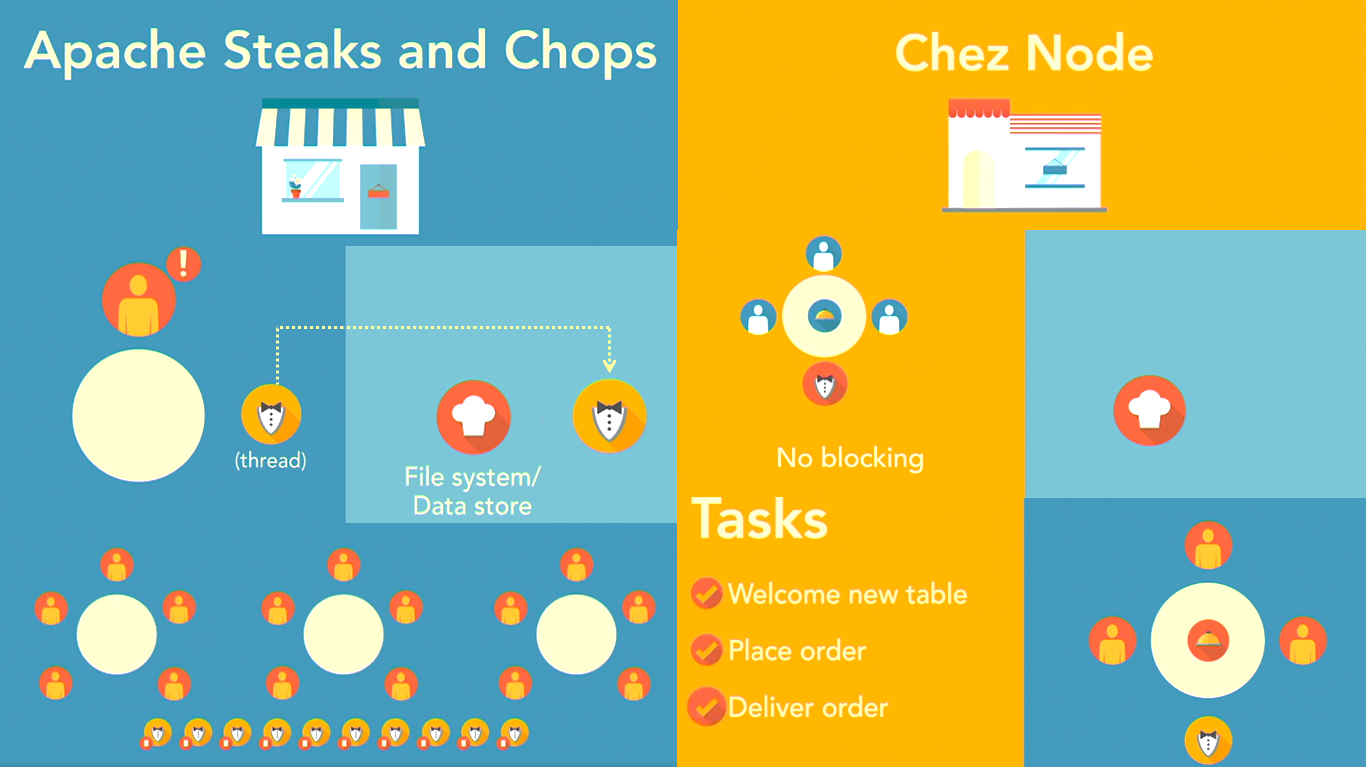
Пример: ако претпоставиме дека секоја нишка потенцијално има придружна меморија од 2МВ со неа, а работи на систем со 8GB RAM тогаш имаме теоретски максимум од 4.000 истовремени врски (пресметките се земени од статијата на Мајкл Абернети „What is Node.js? ”, објавено во IBM developerWorks во 2011 година), плус трошоците за context-switching помеѓу нишките. Ова е типично сценарио со кое би се справувале доколку користиме традиционалната техника на веб-сервирање. Со цел да го избегнеме ова, со Node.js постигнуваме нивоа на приспособливост од преку 1М истовремени врски и над 600к истовремени web sockets конекции.

Секако, тука е и прашањето за споделување на една нишка меѓу сите барања на клиентите, и тоа е потенцијалната стапица во пишувањето на апликациите во Node.js. Како пример, тешките пресметки можат да ја задушат единствената нишка и да предизвикаат проблеми за сите клиенти со што последователните барања (requests) ќе бидат блокирани додека наведената пресметка не се заврши.

**Што може да направи Node.js?**



## Node JS VS. Apache

Со едноставен и сликовит приказ на следната страна ќе демонстрираме како Node JS функционира и зошто е толку брз начин на работење со споредба на два ресторани. Првиот е Аpache. Во овај ресторан, секој нов гостин претставува нов корисник. И правењето нарачка претставува ново барање или request. Со самата нарачка менаџерот ни доделува келнер кој ќе не служи само нас и ќе ги прима само нашите нарачки- келнерот претставува нашата нишка(thread) до кујната која во нашата аналогија ја толкуваме како File System/Data Store. Вака работи Apache, секое барање е single- threaded. Откако ќе ја земе нарачката келнерот ја доставува во кујната и чека додека да биде готова. Во меѓувреме неможе да прими друга нарачка, што значи шефот го блокира корисникот за да направи друга нарачка за чаша вода. Во Apache, нишката -the single thread чека додека File System да заврши со читање на фајловите за да продолжи да прави било што друго. Ова може да се протолкува како блокирање. Доколку има повеќе гости или во нашиот случај повеќе корисници, секој корисник има свој келнер или нишка, што е добра услуга. Но повеќе гости, значи повеќе келнери. Од друга страна пак, во Chez Node има само еден келнер, бидејќи Node JS e single threaded. Тоа значи келнерот ја зема нарачката од еден гостин и ја носи во кујната, и во меѓувреме оди кај другиот гостин за нова нарачка. Врз основа на ова можеме да кажеме дека тука келнерот се однесува асинхроно, односно секоја активност што треба да ја направи келнерот претставува нов настан или event. Настаните ќе бидат преземени во редослед како што се повикани. Келнерот тука не чека, нема блокирање или така наречено- non-blocking event driven IO. Сите корисници ја користат истата нишка (single thread). Node JS е асинхрон, што значи може да прави повеќе од една работа. Оваа способност за мултитаск е она што го прави Node JS толку брз.

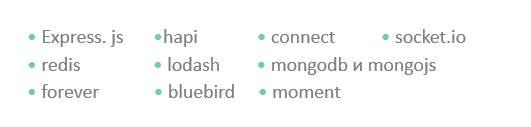
Сл.8 Node.js VS Apache

## Node.js NPM

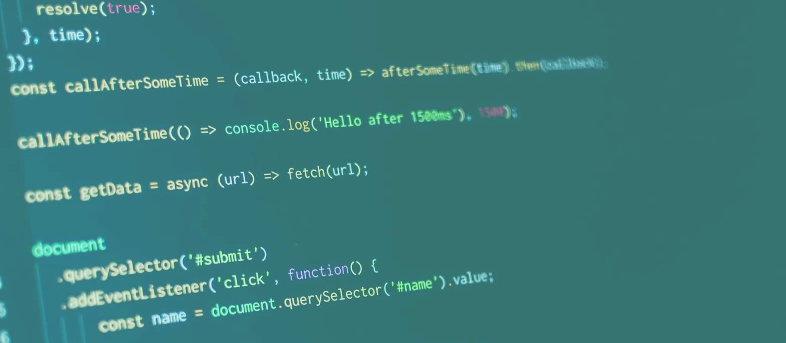
Кога дискутираме за Node.js, едно нешто што дефинитивно не треба да го изоставиме е вградената поддршка за управување со пакети, Node Package Manager (NPM), алатка што стандардно доаѓа со секоја инсталација на Node.js. Идејата за NPM модулите е прилично слична на идејата за Ruby Gems: збир на јавно достапни компоненти за повеќекратна употреба, достапни со лесна инсталација преку мрежно складиште, и менаџмент на верзии и зависности.

Целосен список на спакувани модули може да се најде на веб-страницата на npm или да се пристапи со помош на npm CLI алатката што автоматски се инсталира со Node.js. Екосистемот на модулот е отворен за сите и секој може да објави свој модул што ќе биде наведен во складиштето за npm. Баш поради тоа, и бидејќи нема процес на проверка, треба да бидеме претпазливи за оние кои се со низок квалитет, несигурни или малициозни. npm се потпира на извештаите на корисниците за тоа кои библиотеки ги прекршуваат правила и е потребно да се симнат, но за полесна одлука која библиотека да ја користиме, вклучуваат статистички податоци како број на преземања и број на зависни пакети.

Некои од најкорисните npm модули се:



**3**



# Програмски дел

# Зошто WebSocket сервер?

Доколку ја разбираме HTTP комуникацијата знаеме дека кога прелистувачот испраќа HTTP барање, прелистувачот одговара на ова барање со некои податоци. Откако податоците ќе бидат примени од прелистувачот, врската се прекинува.  
  
Оттука, за секое барање, ние создаваме нова HTTP врска за која е потребно извесно време да се воспостави и исто така содржи некои мета-податоци како заглавија(headers) и носивост(payload). Испраќањето на HTTP барање знае да биде тежокк процес и да одземи многу време.

Серверот не може да испрати податоци на клиентот, доколку клиентот не го побара тоа. Оттука, согледуваме дека HTTP е stateless I unidirectional- еднонасочен протокол за комуникација и не треба да се користи за двонасочна комуникација во реално време.

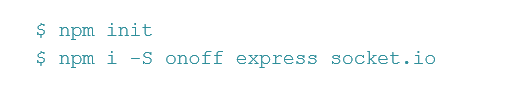
Спротивно на тоа, WebSocket конекцијата е двонасочен протокол за комуникација. Откако ќе се оствари web-socket конекцијата, таа останува отворена сè додека клиентот или серверот намерно не ја затвори. Откако ќе се воспостави врската, клиентот и серверот можат да си испраќаат настани еден на друг.

WebSocket конекцијата е надградба на HTTP конекција. Значи, клиентот прво испраќа HTTP барање до HTTP серверот со connection: Upgrade и Upgrade: WebSocket request headers.  
Тогаш е одговорност на HTTP серверот да воспостави WebSocket врска на истата TCP врска. Откако ќе се воспостави врската, серверот не може да испраќа настани до клиентот или да слуша настани од клиентот. Не сите прелистувачи имаат поддршка за протокол за WebSocket комуникација. Сепак, socket.io и socket.io-клиентот заедно обезбедува безбеден механизам за да се постигне целосно дуплекс механизам користејќи долго анкетирање кога протоколот WebSocket не е поддржан од прелистувачот.  
  
Освен испраќање едноставни настани, socket.io имаат богат API за управување со имиња на простори и простории за врски. Треба да ја разгледате нивната документација за да дознаете повеќе.

WebSocket е комуникациски протокол кој обезбедува full-duplex комуникациски канал преку единствена TCP врска. WebSocket се разликува од HTTP. Двата протоколи се наоѓаат на седмиот слој во моделот OSI и зависат од TCP кој се наоѓа на четвртиот слој. Иако се различни, RFC 6455 наведува дека WebSocket "е дизајниран да работи преку HTTP порти 80 и 443 како и да поддржува HTTP прокси и посредници, " што го прави компатибилен со HTTP протоколот. За да се постигне компатибилност, WebSocket handshake го користи HTTP Upgrade header со цел да се направи промената од протоколот HTTP во WebSocket протокол.

WebSocket ни овозможува интеракција помеѓу веб-прелистувач (или друга клиент апликација ) и веб-сервер со пониски опции од полу-дуплекс, како што се HTTP polling, при тоа го олеснува преносот на податоци од и до серверот во реално време. Ова е овозможено со обезбедување на стандардизиран начин серверот да испраќа содржина до клиентот без претходно да биде побарано од клиентот и дозволува пораките да се пренесуваат напред и назад, додека сеуште ја одржуваме отворена врската. На овој начин, се одржува двонасочен разговор помеѓу клиентот и серверот. Комуникациите обично се вршат преку TCP порта 80 (или 443 во случај на шифрирана врска со TLS), што е од корист за околините кои што ги блокираат non -веб Интернет конекциите со користење на заштитен ѕид-firewall. Слични двонасочни комуникации на прелистувач-сервер се постигнати на нестандардизирани начини со користење на технологии како што е Комета.

Кога ќе ја отвориме веб апликацијата во веб прелистувачот, таа се поврзува со WebSocket серверот, хостиран на Raspberry Pi. Кога корисникот ќе кликне на копчето (на веб-апликацијата), оваа веб-апликација испраќа порака (настан) до WebSocket серверот и серверот може да манипулира со состојбата на GPIO пиновите врз основа на информациите содржани во овој настан.



Од горенаведените команди, ја иницијализиравме датотеката package.json за да ги следиме project dependencies. Пакетот onoff обезбедува високо ниво на API за комуникација со GPIO на Raspberry Pi.

Пакетот express обезбедува АПИ на високо ниво за да се создаде HTTP-сервер во Node.js за да им служи на нашата веб-апликација која вклучува ги датотеките .html, .css и .js. Исто така, ќе се користи за поставување на WebSocket серверот.

Пакетот socket.io обезбедува API за создавање на WebSocket-сервер во Node.js. Овој пакет се користи од страната на серверот (Node.js). Сепак, socket.io-клиент обезбедува JavaScript датотека за веб-апликацијата (во прелистувачот) за комуникација со WebSocket серверот.

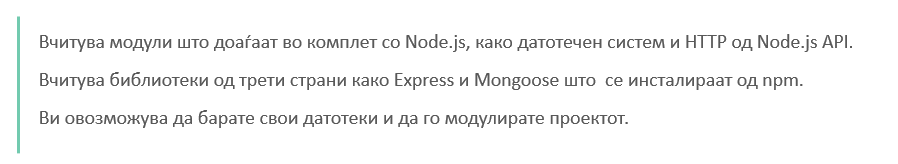
*****led-control.js* датотеката содржи GPIO логика за контрола на LED светлата. Оваа датотека извезува API функции за серверот.

**** *server.js* датотеката содржи express -сервер за да биде домаќин на нашата веб-апликација и WebSocket сервер кој се поврзува со оваа веб-апликација за да слушате прилагодени настани испратени од веб-апликацијата, како што се притискање на копчињата.

****веб-апликација / директорка содржи средства за нашата веб-апликација. Кога корисникот отвора специфична URL-адреса во прелистувачот, HTML. датотеката HTML ќе биде опремена по дифолт. Увезува лист за стилови.css во копчиња за стилови.

**** *main.js* содржи логика да се поврзете со серверот WebSocket и да ги слушате настаните за кликнување на копчето DOM. Се увезува според индексот.html

**Require**

Require е функција и прави три работи:

led-control.js

// import `onoff` package

const { Gpio } = require( 'onoff' );

// configure LED pins

const pin\_red = new Gpio( 13, 'out' );

const pin\_yellow = new Gpio( 19, 'out' );

const pin\_green = new Gpio( 26, 'out' );

const pin\_yellow\_2 = new Gpio (17, ‘out’);

const pushButton = new Gpio (14, ‘out’);

// toggle LED states

exports.toggle = ( r, y, g, l) => {

pin\_red.writeSync( r ? 1 : 0 );

pin\_yellow.writeSync( y ? 1 : 0 );

pin\_green.writeSync( g ? 1 :0 );

};

Како што спомнавме и претходно, пакетот onoff го користи BCM системот за нумерирање на пиновите. Според дијаграмот на колото, ние доделивме соодветен број на пин на секоја боја на LED диодите. Воведуваме toggle функција за вклучување што зема булова (boolean) вредност за секоја светилка. Врз основа на овие вредности, 0 или 1 се запишува на соодветниот GPIO што прави излезот да биде 3.3V – светилката да светне или 0V.

## Server Setup

Следното што ни треба е HTTP и WebSocket сервер.

server.js

const path = require( 'path' );

const express = require( 'express' );

const socketIO = require( 'socket.io' );

// import LED control API

const { toggle } = require( './led-control' );

// create an express app

const app = express();

// send `index.html` from the current directory

// when `http://<ip>:9000/` route is accessed using `GET` method

app.get( '/', ( request, response ) => {

response.sendFile( path.resolve( \_\_dirname, 'webapi/index.html' ), {

headers: {

'Content-Type': 'text/html',

}

} );

} );

// send asset files

app.use( '/assets/', express.static( path.resolve( \_\_dirname, 'webapi' ) ) );

app.use( '/assets/', express.static( path.resolve( \_\_dirname, 'node\_modules/socket.io-client/dist' ) ) );

// server listens on `9000` port

const s = server.listen( 9000, () => console.log( 'Express server started!' ) );

// create a WebSocket server

const io = socketIO( server );

// listen for connection

io.on( 'connection', (socket) => {

console.log('A client connected', socket.id );

// listen to `led-toggle` event

socket. on( 'led-toggle', ( data ) => {

console.log( 'Received led-toggle event.' );

toggle( data.r, data.y, data.g, data-l ); // toggle LEDs

} );

//static variable for current status

var lightvalue = 0;

//Watch for hardware interrupts on pushButton  
 pushButton.watch(function (err, value) {    if (err) {      console.error('There was an error', err);

      return;  
    }  
    lightvalue = value;

//send button status to client  
 socket.emit('light', lightvalue);  });

//get light switch status from client  
  socket.on('light', function(data) {   
    lightvalue = data;

//only change LED if status has changed  
 if (lightvalue != LED.readSync()) {

//turn LED on or off

      LED.writeSync(lightvalue);   
    }  
  });  
  
  
process.on('SIGINT', function () {

// Turn LED off

  LED.writeSync(0);

// Unexport LED GPIO to free resources  
LED.unexport();

// Unexport Button GPIO to free resources  
 pushButton.unexport();

//exit completely  
 process.exit();});

});

Во горната датотека server.js, ги додаваме express и socket.io библиотеките. Бидејќи сакаме да ги контролираме GPIO пиновите од оваа датотека исто така треба да ја воведиме и функција toggle од датотеката led-control.js.

Прво, креираме HTTP сервер што не е ништо друго туку express апликација. Функција Еxpress() враќа таква апликација. Потоа треба да поставиме неколку рути за да ги опслужиме .html и други assets датотеки. Користиме app.get ('/', handler) што значи секој што пристапува до нашата веб-апликација со / URL патека, ќе ја добие датотеката index.html сместена во webapi директориумот. Исто така, треба да се погрижиме да испратиме точни заглавија или headers. Следно нешто е то setup assets delivery. Бидејќи на датотекaта index.html и се потребни и main.js и style.css, ќе искористиме стандарден express.js setting to deliver assets.

Датотеката socket.io.js е исто така потребна во веб-апликацијата за комуникација со самиот SocketServer што го градиме. Оваа датотека се наоѓа во директориумот node\_modules / socket.io-client / dist, ние едноставно го упатуваме express.js да ги бара the assets датотеките во рамките на овој директориум, ако не се најдени во web-app директориумот.

Откако нашата express апликација е правилно конфигурирана, креираме HTTP сервер инстанца од оваа апликација. Со app.listen () повикот, HTTP-серверот се стартува на порт 9000. Аpp.listen повикот ја враќа референцата на HTTP-серверот што започна. Ова е она што му треба на нашиот WebSocket сервер.

Креираме WebSocket сервер користејќи ја инстанцата на HTTP серверот со socketIO (сервер). За разлика од HTTP серверот, врската со WebSocket серверот е постојана. Оттука, комуникацијата помеѓу клиентот и серверот се случува преку настани(events).  
  
Главната идеја зад Socket.IO е дека можеме да испраќаме и примаме настани со какви било податоци. Било какви објекти можат да бидат кодирани, а исто така се поддржани и бинарни податоци.

Променливата io го претставува WebSocket серверот. Кога клиент (веб-апликација) е поврзана со овој сервер, (connection event) настанот за конекција е примен од серверот. Овој настан содржи податоци за конекцијата на клиентот, како што е ID на клиентот исто така како и API за комуникација само со поврзаниот клиент. Откако ќе се воспостави врската помеѓу серверот и клиентот, серверот може да емитува и слуша различни /custom настани од и до клиентот. Секој настан може да содржи дополнителни податоци или payload.

Сепак, во горенаведениот пример, ние само го слушаме led-toggle настанот предводен од клиентот. Клиентот го испраќа овој настан кога ќе притисне на копчето во веб-апликацијата. Овој настан содржи информации за состојбата (вклучена или исклучена) на различните копчиња. Врз основа на led-toggle настанот, можеме да ја смениме состојбата на GPIO-пиновите поврзани со RGB LED светлата. Бидејќи веќе имаме API за да се справиме со таа датотека led-control.js, ние едноставно ја повикуваме функцијата toggle кога ќе го прими настанот што се води со вклучување односно led-toggle event.

Веб-апликација е само веб-страница која обезбедува корисни статични информации или многу богат кориснички интерфејс за комуникација со серверот.  
Во нашиот случај, дури и нашата веб-апликација ќе содржи само некои копчиња на кои треба да кликнеме, ние треба да комуницираме со серверот за да ги емитуваме овие настани со притискање на копчињата. Бидејќи треба да ги испратиме овие настани во реално време, ние користиме WebSocket врска преку традиционална HTTP врска.  
Прво, треба да дизајнираме релативно добар кориснички интерфејс.

Во продолжение пишуваме код за index.html датотеката која обезбедува кориснички интерфејс со часовник, временска прогноза, div со три копчиња со кои ги контролираме светилките, toggle кога е притиснато копчето за update и live streaming.

Исто така, треба да го обрнеме внимание на the asset files import. Откако го конфигуриравме express серверот да ги доставува asset files од /assets path , тоа е патеката која ние треба да ја користиме во .html фајлот за да ги импортираме assets како слики, икончиња и слично. Пример:

<link rel="stylesheet" href="/assets/style.css"/>

Во index.html датотеката го додаваме <script> tag-от со следниот код за активација на копчето:

<script>

//load socket.io-client and connect to the host that serves the page  
let socket = io();  
window.addEventListener("load", function(){

  let lightbox = document. getElementById("light");

//added event listener   
 lightbox.addEventListener("change", function() {

//send button status to server (as 1 or 0)  
 socket.emit("light", Number(this.checked));  });  
});

//get button status from client  
socket.on('light', function (data) {

//toggle according to push button on Raspberry Pi

document.getElementById("light").checked = data;

//send push button status to back to server  
 socket.emit("light", data); });  
</script>

Последното што ни преостанува е да ја имплементираме вистинската логика на клиент-сервер комуникацијата во следната датотеката main.js.

main.js

// get button elements

let button\_red = document. getElementById( 'button-red' );

let button\_yellow = document. getElementById( 'button-yellow' );

let button\_green = document. getElementById( 'button-green' );

// initial button states

let button\_red\_state = false;

let button\_green\_state = false;

let button\_blue\_state = false;

// check for active connection

let isConnectionActive = false;

// connect to the Web Socket server

let connection = io (‘http://192.168.1.5:9000’);

// when connection is established

connection.on( 'connect', () => {

isConnectionActive = true;

} );

connection.on( 'disconnect', () => {

isConnectionActive = false;

} );

// WebSocket event emitter function

let emitEvent = function(event) {

if( ! isConnectionActive ) {

return alert( 'Server connection is closed!' );

}

// change button state

if( event.target.id === 'button-red') { button\_red\_state = ! button\_red\_state; }

if( event.target.id === 'button-yellow') { button\_yellow\_state = ! button\_yellow\_state; }

if( event.target.id === 'button-green') { button\_green\_state = ! button\_green\_state; }

// emit `led-toggle` socket event

connection.emit( 'led-toggle', {

r: button\_red\_state,

y: button\_yellow\_state,

g: button\_green\_state,

} );

};

// add event listeners on button

button\_red.addEventListener( 'click', emitEvent );

button\_yellow.addEventListener( 'click', emitEvent );

button\_green.addEventListener( 'click', emitEvent );

Во горенаведената JavaScript main.js датотека, создадовме референци до секое индивидуално копче во ДОМ и ја одржуваме состојбата на секое копче (вклучено или исклучено) преку линиите бр. 1 до бр. 9. Со импортирање на датотеката socket.io.js во претходниот index.html, таа ни обезбедува API за комуникација на високо ниво со WebSocket серверот што го изградивме со употреба на socket.io. Тоа го обезбедува интерфејсот io кој се користи за поврзување со далечински/remote SocketServer креиран со употреба на socket.io. Во горенаведениот пример, ја користевме IP адресата на Raspberry Pi. Бидејќи се обидуваме да се поврземе со сервер, треба да го обезбедиме и бројот на портот на тој сервер за да направиме успешна socket конекција.  
  
Откако успешно ќе се креира socket врската, таа содржи API за комуникација со socket серверот. Веднаш штом врската e oстварена, таа го прима настанот за поврзување (connection event) од серверот. Ова е безбедна област за иницијализирање на какви било зависни од конекција (connection dependent) променливи или за испраќање настани до серверот. Во случајот ние ги слушаме настаните со кликнување на копчето и откако ќе биде притиснато копче, ние го емитираме led-toggle настанот до WebSocket серверот на објектот за врска(connection object).

Повикот connection. Emit() прима два аргументи. Прво е името на настанот што треба да се емитува и втор е payload-oт или товарот што треба да се испрати заедно со настанот. Структурата на носивоста(payload) на настанот треба да одговара на структурата на носивоста што ја очекува SocketServer (видете ја датотеката server.js).

Во следниот фајл app.js го разработуваме кодот за временската прогноза што собира податоци од http://api.openweathermap.org api провајдерот.

app.js

const weather = {};  
weather.temperature = {  
 unit : "celsius"}

//Consts & Vars  
const KELVIN = 273;

// API KEY  
const key = "39fbf2026f2c7591e52ed1125110a4d5";

// Check if browser supports geolocation

if('geolocation' in navigator){  
 navigator.geolocation.getCurrentPosition(setPosition, showError);  
}else{  
notificationElement.style.display = "block";  
notificationElement.innerHTML = "<p>Browser doesn't Support Geolocation</p>";}

// Set user’s position

function setPosition(position){  
let latitude = position.coords.latitude;  
let longitude = position.coords.longitude;  
getWeather(latitude, longitude);}

// Get weather from api provider

function getWeather(latitude, longitude){  
let api = `http://api.openweathermap.org/data/2.5/weather?lat=${latitude}&lon=${longitude}&appid=${key}`;  
 fetch(api)  
 .then(function(response){  
 let data = response.json();  
 return data;})  
 .then(function(data){   
 weather.temperature.value = Math.floor(data.main.temp - KELVIN);  
 weather.description = data.weather[0].description;  
weather.iconId = data.weather[0].icon;  
weather.city = data.name;  
weather.country = data.sys.country;})  
 .then(function(){  
 displayWeather();});}

video.js

//Loads the IFrame Player API code asynchronously  
 let tag = document.createElement('script');

      tag.src = "https://www.youtube.com/iframe\_api";  
 let firstScriptTag = document.getElementsByTagName('script')[0];  
 firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);  
//Function that creates an <iframe> (and YouTube player)

      let player;  
 function onYouTubeIframeAPIReady() {  
 player = new YT.Player('player', {  
 height: '390',

width: '640',  
 videoId: 'M7lc1UVf-VE',  
 events: {  
'onReady': onPlayerReady,  
'onStateChange': onPlayerStateChange } }); }

 // The API will call this function when the video player is ready

      function onPlayerReady(event) {  
 event.target.playVideo();

      }

 // The API calls this function when the player's state changes  
 //The function indicates that when playing a video (state=1),  
 //the player should play for six seconds and then stop

let done = false;  
function onPlayerStateChange(event) {  
if (event.data == YT.PlayerState.PLAYING && !done) {  
setTimeout(stopVideo, 6000);  
done = true;  } }function stopVideo() {  
player.stopVideo();

      }

### Чекор по чекор

## 

## The Command Line

Пред да започнеме со кодирање, прво треба да ги прочитаме и научеме командите за работа во терминалот. На самата страна на Raspberry Pi е посочена следната книга како есенцијална во работењето со Raspberry Pi.



## Се поврзуваме на безжична конекција

Го отвораме термналот и ја испишуваме следната наредба:

sudo nano /etc/wpa\_supplicant/wpa\_supplicant.conf

На крајот од фајлот што ќе ни се отвори го додаваме id и пасвордот на мрежата што сакаме да се поврземе, како на пример:

network={

ssid="Таlevska"

psk="Password123"

}

Откако ќе поставиме интернет конекција, ја подготвуваме околината за работа, односно

### Го инсталираме Node.js, npm и express:

sudo apt install nodejs

или за да го инсталираш прво треба да провериме каков тип на ARM chip имаме. Чип-сетот се разликува од еден модел до друг:

**ARM8:** all RPi 4, all RPi 3, RPi 2 Model B v1.2  
 **ARM7:** RPi 2 Model B  
 **ARM6**: all Zero, all RPi 1

Или едноставно внесуваме uname -m во терминалот. Следно:

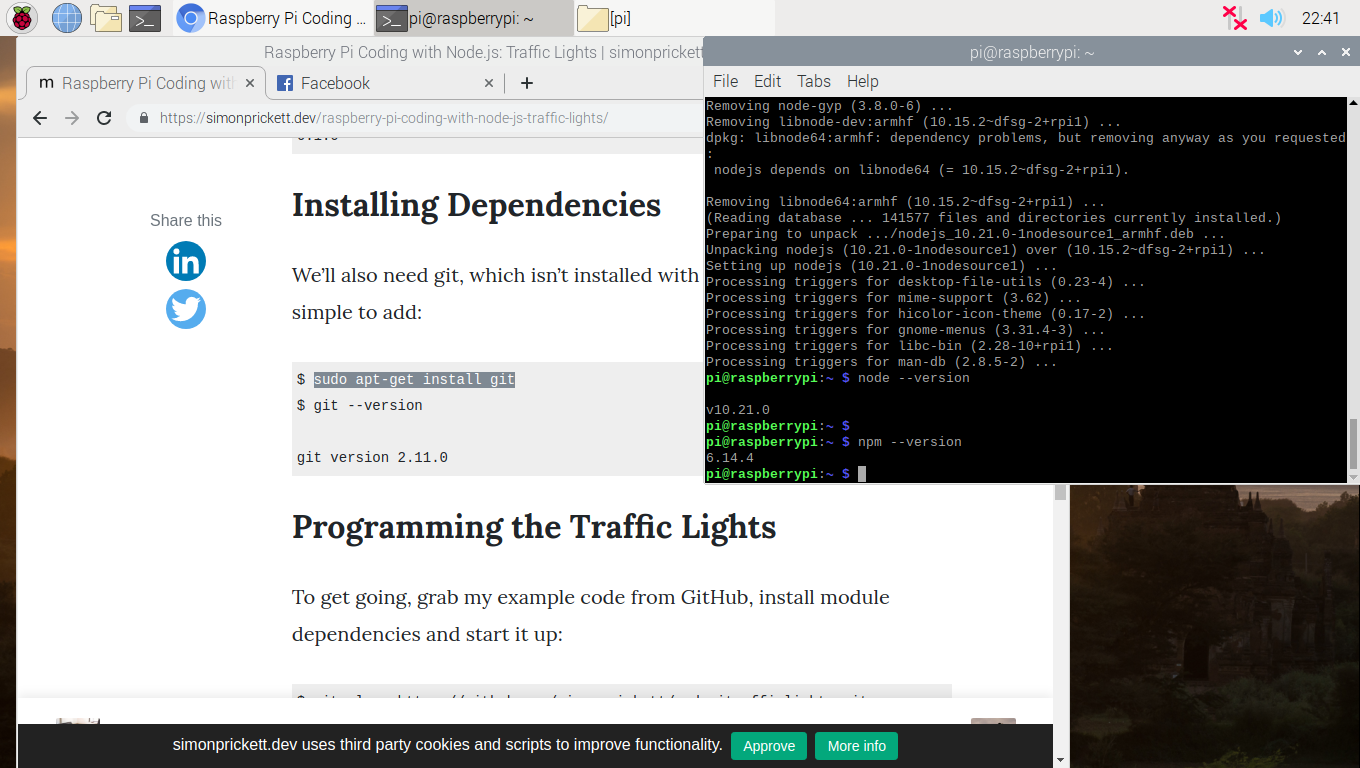
wget https://nodejs.org/dist/v10.16.1/node-v10.16.1-linux-arm64.tar.xz

tar -xJf node-v10.16.1-linux-armv6l.tar.xz

cd node-v10.16.1-linux-armv6l/  
sudo cp -R \* /usr/local/

### Проверуваме дали сè е правилно инсталирано

Сега, кога веќе имаме инсталирано Node.js, можеме да го користиме терминалот за да бидеме сигурни дека сè е правилно инсталирано. Со следнава команда на промптот: *node-v* и *npm -v* ќе ја покаже верзијата на Node.js и npm што ја инсталиравме.



Сл.9 Checking the node.js and NPM version

## Setup Nodemon за автоматско рестартирање на Node.js серверот за апликации

Постојат неколку алатки што можеме да ги користиме за автоматско рестартирање на Node.js серверот по секоја промена. Најпреферирана алатка е Nodemon. Ако ја погледнеме датотеката package.json, ќе видите дека нодемонот е наведен под devDependency, па може да се користи веднаш.

### Ги инсталираме потребните модули

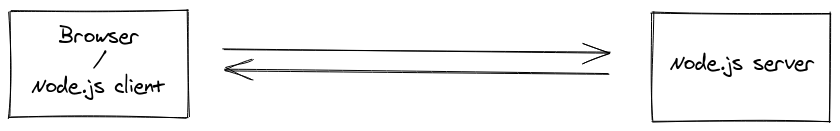
За да работиме со GPIO pins на Raspberry Pi со Node.js код, ќе користиме модул кој се вика"onoff". Го инсталираме со помош на npm:

sudo npm install onoff

Исто така и за да користиме Web sockets треба да го инсталираме socket.io module.

sudo npm install socket.io

Note: Socket.IO ни овозможува да имаме real-time, двонасочна event-базирана комуникација. Се состои од: Node.js server и JavaScript client library за пребарувачот (или Node.js client).



Socket.IO НЕ е имплементација на WebSocket. Иако Socket.IO навистина го користи WebSocket како транспорт, тој додава дополнителни метаподатоци на секој пакет. Затоа не секогаш WebSocket клиент нема да може успешно да се поврзе со Socket.IO сервер и Socket.IO клиент нема да може да се поврзе со обичен WebSocket сервер.

### Во комбинација со Express

### Почнувајќи со верзија 3.0, експресните апликации станаа request handler функции што ги пренесуваме на инстанците http или http Server. Треба да го наведеме серверот на socket.io, а не на функцијата од express апликација. Исто така, проверуваме . listen да слуша на серверот, а не на апликацијата.

const app = require('express')();

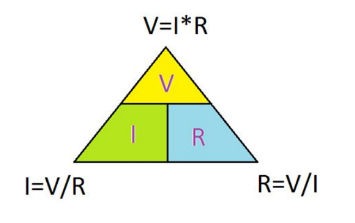
const server = require('http').createServer(app);

const io = require('socket.io')(server);

io.on('connection', () => { /\* … \*/ });

server.listen(9000);

## Поврзување на лед-диодите и копчето на Raspberry-Pi

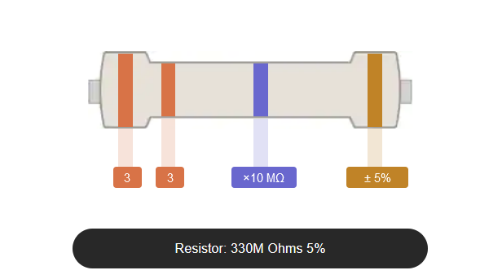
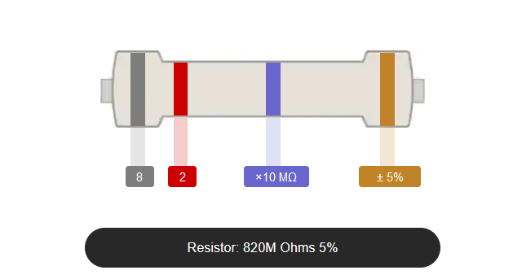
Пред да започнеме да пишуваме код треба да ги поврземе сијаличките на breadboard-от со Raspberry pi. Во нашиот проект користиме три лед-диоди кои ги поврзуваме со катодата(пократката нога) на негативниот крај или ground и анодата(подолгата нога на сијаличката) на напојувањето или power source. До нив соодветно ги поврзуваме пинските каблиња кои ни требаат да бидат машко-женско, за женското да го поврземе на ground GPIO pin и машкото на ground bus, каде цела колона на breadboard-от е електрично поврзана. Напонот на едно gpio е 3.3V , една лед користи само 2.1V, што значи треба да употребиме отпорници со најмалку 75 оми:

*Ω*

Ние користиме комбинација од 820M Ohms и од 330M Ohms отпорници за сијаличките и копчето соодветно. На следниот линк во продолжение можеме да искористиме калкулатор на отпорници, со внесување на конкретни параметри за отпорникот како Number of Bands и Resistor Parameters: <https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-resistor-color-code-4-band>.

*2nd Band of Color*

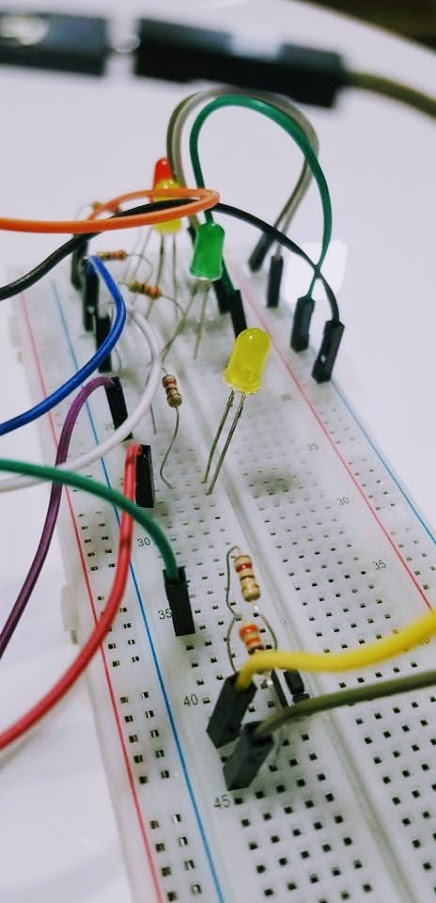
*Multiplier*



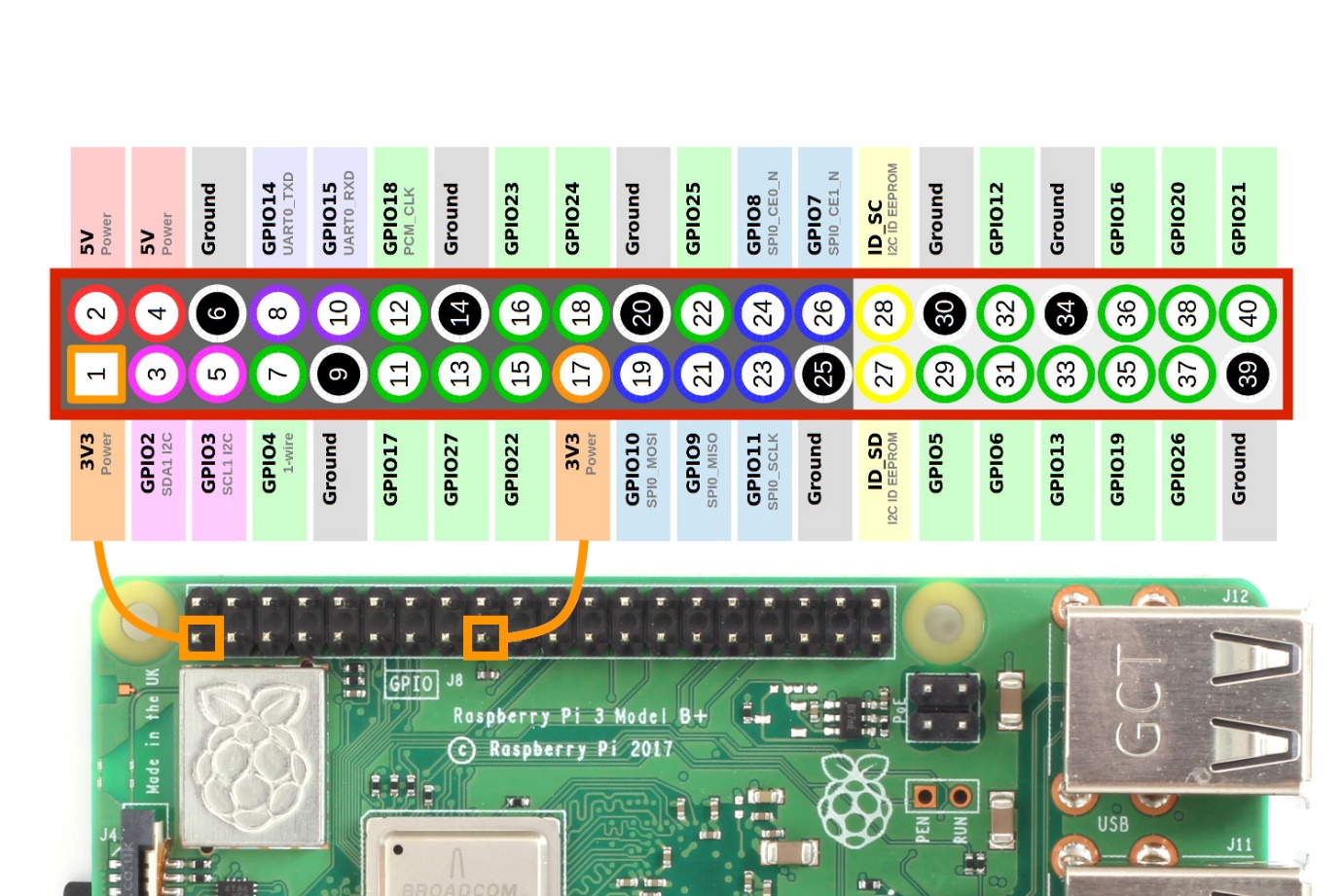
*1st Band of Color*



*Tolerance*



Сл.10 Како се поврзани лед диодите на breadboard-от

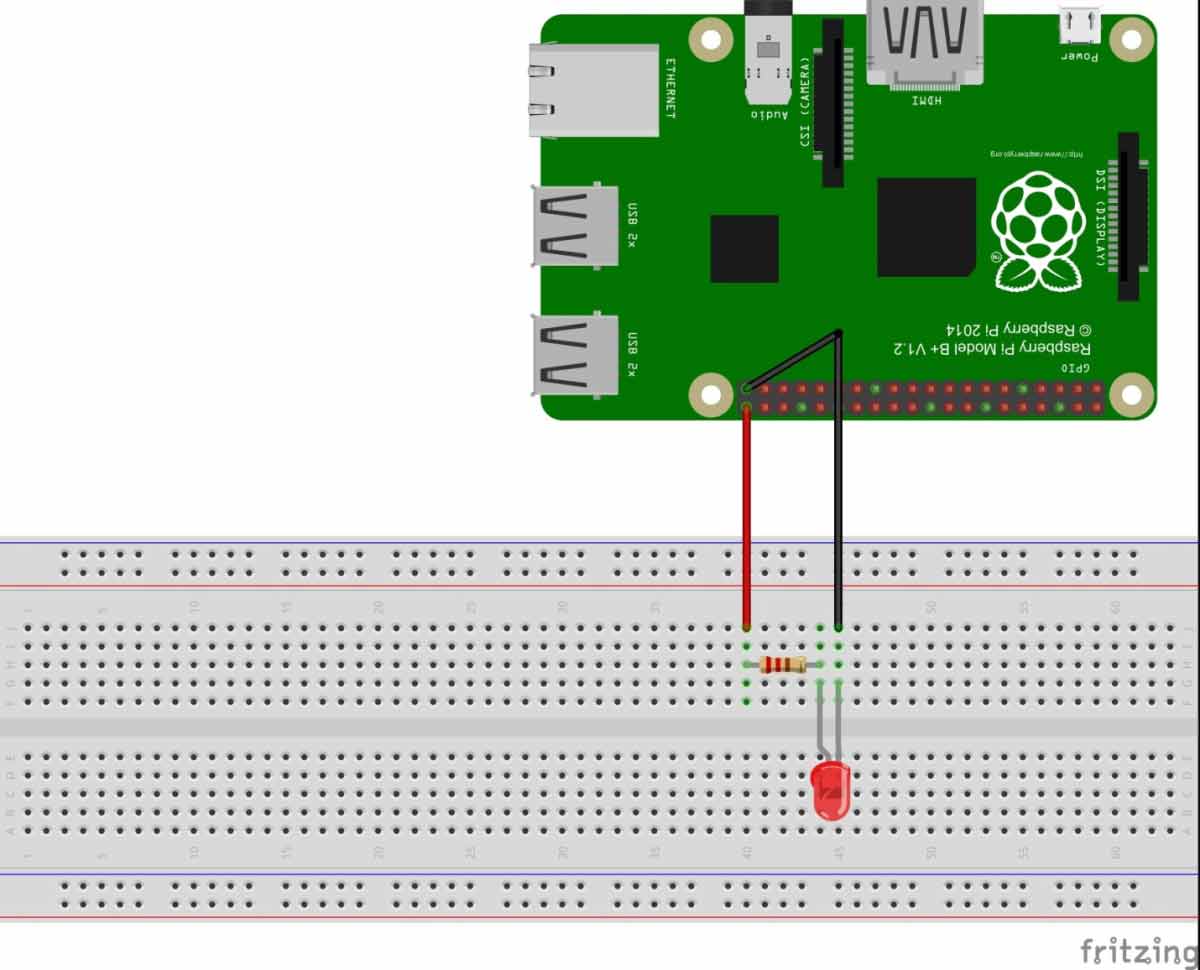
ја поврзуваме на GPIO13

 ја поврзуваме на GPIO19

 ја поврзуваме на GPIO26



го поврзуваме на GPIO14



Сл.11 GPIO pins Raspberry Pi



Long leg GPIO pin

Short leg (GND)



Сл.12 Поврзување на лед диодите со Raspberry Pi

Следно го пишуваме кодот, односно ги креираме соодветните фајлови кои ни требаат, и тоа го правиме преку терминалот со наредбите:

sudo touch server.js

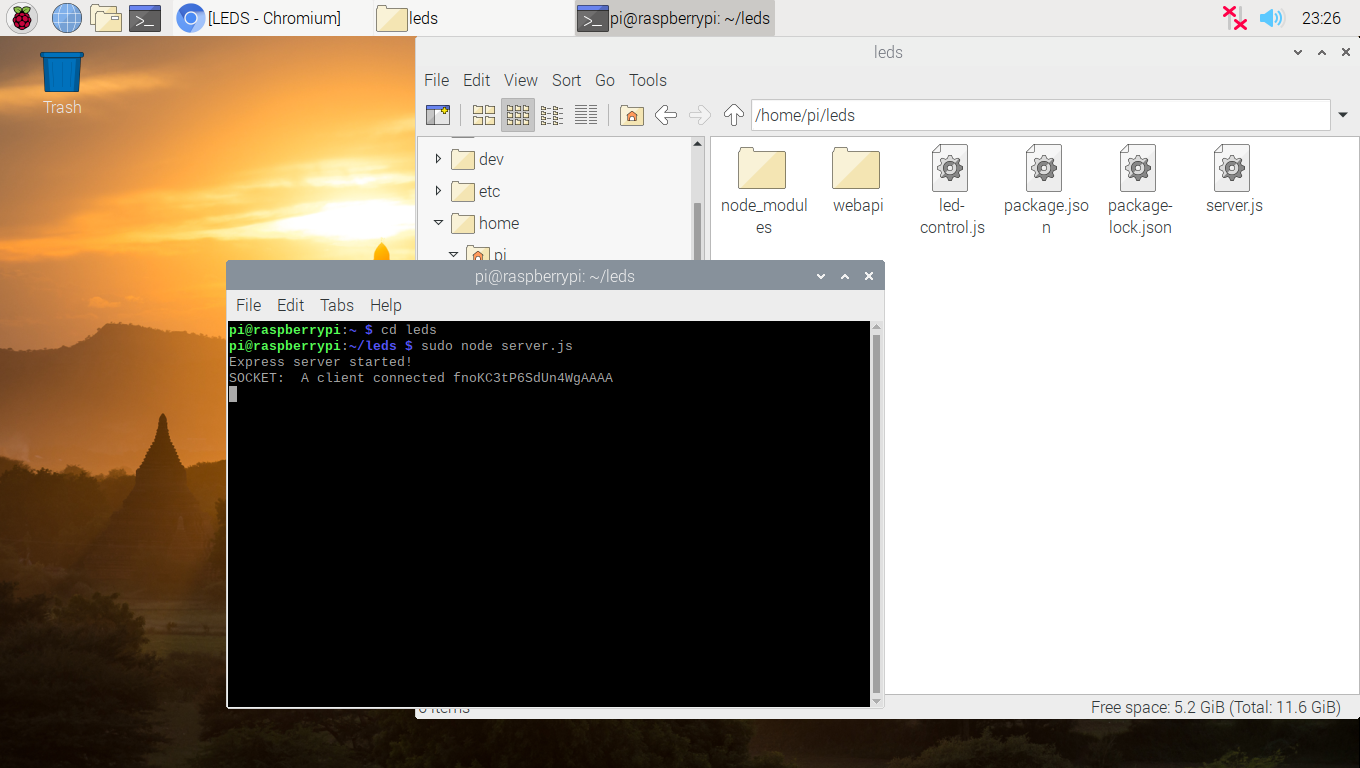
sudo nano server.js //edit the file

## Стартување на веб-апликацијата и Socket Server

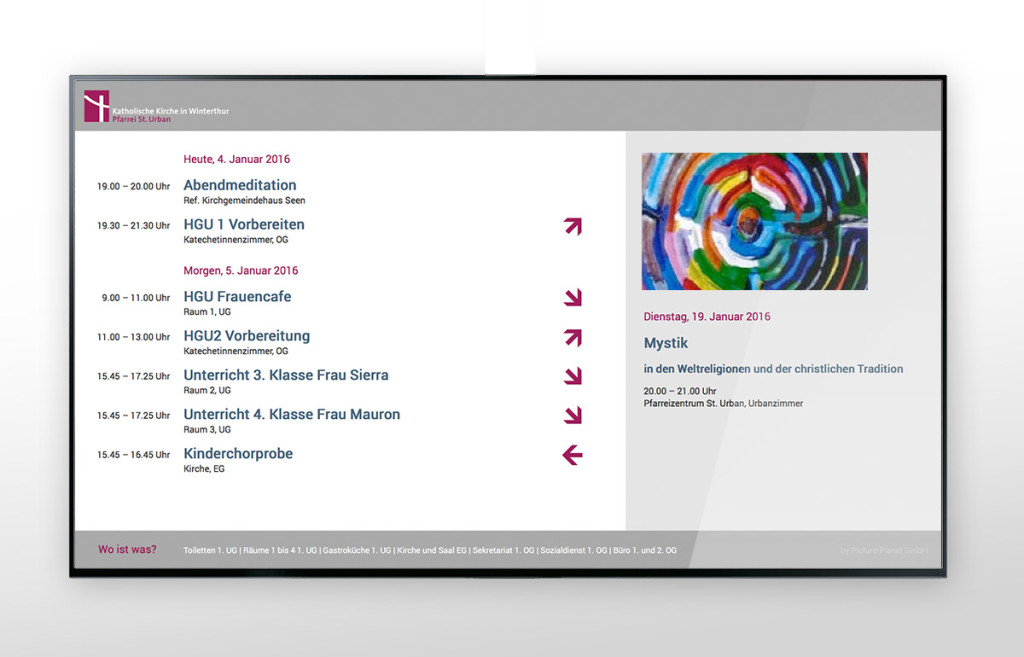
Сега кога сè е на свое место, време е да го ја стартуваме апликацијата. Единственото нешто што треба да направиме е да ја стартуваме датотеката server.js со Node.js. Ние го правиме тоа со командата: *node server.js.*

Откако ќе ја извршиме оваа команда, ќе се стартува HTTP-сервер на порт 9000. Исто така, ќе започне WebSocket-сервер, кој работи на истиот порт. Можеме да отвориме URL: http: // <rpi-host>: 9000 во прелистувачот за пристап до веб-апликацијата што ја изградивме.

Откако ќе ја отвориме веб-апликација, таа ќе се поврзе со серверот WebSocket.   
Доколку оваа порака се испише во терминалот каде што е извршена командата за node server.js, тогаш комуникацијата клиент-сервер е воспоставена. Сега со притискање на Црвеното, Зеленото или жолтото копче во веб-апликацијата ќе видеме дека Raspberry Pi ги вклучува LED сијаличките.



Сл.13 Стартување на серверот



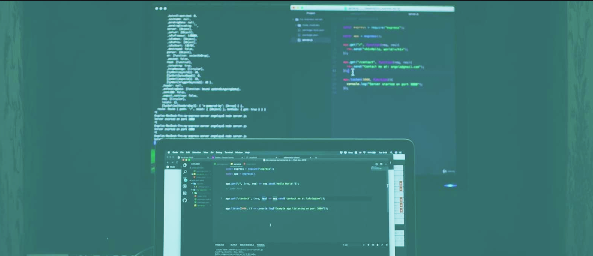
**Факултет за   
информатички и  
комуникациски технологии**

Сл.14 Инфо дисплеј



Со кликнување на копчињата (play), (pause), (stop) светнуваат соодветно зелената, жолтата и црвената сијаличка. Со нивно повторно кликнување тие се изгаснуваат. Копчето работи на принцип што додека го притиснуваме односно задржуваме на него жолтата сијаличка светнува и лизгачот под статусот се поместува надесно.

**4**



# Заклучок

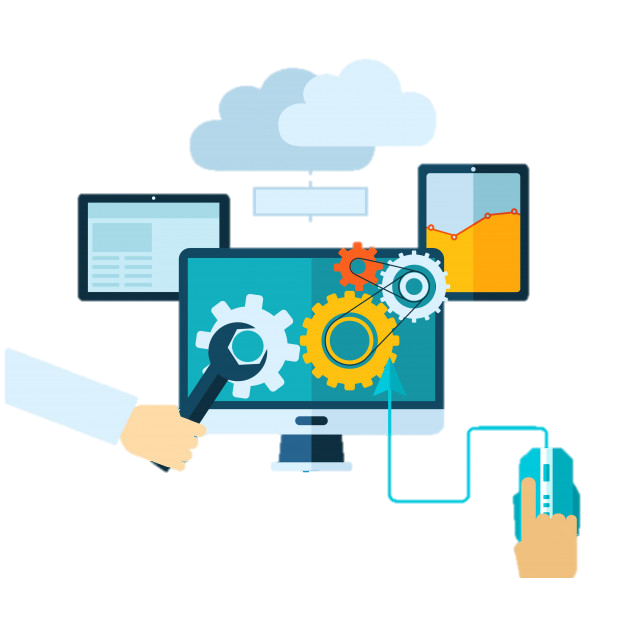
IoT е навистина следниот чекор во информатичката ера. Интернетот на нештата може да ги промени нашите секојдневни животи, нашето секојдневно работење и нашите секојдневни заедници. Технологијата го поставува нашиот живот на дланка, веќе нашите паметени телефони стануваат центар на нашите секојдневни активности. IoT не е само едноставна идеја, туку, веќе е главен дел од нашите животи.

„Интернетот ќе исчезне“, рече претседателот на Гугл, Ерик Шмит. „Ќе има толку многу IP-адреси, толку многу уреди, сензори, работи што ги носите, работи со кои комуницирате, па дури и нема да почувствувате. Ќе биде дел од нашето присуство цело време. Замислуваме дека влегуваме во соба, а просторијата е динамична, и со наша дозвола ќе можеме да комуницираме со работите во собата.

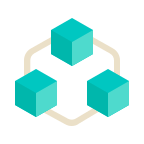
Моќта на IoT ќе расте само кога компјутерската моќ на уредите се зголемува. Во принцип, компјутерската моќ на одделни паметни уреди ќе продолжи да ги проширува способностите и можностите на секој паметен уред. Како што можностите се шират и поврзаноста се зголемува, потенцијалот за паметни компании, паметни градови и паметни домови исто така се зголемува експоненцијално.

IoT е идеална технологија што се појавува за да влијае на секој домен преку обезбедување нови развојни податоци и потребни пресметковни ресурси за создавање револуционерни апликации, со потенцијал драматично да ја зголеми достапноста  
на информации, и веројатно ќе ги трансформира компаниите и организациите  
буквално во секоја индустрија низ целиот свет. Со помош на Raspberry Pi што е одлична платформа за создавање вградени системи за ниска цена, се олеснуваа употребувањето на модулите користејќи едноставна ефтина електроника и со потребната конфигурација можат да се создадат функционални и флексибилни системи.

На крајот, со цел да се приклучиме на IoT мисијата, треба да се запрашаме: „Дали треба ова нешто да се поврзе?“ Ако да, „Како да го поврземе?“



## Користена литература:



https://www.raspberrypi.org/documentation/

<https://nodejs.org/en/about/>

Raspberry Pi Cookbook

https://www.raspberrypi.org/magpi-issues/Essentials\_Bash\_v1.pdf

Building the Web of Things: With examples in Node.js and   
Raspberry Pi Book  
https://www.manning.com/books/building-the-web-of-things

Internet-of-Things (IoT) Systems Architectures, Algorithms, Methodologies

Dimitrios Serpanos & Marilyn Wolf

<https://thinkster.io/tutorials/node-json-api/initializing-a-starter-node-project>

Node.js in Practice by Alex Young & Marc Harter

https://www.programmer-books.com/node-js-in-practice-pdf/

<https://developers.google.com/youtube/v3/getting-started>

https://socket.io/get-started/chat

https://socket.io/docs/

