**Глава 2. Теоретична част**

Дипломната работа се състой от на два самостоятелни логически модула. Модул за осигуряване на достъп състоящ се от четец и сървърна част, и модул за администрация състоящ се от website обслужван от сървърна част.

**2.1. Модул за осигуряването на контрол на достъп**

**Принцип на работа**

Четеца е изграден на базата на Arduino Uno REV3 модел [1], MFRC522 RFID модул [2], пасивен RFID идентификатор MIFARE S50 [3] под формата на чип прикрепен към ключодържател и Wi-Fi модул ESP8266 [4]. Ардуиното играе ролята на управляващ елемент за Wi-Fi и RFID модулите. Двата модула са свързани към съответни цифрови щифтове на ардуино като комуникацията се осъществява чрез електрически сигнали.

При доближаване на чипа до четеца на максимално разстояние от няколко сантиметра, той бива захранен от електромагнити вълни на честота от 13.56 MHz изпращани от четеца благодарение на принципa за индуктивно-резонансен пренос на енергия [5], при което чипа изпраща обратно своята идентификационна информация чрез електромагнитна вълна. RFID модула декодира информацията, след което бива предадена на ардуиното, което от своя страна изпраща HTTP GET заявка, която съдържа идентификационната информация за чипа и уникален идентификационен номер за ардуиното, към REST API-то посредством Wi-Fi модула. Сървъра и четеца са свързани към една и съща Wi-Fi мрежа. Заявката се изпраща в JSON формат [6]

Сървъра сравнява получената информация със информацията налична за четеца в база данни. В базата данни се пази информация за минималното ниво на достъп нужно на даден потребител за да му бъде позволен достъп. REST API-то и базата данни комуникират помежду си чрез SQL комуникационен протокол [7], който работи на базата на TCP/IP протокола.

В базата се пази информация за нивото на достъп на всеки потребител / идентификатор както и минималното ниво на достъп, което изисква дадена точка за достъп (четец) за да бъде допуснат потребителя през нея.

Сървъра връща информация от сравнението в текущата HTTP заявка, която зависи от това дали е намерена информация за четеца и потребителя в базата данни, и дали нивото на достъп на потребителя е достатъчно. Когато потребителя има нужното ниво на достъп се връща HTTP отговор с код 200 (ОК), когато няма достъп се връща 401 (Unauthorized), а когато няма информация за потребителя се връща 404 (Not found).

Получения отговор от сървъра се декодира от Wi-Fi модула и се предава като информация на ардуиното, което от своя страна дава индикация дали потребителя има право да достъпи защитения ресурс.

При всяка заявка от четеца към сървъра в базата се прави запис за събитието, като в последствие тази информация се ползва от модула за администрация за анализ на данните под формата на генериране на справки.

За осигуряване на качествена поддръжка на класическа система работеща с база данни, мрежи се генерира подробен log файл при всяка възникнала грешка на сървъра.

За осигуряването на висока защита на комуникация между четеца и сървъра се използва разновидност на протокола HTTP – HTTPS. Защитата се осигурява чрез криптиране на данните в двете посоки.

**Блокова схема**



Фигура 2. Блокова схема на комуникацията между Wi-Fi базиран ардуино четец и REST API посредством HTTP протокол

**RFID**

RFID е безжична система за идентификация чрез радио вълни. Състой от четец и идентификатор. В зависимост от нуждите на системата RFID се дели на няколко честотни диапазона.

|  |  |  |  |
| --- | --- | --- | --- |
| **Честотен диапазон** | **Разстояние** | **Скорост на трансфер** | **Приложение** |
| 120 – 150 kHz ниско честотни (LF) | До 10 см | ниска | Идентификация на животни и предмети |
| 13.56 MHz  високо честотни (HF) | До 1 м | ниска до средна | Идентификация, контрол на достъп, инвентаризация, обмен на данни |
| 433 MHz утра-високо честотни  (UHF) | 1 – 100 м | средна | Контрол на достъп в комбинация с активен таг |
| 865 – 868 MHz (Европа)  902 – 928 MHz (САЩ) ултра-високо честотни | 1 – 12 м | средна до висока | Използва се при релсови пътища |
| 2450 – 5800 MHz  микровълнови | 1 – 2 м | висока | 802.11 WLAN Bluetooth |
| 3.1 – 10 GHz микровълнови | До 200 м | висока |  |

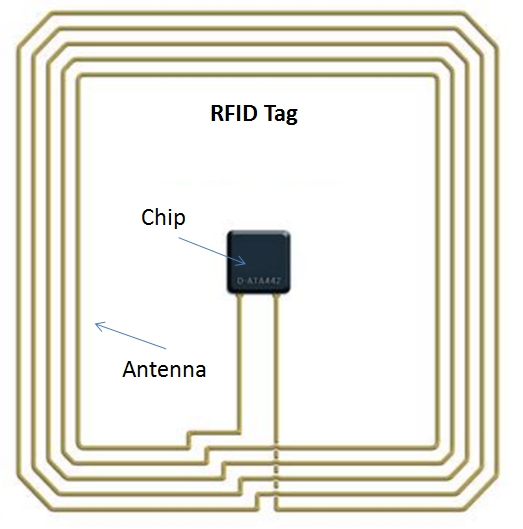
Таблица 1. Честотни диапазони на RFID системите

Според начина на захранване на идентификатора RFID се дели на активна и пасивна система. Текущата дипломна работа има за цел изграждането на пасивна система.

При пасивната система идентификатора се захранва чрез пренос на енергия от четеца. Използват се за комуникация от близко разстояние.

**Идентификатор -** в най – простия си вид се състой от три елемента: интегрална схема, която се използва за съхранение, обработка на информация и модулация и демодулация на радио сигнала. Начин за преобразуване на сигнала от четеца до постоянен ток и антена за получаване и предаване на сигнал. Информацията на идентификатора се съхранява на енерго-независима памет. Елементите се вграждат най – често в самата пластика на карата, или в пластмасови обекти пригодени за прикачване към ключодържател.  
Пасивните идентификатори нямат батерия и тяхното захранване зависи изцяло от силата на сигнала от четеца. Информацията която се съдържа най – често е сериен номер. Серия номер се записва по време на производствения процес, като той няма връзка с конкретен обект или лице, а в последствие тази информация се установява най – често в база данни, към която се записват обработените данни. Има и пасивни идентификатори, които подлежат на манипулация и тяхната информация може да бъде презаписана. Скоростта на четене от тези чипове е относително ниска, като подобна операция отнема между 25 до 50 ms.

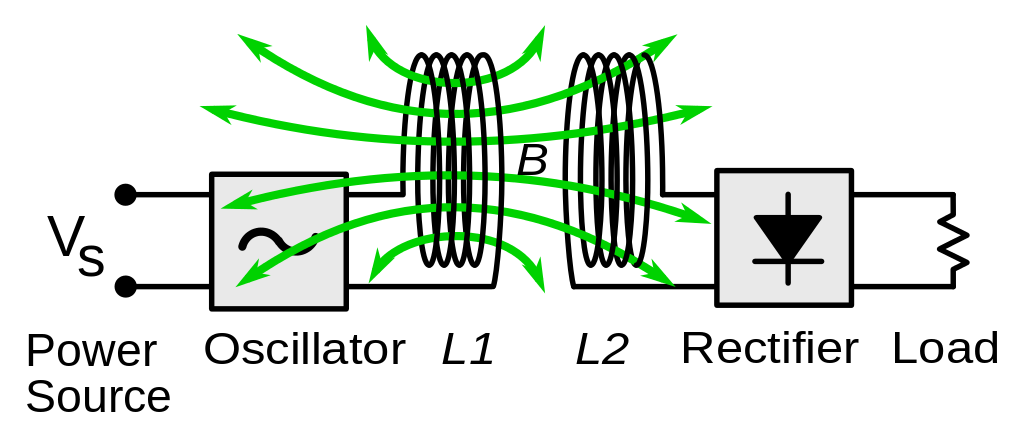
Антената на идентификатора събира енергия и я пренасочва към интегралната схема, като по този начин я захранва. Обема на енергията пряко зависи от размера и формата на антената. Също така антената е отговорна и за предаването на информацията, която е запазена върху идентификатора.



Фигура 6. Схема на организацията в RFID антена

**Четец –** той е отговорен за захранването на идентификатора, разчитането на данните изпратени от него и предаването им към сървърната част.

Начина на комуникация и захранване става на принципа на резонансно-индуктивен пренос на енергия. Антената на четеца се състой от бобина свързана към източник на променлив ток като по този начин се образува магнитно поле около него. Когато поставим идентификатора в близост до полето се създава електро-моторна сила по закона на Фарадей за електромагнитна индукция. Напрежението води създаване на електрически поток в бобината на идентификатора. Посредством токоизправител променливия ток се преобразува до постоянен, чрез който се захранва интегралната схема. При което антената на идентификатора също създава магнитно поле, което бива разчетено от антената на четеца. Двете антени са настроени да резонират към една и съща резонансна честота, което позволява по – оптимално използване на използваната енергия, което позволява пренос на енергия на по – големи разстояния.



Фигура 7. Индуктивно-резонансен пренос на енергия

**Wi-Fi**

Представлява технология за безжична комуникация базирана на IEEE 802.11 стандарта [8], който се използва за локална мрежова комуникация (LAN) и за достъп до интернет. [9]. Wi-Fi комуникацията използва пакети за пренос на данни, които се предават чрез радио вълни между отделните устройства. Това се постига чрез модулация и демодулация на носещата вълна. По стандарт всяко устройство разполага с MAC адрес, който представлява 48 битов уникален адрес. MAC адреса се използва за обозначаването на източника и получателя за даден пакет.

Самата технология следва препоръките на OSI модела, но е направена с идеята да е съвместима с Ethernet стандарта, който се използва за пренос на TCP/IP [13] трафик, който от своя страна не следва OSI модела.

TCP/IP [13] модела е създаден преди дефинирането на OSI модела. Той дефинира стандарти за пакетиране, адресиране, пренос, пренасочване и получавана на данни между клиент-сървър в една компютърна мрежа и е един от най – разпространените в днешно време. Функционалността на тези протоколи е организиране в четири абстрактни слоя, към които са класифицирани съответните протоколи според тяхната роля.

Двата най – важни протокола са съответно Transmission Control Protocol (TCP) и Internet Protocol (IP).

TCP - дефинира надежден начин за пренос на данни под формата на пакети между приложения. Едни от най – използваните приложения в днешно време като World Wide Web (WWW) и емайл клиентите използват този протокол.

Протокола гарантира получаване на изпратените пакети такива каквито са били изпратени от сървъра и тяхното им получаване в правилен ред за да може да се възстанови изпратеното съобщение. При възникване на грешка в мрежата или претоварване, някои пакети могат да бъдат изгубени или получени в грешен ред при клиента. Протокола засича тези проблеми и използва техника позната като ‚позитивно потвърждение‘ (positive acknowledgment) с повторно изпращане на данните. Това задължава получателя на данните да изпрати потвърждение, че пакетите са получени успешно. Ако проблема остане не разрешен, първоизточника на данни бива известен за проблема.

TCP сегмента (пакет) – TCP протокола приема данни под формата на пакети, като добавя TCP header, като по този начин създава TCP специфичен пакет. Сегмента се състой от header част и част от данни. Всеки header съдържа десет задължителни полета: порт на източника, порт на получателя, пореден номер, номер за потвърждение, отстояние на данните, резервирани данни, флагове, размер на прозореца, контролна сума, поле за начало / край на пакета (padding).

Преди изпращането на пакетите сървъра и клиента установяват връзка помежду си. Това става на принципа на three-way handshake. Клиента прави опит да се свърже със сървъра, сървъра от своя страна слуша на определен порт за опити за свързване с него: това се нарича пасивно отворена връзка (passive open). След като тя е установена, клиента може да започне изграждането на активно отворена връзка (active open), при което се изпълнява и three-way handshake-а.

1. Първата стъпка: SYN – при отваряне на активно отворена връзка клиента изпраща SYN до сървъра. Клиента записва произволна стойност A на мястото на поредния номер (sequence number) на пакета.
2. Втора стъпка: SYN-ACK – в отговор сървъра отговаря с SYN-ACK. Изпраща се нов пакет към клиента като номера за потвърждение (acknowledgment number) се записва като към поредния номер на получения пакета се добави единица (A + 1), а поредния номер на новия пакет се задава с нова произволна стойност (B).
3. Трета стъпка: ACK – накрая клиента изпраща ACK до сървъра. За поредния номер на пакета се ползва номера за потвърждение от получения пакет, а номера за потвърждение на новия пакет с едно повече от поредния номер на получения пакет.

На този етап вече има изграден двустранен канал за комуникация между клиента и сървъра.

IP – дефинира как да адресира и пренасочва всеки един пакет, така че да достигне до правилата дестинация на базата на записаните IP адреси в самите пакети. Най – разпространената версия на протокола е IPv4, но съществува и по – нова имплементация наречена IPv6, която има за цел да се справи с проблема на броя IP адреси, които се подържат от IPv4.

Всеки пакет е представен от два сегмент: header и данни – както при TCP. В header-a se включва информация за IP адреса на източника и на получателя, както и допълнителна информация нужна за пренасочването на пакета. Този пакет вече се нарича IP пакет.

IEEE 802.11 [8] стандарта е част от IEEE 802 [10] стандарта, като дефинира допълнителни протоколи Media Access Control (MAC) [11], които са част от каналния слой (Data Link) на OSI модела [12], както и протоколи част от физическия слой (PHY) на OSI модела [12], за имплементирането на WLAN Wi-Fi комуникация за различни честоти.

Data Link слой - отговорен за създаването, предаването и приемането на пакети – управлява физическия слой. Data Link се състой от два под-слоя: Logical Link Controller (LLC) и MAC. Когато LLC получи IP пакет от Network слоя, той го разширява като добавя допълнителна информация за неговото адресиране и го предава към MAC под-слоя. Пакети минали през LLC се наричат frame. MAC допълва адресиращата информация, с цел осигуряване на формат подходящ за Physical слоя.

Physical слой – отговорен за предаването на пакетите под формата на битове във физическа среда. В контекста на Wi-Fi тази среда на разпространение е въздух. Данните се модулират и се предават под формата на радио вълни.

Съществуват различни версии на Wi-Fi, които дефинират различни работни честоти, максимална площ на покритие и максимална скорост на предаване на данни.

|  |  |  |  |
| --- | --- | --- | --- |
| **Стандарт** | **Максимална скорост** | **Година на експлоатация** | **Честота** |
| Wi-Fi 6 (802.11ax) | 600 - 9608 Mbit/s | 2019 | 2.4 / 5 GHz  1 - 6 GHz ISM |
| Wi-Fi 5 (802.11ac) | 433 - 6933 Mbit/s | 2014 | 5 GHz |
| Wi-Fi 4 (802.11n) | 72 - 600 Mbit/s | 2009 | 2.4 / 5 GHz |
| 802.11g | 3 – 6 Mbit/s | 2003 | 2.4 GHz |
| 802.11a | 1.5 – 54 Mbit/s | 1999 | 5 GHz |
| 802.11b | 1 – 11 Mbit/s | 1999 | 2.4 GHz |

Таблица 2. Различни версии на Wi-Fi.

Wi-Fi е маркетинговото име на устройствата, които имплементират горе посочените стандарти.

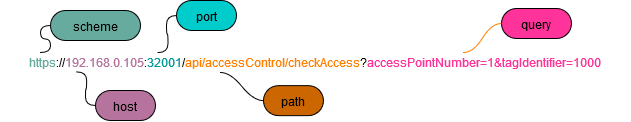
**HTTP**

Протокол за пренос на данни между сървър и клиент. Работи на Application слоя както на OSI модел така и на TCP/IP модела. HTTP е основния протокол, който се използва от WWW. Основните данните, които се пренасят са формати свързани с изграждането на Web страници: HTML документи, JavaScript файлове, CSS файлове, JSON документи, картинки, видеа, музика и други медия файлове [14], както и информация предназначена за сървъра от клиента: най – често под формата на JSON документи.

За да бъде достъпен един ресурс, той трябва да бъде идентифициран. За тази цел всеки ресурс е представен чрез уникален ресурсен идентификатор – Uniform Resource Identifier (URI). URI формат е текстово базиран и включва в себе си няколко основни елемента.

**Scheme**: вида на протокола който се използва  
**Host**: IP адрес или домейна  
**Port**: порт на който сървъра обслужва дадения website или REST API  
**Path**: фрагменти разделени с дясно наклонена черта ‘/’. Обикновено са описателни текстове, който дават информация за същността на ресурса  
**Query**: комбинация от двойка ключ и стойност допълващи информацията за целта на заявката.

Пример за URI



Фигура 3. URI стандарта разбит на основни компоненти

Клиента и сървъра комуникират чрез размяна на съобщения. Съобщенията изпратени от клиента се наричат request съобщения, а тези изпратени от сървъра се наричат response съобщения. Протокола е текстово базиран с цел по – лесното разчитане от човек.

Всяко request съобщение дефинира информация относно целта на заявката под формата на полета (headers) [15] състоящи се от име и стойност, и тяло на заявката в зависимост от вида на HTTP метода. Където най – често срещаните са: accept, accept-encoding, cookie, host, user-agent, content-length, content-type и други.

За да се окаже какво действие искаме да извършим на дадения ресурс, HTTP протокола дефинира така наречените методи (HTTP methods / verbs):

**GET –** заявката очаква даден ресурс да бъден върнат от сървъра в определен вид и нищо повече.

**HEAD –** заявката е същата като **GET,** но без да се връща самия ресурс. Полезно е когато искате информация за мета данни за ресурс, но без да ви трябва самия ресурс. Например: може да се върне информация колко е голям един файл, без да се налага целия файл да бъде част от response съобщението.

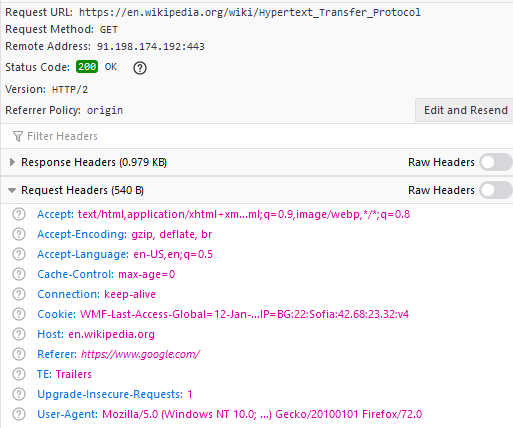
**POST –** заявката се състой от тяло и оказва на сървъра, че тялото на съобщението трябва да бъде третиран като нов ресурс, т.е. след успешна заявка, новия ресурс ще има собствен URI. Например: най – често се ползва, когато искате да запазите нов запис в базата данни на сървъра.

**PUT –** също като **POST**, оказва на сървъра, че тялото на заявката трябва да създаде нов ресурс под съответното URI. Различава се с това, че ако ресурс вече съществува, се очаква сървъра да модифицира стойността за това URI с тази изпратена от заявката.

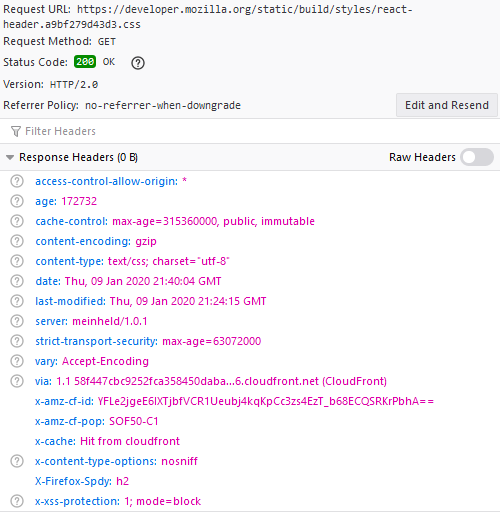
**DELETE –** оказва на сървъра, че ресурса със съответното URI трябва да бъде изтрит.

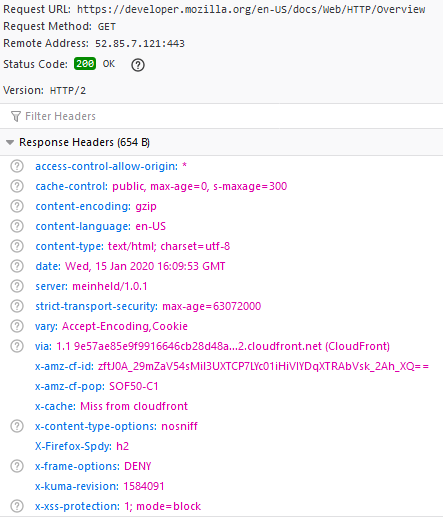
**PATCH –** оказва на сървъра, че искаме изцяло или частично да променим ресурса на опредено URI със ресурса, който изпращаме като част от заявката.

Пример за HTTP GET request съобщение направено на Firefox 72.0.1 към следната страница: <https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol>



Response съобщенията също дефинират набор от полета (headers) [15]: cache-control, content-encoding, content-length, content-type и други.

За да се окаже какъв е резултата от response заявката, в request-а HTTP протокола дефинира Status code. Те са числови стойности, които се разделят на пет групи в зависимост от числото с което започва:  
**1XX** – информационни.  
**2XX** – оказват, че заявката е била успешна. Пример: 200 – OK  
**3XX** – оказват, че заявката е пренасочена. Пример: 301 – оказва, че търсения ресурс е вече се намира на друго URI.  
**4XX** – оказват, че заявката е невалидна спрямо очакванията на сървъра. Пример: 404 – оказва, че поискания ресурс вече не съществува  
**5XX** – оказват, че на сървъра е възникнала грешка. Пример: 500 – грешка на сървъра.   
Пример за response съобщение HTTP GET заявка направено на Firefox 72.0.1 към: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>



**JSON**Формат за описване на данни. Силно заложен в HTTP комуникацията и е един от основните формати за обмен на данни между клиент и сървър. Представлява текстов формат имащ за цел да бъде лесно разчитан както от човек така и от компютър, като едновременно с това се опитва да е с минимален размер. Това е постигнато чрез просто представяне на данните под формата на колекция от ключ и стойност, където ключа е текст а стойност може да е текст, списък или друг JSON документ.

Пример за JSON обект, който описва информация за потребител:

{

"firstName": "vasil",

"lastName": "oreshenski",

"age": "29"

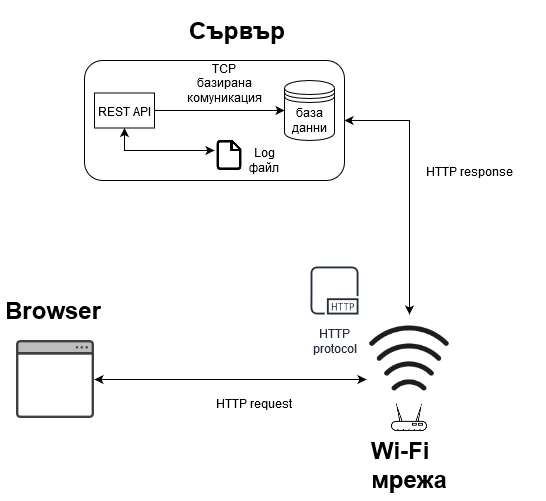
}

**2.2. Модул за Администрация**

**Принцип на работа**

Модула е реализиран под формата на website обслужван от сървър на определен адрес и порт. За да може потребителя да достъпи website-а, те трябва да се в една мрежа. Website-а дава възможност за администриране на идентификаторите и точките на достъп. Под точка на достъп се разбира четец, но от потребителска гледна точка този термин е по - подходящ. Комуникацията се извърша на базата на HTTPS протокола.   
При първоначално отваряне на website-а, потребителя ще бъде пренасочен към страница за автентикация, където ще трябва да въведе потребителско име и парола, след успешна автентикация, потребителя е пренасочен към началната страница на website-а.

**Блокова схема**

****

Фигура 4. Блокова схема на комуникация между browser и сървър

**Автентикация**

За да бъде предпазен website-а от не желана външна намеса са взети няколко мерки. Пренасочване на не-автентикирани потребители към страницата за въвеждане на потребителско име и парола, и запазване на нужната информация за автентикиране при последващи заявки към сървъра при клиента. Това е постигнато чрез JSON Web Token (JWT) [16] стандарта.

JWT предоставя лесен и компактен подход за автентикация, чрез обмен на съобщения под формата на JSON. Информацията, която се пренася винаги се подписва с таен ключ, който е познат само на сървъра генерирал token-а. Чрез подписване сървъра може да провери при следваща заявка дали token-а бил подписан със същия ключ и съответно да предприеме нужното действие ако не е. Това ни предпазва от възможността, външен потребител да изфабрикува фалшив token, тъй като той не знае ключа с който трябва да бъде подписан за да се валидира от сървъра. Подписването се извърша чрез HMAC алгоритъм [17].  
Стандарта се състой от три части: Header, Payload и Signature.  
**Header** – обикновено съдържа информация за типа на token-а и вида на алгоритъма използва при подписването му. Например:  
{ "alg": "HS256", "typ": "JWT" }

**Payload** – съдържа информация свързана с потребителя като уникален номер в базата, роля (например: администратор). Например:  
{ "id": "126", "role": “admin” }

**Signature** – Header-а и Payload-а се кодират чрез Base64 [18] алгоритъм и се криптират с посочения алгоритъм в Header-а и тайния ключ и получения резултат представлява Signature-а.

Накрая се получава текстов формат, който представлява самия token. Например: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6InRlc3RAdGVzdC5jb20iLCJyb2xlIjoiQWRtaW4iLCJuYmYiOjE1Nzg4NDcwMzQsImV4cCI6MTU3ODg0NzIxNCwiaWF0IjoxNTc4ODQ3MDM0fQ.zk3wJXnPm6RL-yQ17xJ7vHNwI0wvtxBD7hQBXE0e-JE

Интегрирането на token-а в HTTP протокола става чрез authorization header. След като потребителя въведе валидни потребителско име и парола в response съобщението се връща и token, който се запазва с browser-а докато не бъде затворен. В header-те на всяка следваща заявка към сървъра се включва token-а, по този начин на потребителя няма да му се налага да въвежда потребителско име и парола след всяко действие.  
С цел допълнителна сигурност token-а се генерира с определена продължителност на живота. След изтичането на този период, token-а става невалиден и потребителя бива пренасочен към страницата за въвеждане на потребител и парола, при което му се генерира нов token.

**Потребителска парола**

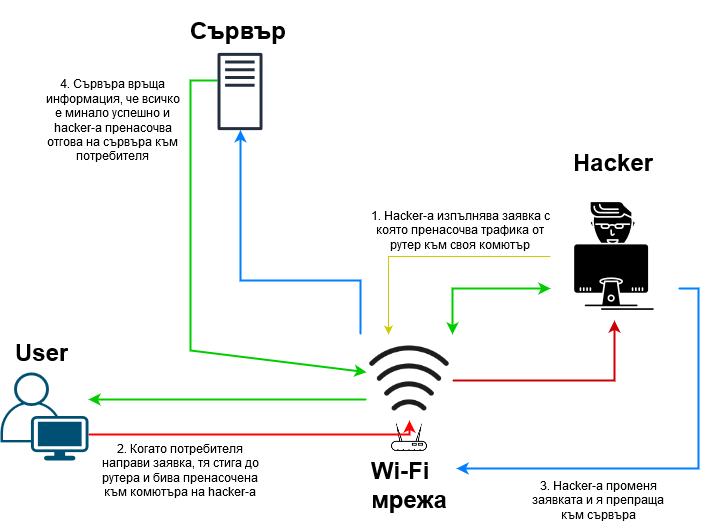
При работа с потребители, е важно паролата да не бъде запазвана в чист вид в базата данни от гледна точка на сигурност. При регистриране върху паролата на потребителя се прилага подход наречен hashing.   
Hashing има за цел да видоизмени стойността по такъв начин, че да е невъзможно да бъде възстановена оригиналната стойност. Един от най – популярните алгоритми за hashing е SHA (Secure Hash Algorithms).

По – този начин ако базата данни или данните в нея попаднат в грешни ръце, чувствителната информация като пароли ще бъде неизползваема.

Тъй като във времето е имало не малко на брой откраднати данни от не защитени сайтове също така има и публично достъпни данни за hash-те стойности на най – често използваните пароли, е възможно злонамерен потребител да опита да налучка паролата на даден потребител чрез атака наречена pre-computed hash attack (атака с предварително генериран hash).   
Това е сериозен проблем за сигурността, защото потребителите често използват една и съща парола за всичките си регистрации и е достатъчно да се регистрирате в един не защитен сайт, при което всичките ви останали регистрации също биват застрашени от злоупотреба.  
За да се избегне това в днешно време hashing алгоритмите използват допълнителна произволна стойност (salt) при генерирането на hash-а, която се добавя към паролата на потребителя.

**2.3. Софтуерна сигурност**

За да се постигне цялостна сигурност на комуникация се използва HTTPS комуникация между ардуиното и сървър, както и между web browser-а и сървъра (в контекста на модула за администрация).

Един от недостатъците на Wi-Fi мрежите, е че са много по – лесно достъпни от колкото LAN мрежата, където е нужна физическа връзка чрез кабел. Това отваря възможност за така наречената middle-man-attack. Тя се изразява във възможността на даден потребител да подслушва или манипулира трафика, който се предава през рутер или друго комуникационно устройство, чрез пренасочване на този трафик до сървър / компютър, до което той има пълен контрол. Това става проблем с масовото навлизането на Wi-Fi мрежите в последното десетилетие.

Фигура 5.  
Блокова схема  
на принцип на  
middle-man-attack

Частично проблема може да се реши като се защити създадената мрежа от рутера чрез парола, което би попречило на външен потребител, който не знае паролата, но проблема остава наличен за потребители, които вече знаят паролата или когато мрежата към която са свързани е публично достъпна. Една от основните задачи на HTTPS е да реши този проблем.

HTTPS има за цел да осигури защитена комуникация между клиент-сървър, чрез криптиране на трафика. HTTPS стъпва върху Transport Later Security (TLS). За да се установи TLS връзка, клиента и сървъра изпращат редица съобщения с които се договарят за това какъв алгоритъм за криптиране ще се използва и се установява автентичността на сертификата представен от сървъра.

Първоначално клиента изпраща заявка до сървъра, в която оказва, че иска да използва TLS комуникация. Във заявка клиента е оказал различните версии на протокола, hash функции и видове алгоритми за криптиране, които поддържа. Обикновено това пряко зависи от версията на операционната система и версията на клиентското приложение, което се използва за комуникация (пример: Firefox v60.0.1). След като съобщението достигне до сървъра, според неговите възможности, избира възможно най – сигурните версия на протокол, hashing и криптиращ алгоритъм, като това отново до голяма степен зависи от версията на операционна система на сървъра и сертификата, който е приложен към него и с ново съобщение оказва на клиента какви алгоритми ще се използват за комуникацията.  
След това сървъра се идентифицира пред клиента като изпраща асиметричен публичен цифров сертификат (digital certificate) предназначен за криптиране на web трафик.

Асиметричния сертификата представлява файл, който съдържа името на сървъра, до кога е валиден, сертифициращ орган (certificate authority), публичен ключ (pubic key), таен ключ (private key) и какви алгоритми поддържа. При асиметричните алгоритми за криптиране, публичния ключ се ползва за криптиране, а тайния ключ се ползва за декриптиране. Този вид алгоритми се води еднопосочен, тъй като тайния ключ е известен само на притежателя на сертификата, а на всички останали се предоставя така наречения публичен сертификат (public certificate), който не съдържа в себе си тайния ключ. По този начин всички, които имат публичния сертификат могат да криптират данните и да ги изпращат до сървъра, но само сървъра има нужния ключ за да декриптира съобщенията.

Клиента проверява достоверността на публичния сертификат чрез така наречения сертифициращ орган (certificate authority). Всяка операционна система има предварително инсталирани сертификати наречени корен на сертификата (certificate root). За да се довери клиента на този публичен сертификат трябва операционната система да има инсталиран certificate root за конкретния сертифициращ орган. Едни от най – разпространените сертифициращи органи са VeriSign и DigiCert. Операционните системи (Windows, Linux, Mac OS) имат инсталирани certificate root от тези компании и когато сървъра върне публичен сертификат със сертифициращ орган някоя от тези компании, той автоматично му се доверява, като това се случва на ниво операционна система или софтуерно приложение.

Следа размяна на ключове, които ще се използват за криптирането на трафика. Клиента генерира ключ под формата на произволен низ от символи (нарича се session key). Ключа бива криптиран с публичния ключ на сертификата и го изпраща до сървъра. На този етап единствено клиента и сървъра знаят не-криптираната стойност на този низ, тъй като клиента е този който го е генерирал, а след криптирането му и изпращането му до сървъра само сървъра знае как да го декриптира. Дори и да има потребител, който да следи трафика, той не може да декриптира ключа тъй като само сървъра разполага с тайния ключ (private key) за декриптиране.

От този момент всяко съобщение изпратено от клиента или сървъра се криптира с ключа генериран от клиента (session key) в рамките на един връзка докато е отворена. При затваряне на връзката процеса се изпълнява на ново.

За да окаже клиента на сървъра, че иска да използва защитена връзка, обикновено това става чрез промяна на порта към който клиента иска да се свърже, като за HTTPS това е 443. По отношение на HTTP, това става чрез промяна на URI схемата от http:// на https://.  
В днешно време е честа практика, при опит за свързване със сървър по http, който е настроен да работи по https, той автоматично да пренасочи комуникацията с цел осигуряване на защита, тъй като това не коства нищо на потребителя.

**2.4. Лог (Log)**

Във всяка софтуерна система в различните и етапи на разработка, могат да се появят грешки по време на нейното функциониране. За да не се губи ценна информация всяка грешка се записва в така наречения log. В него се съдържа информация за грешката и състоянието на системата по време на нейното възникване, както и друга информация, която би показала причината заради, която е възникнала. Най – често под формата файл.