**3. Инженерно решение на поставената задача**

**3.1. Четец**

Четеца е изграден на базата на Arduino Uno REV3 модел [1], MFRC522 RFID модул [2], пасивен RFID идентификатор MIFARE Classic 1k [3] под формата на 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.

**Arduino Uno REV3**

Фигура 8. Снимка на Arduino Uno REV3

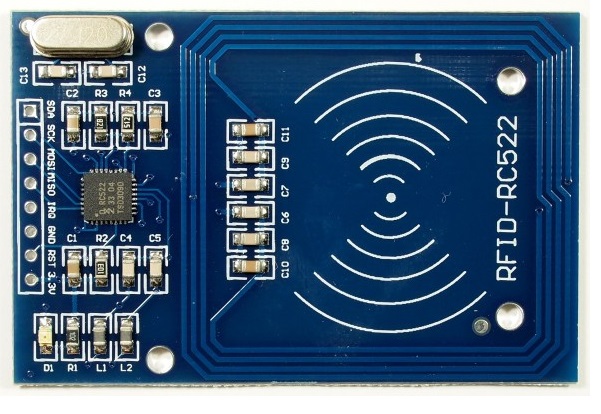


Едноплатков микро-контролер базиран на ATmega328P. Може да бъде захранено чрез USB връзка или от външен източник чрез адаптер преобразуващ променлив към постоянен ток. Широко разпространен заради големя брой щифтове и разумна цена. Предварително зареден с bootloader, който позволява зареждане на нов програмен код без да се налага използването на програматор. Езиците за програмиране са C/C++. [1]

Таблица 3. Технически параметри на Arduino Uno REV3

|  |  |
| --- | --- |
| Микро-контролер | ATmega328P |
| Работно напрежение | Работно напрежение 5V |
| Входно напрежение | 6-20V |
| Цифрови щифтове | 14 (от които 6 позволяват широчинно-импулсна модулация) |
| Щифтове с Широчинно-импулсна модулация | 6 |
| Аналогови щифтове | 6 |
| Постоянен ток на щифтовете | 20 mA |
| Максимален толериран постоянен ток на входно-изходните щифтовете | 40 mA |
| Постоянен ток на 3.3V щифт | 50 mA |
| Флаш памет | 32 KB (ATmega328P) от които 0.5 KB използвани от bootloader-а |
| Оперативна памет (SRAM) | 2 KB (ATmega328P) |
| EEPROM | 1 KB (ATmega328P) |
| CPU работна честота | 16 MHz |
| Вградени LED индикатори | 13 |
| Дължина | 68.6 mm |
| Широчина | 53.4 mm |
| Тегло | 25 g |
| Цена | 20 € |

**MFRC522 RFID**  
  
Фигура 9. Снимка на MRFC522 RFID четец



Интегрална схема предназначена за безконтактна комуникация на честота от 13.56 MHz. Поддържа стандартите ISO/IEC 14443 A/MIFARE и NTAG. [2]

Таблица 4. Технически параметри на MRFC522

|  |  |
| --- | --- |
| Работен ток | 13 – 26 mA |
| Работно напрежение | 3.3 V |
| Работна честота | 13.56 MHz |
| Разстояние за прочитане | 0 ~ 35 mm |
| Дължина | 60 mm |
| Широчина | 40 mm |
| Цена | ~ 5 € |

**MIFARE Classic 1k идентификатор**



Фигура 10. Снимка на MIFARE 1K Classic ключодържател

Семейство интегрални схеми за безконтактна комуникация по стандарт ISO/IEC 14443 A. Малките размери позволяват вграждането в пластмасови карти и ключодържатели. [3]  
В текущата дипломна работа е използвана реализация под формата на ключодържател.

**ESP8266 Wi-Fi**

Напълно завършен и самостоятелен Wi-Fi продукт с интегрирана поддръжка за TCP/IP протокола. Модула е способен самостоятелно да обслужва софтуерен продукт или да се използва като допълнение към друго устройство. Модула се разпространява с firmware, който поддържа набор от инструкции за управление наречени AT Commands. [4]

Таблица 5. Технически параметри на ESP8266

|  |  |
| --- | --- |
| Поддържани стандарти | 802.11 b/g/n, Wi-Fi Direct, Soft-AP |
| Работно напрежение | 3.3 V – 3.6 V |
| АT Commands | Да, част от firmware |
| TCP/IP протокол | Да, интегриран |
| Цена | ~ 5 € |

**ESP-01 адаптер с регулатор на напрежение**

Фигура 11. Снимка на ESP-01 ESP8266 адаптер



Адаптер за ESP8266 Wi-Fi модула използван като регулатор на входното напрежение до 3.3 V. [19]  
Макар и ардуиното да предлага 3.3 V изходно напрежение, не винаги достига до тази стойност и тъй като ESP8266 е чувствителен на подобни падове, не винаги работи с пълни възможности. За да използва 5 V напрежение като захранване е нужен регулатор на напрежение, тъй като толеранса на ESP8266 е до 3.6 V.

Таблица 6. Технически параметри на ESP-01 адаптер

|  |  |
| --- | --- |
| Входно напрежение | 5 V – 12 V |
| Изходно напрежение | 3.3 V |
| Цена | ~ 2 € |

**Индикация за пропуск**



Фигура 12. Снимка на LED

Използва се LED за да се окаже на потребителя дали достъпа е разрешен. Зелен диод за да се окаже успешен пропуск и червен цвят да се окаже забранен достъп.

Таблица 7. Технически параметри на LED

|  |  |
| --- | --- |
| Пад на напрежение – зелен цвят | 1.9 V – 4 V |
| Пад на напрежение – червен цвят | 1.6 V – 2 V |
| Работен ток | 20 mA |

**Интеграция на MFRC522 RFID модул**

Използва се библиотека с отворен код [20], която предоставя лесен начин за работа с четеца. Комуникацията между микро-контролера и четеца се осъществява чрез Serial Peripheral Interface (SPI) [2] протокола.  
Библиотеката има определени очаквания към разположението на щифтовете за да работи коректно.

Таблица 8. Разположение на щифтове при MRFC522

|  |  |  |
| --- | --- | --- |
| **Сигнал** | **Четец** | **Ардуино** |
| RST/Reset | RST | Digital 9 |
| SPI SS | SDA | Digital 10 |
| SPI MOSI | MOSI | Digital 11 |
| SPI MISO | MISO | Digital 12 |
| SPI SCK | SCK | Digital 13 |
| GND | GND | GND |
| VCC | 3.3 V | 3.3 V |

**Интеграция на ESP8266 Wi-Fi модул**

Модула и притежава собствена изчислителна мощ (32-bit micro-CPU) и се разпространява с firmware зареден с AT Commands. [4]  
Комуникацията между модула и ардуиното става чрез серийна комуникация на базата на universal asynchronous receiver-transmitter (UART). Тъй като ардуиното има само два серийни щифта: 0 – RX, 1 – TX, които се използват за връзка с компютър по – време на разработка и тестване на кода, се използва библиотека SoftwareSerial, която дава възможност за серийна комуникация чрез използване на другите цифрови щифтове. Библиотеката е част от ядрото на ардуиното, което представлява група библиотеки, които се разпространяват с него. [21]

Модула е свързан към ESP-01 адаптера, който от своя страна е свързан към ардуиното.

Таблица 9. Разположение на щифтовете при ESP8266

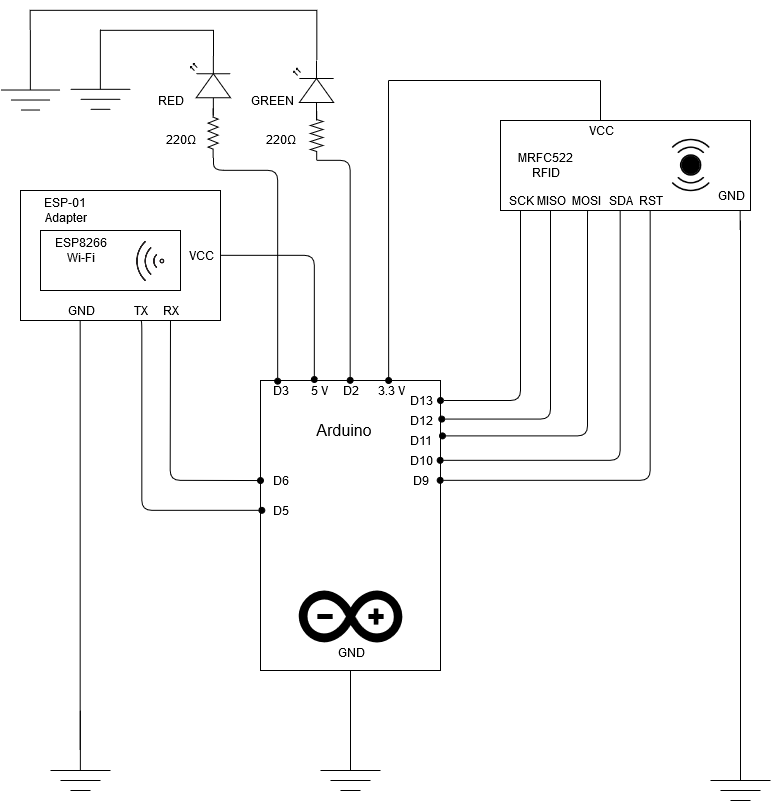
|  |  |  |
| --- | --- | --- |
| **ESP8266** | **ESP-01 адаптер** | **Ардуино** |
| RX | RX | Digital 6 (TX) |
| TX | TX | Digital 5 (RX) |
| GND | GND | GND |
| 3.3 V | VCC | 5 V |
| EN | VCC | 5 V |

**Принцип на работа**

Bootloader-а на ардуиното ни позволява да подменяме изпълняващия се код без да се налага да използваме директно програматор. Има две основни функции, които ни предоставя bootloader-а за да работим с ардуиното, това са setup и loop функциите.   
setup функцията се изпълнява само веднъж – при първоначалното захранване на ардуиното. Тук се поставя инициализиращ код.  
loop функцията се изпълнява периодично, чрез нея имаме възможност да работим с текущото състояние на ардуиното през много кратък период, което позволява да реагираме на промени при различните щифтове и външните модули. Тук се поставя код, който следи за състоянието на съответните щифтове и външни устройства и взема съответните решения какво следва да се изпълни.  
Скрипта зареден в ардуиното първоначално зарежда SoftwareSerial библиотеката за работа със серийна комуникация, след това се зарежда SPI библиотека за работа с SPI протокола и накрая се зарежда библиотека за работа с MFRC522 модула.

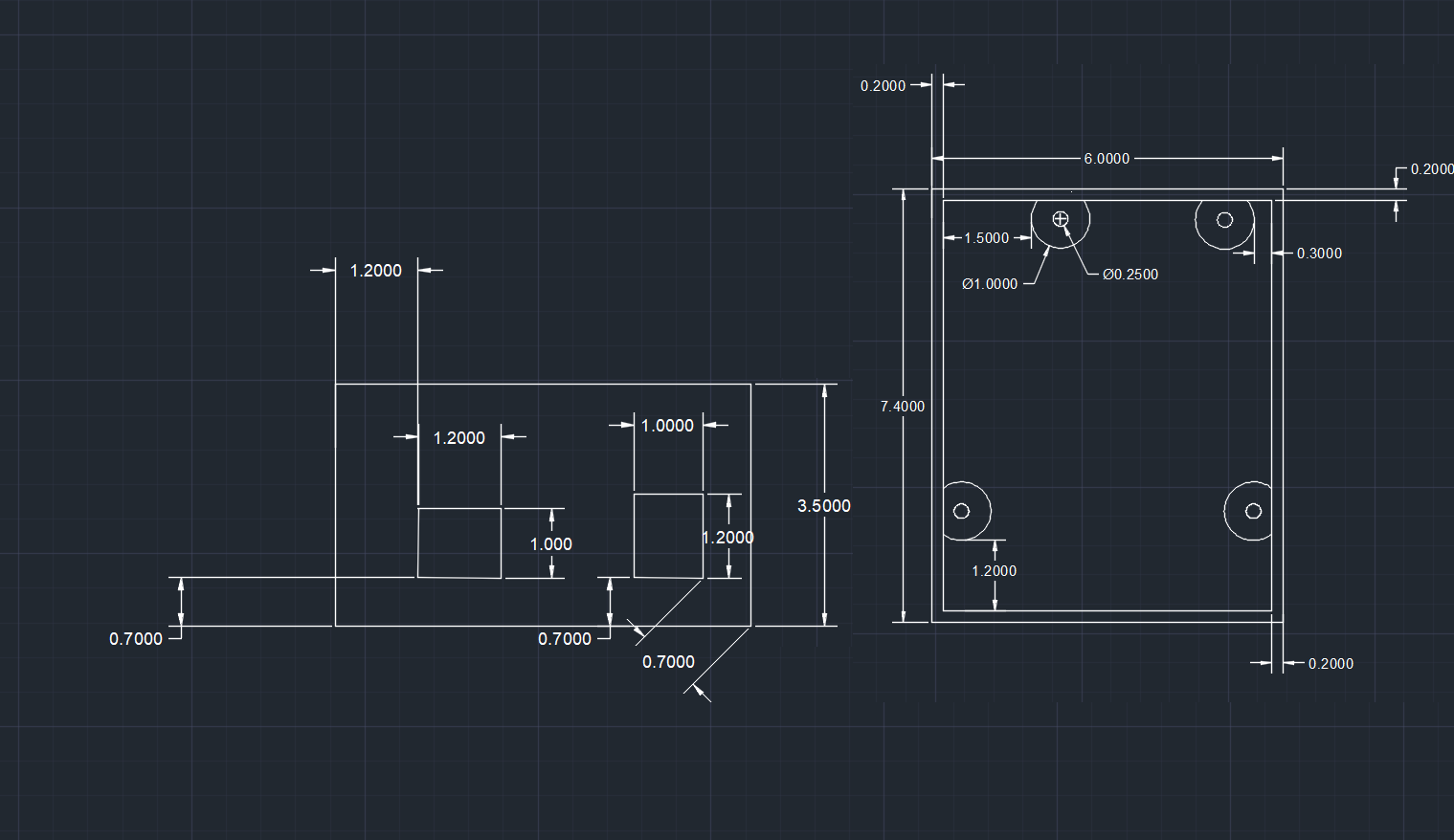
В setup функцията се инициализира променлива, която съдържа уникалния номер на ардуиното. В случая това е променлива, която се избира от програмиста, тъй като ардуино платформата не дава възможност за достъпа до серийна номер на устройството.   
Цифрови щифтове 2 и 3 се инициализират като изходни, с цел захранване на зеления и червения LED.  
SoftwareSerial библиотеката се настройва на цифрови щифтове 5 и 6 от ардуиното за серийна комуникация с Wi-Fi модула. Където 5-ти се използва за приемане, а 6-ти се ползва за изпращане на данни – огледално на тези на Wi-Fi модула. По – този начин изходните данни от модула ще се третират като входни за ардуиното, а изходните от ардуиното ще се третират като входни за модула.  
Инициализира се URI адреса и порта на сървъра, които са част от скрипта, който се изпълнява. Тъй като те трябва да се знаят предварително, за текущата дипломна работа сървъра е конфигуриран на адрес 192.168.0.105 и порт 443. Изпраща се AT команда до Wi-Fi модула да създаде WLAN мрежа с име “arduino-master” и парола “test1234”, към която четеца се свързва с IP 192.168.0.100. Накрая на setup функцията червения и зеления LED се захранват в продължение на секунда за да се индикира, че инициализацията е преминала успешно.  
В loop функцията на всяка итерация се проверява дали четеца е успял да прочете данните на идентификатор в близост. При успешно засечен идентификатор, неговия уникален номер бива прочетен. Това става чрез използването на MFRC522 библиотеката. След това изпраща AT команда за отваряне на HTTPS връзка към сървъра и се изпраща заявка до “/accesscontrol/api/tags/checkaccess?TagNumber={tagNumber}&AccessPointSerialNumber={accessPointSerialNumber}”  
където {tagNumber} е уникалния номер на идентификатора, а {accessPointSerialNumber} е уникалния номер на четеца.  
Резултата от всяка AT команда връща съответно OK за успешна операция и ERROR при грешка, а при HTTP/HTTPS заявки се връща информация и за status code-а на response съобщението. Следва проверка за успешна операция и HTTPS заявка с status code 200, при което се захранва зеления LED за една секунда, а при не успешна проверка или не достатъчно ниво на достъп се захранва червения LED за една секунда.

**Схема на свързване**

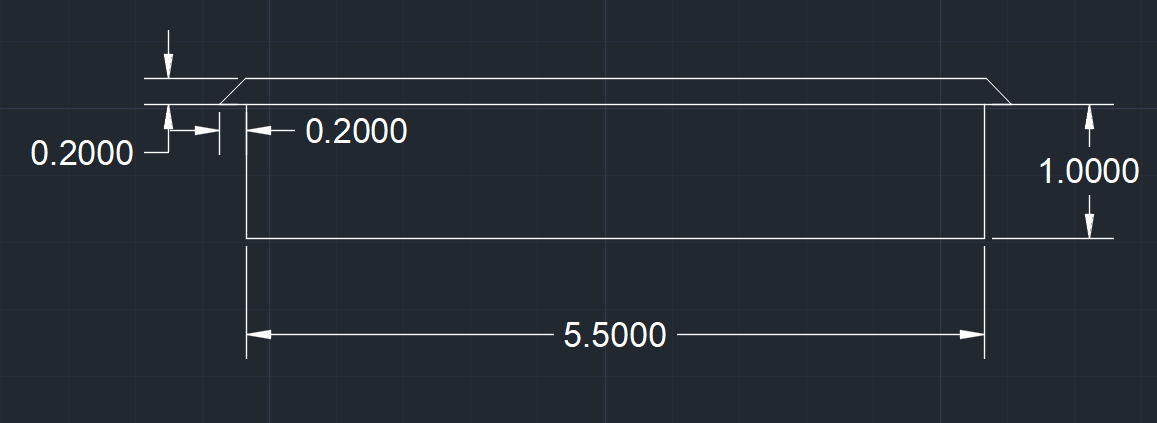


**Кутия**

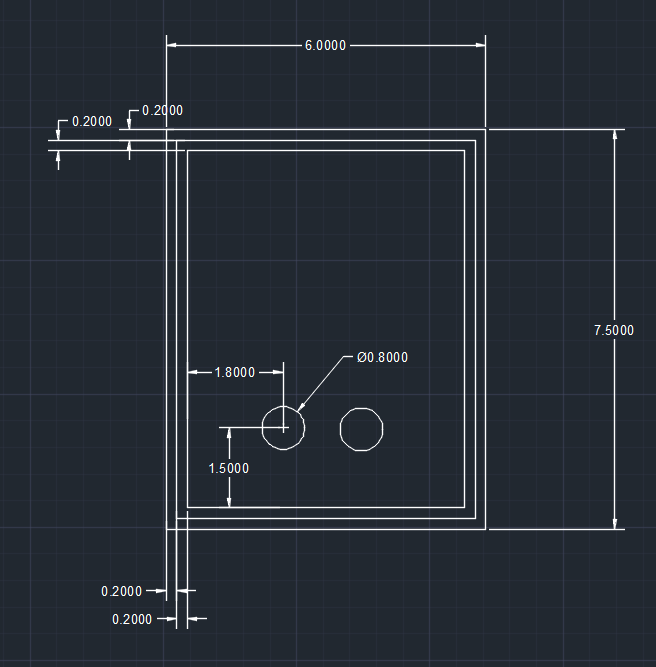
2D чертежите са направени на Autodesk AutoCAD LT 2020



Фигура 3.5. 2D чертеж на кутията от пред (вляво) и отгоре (в дясно)

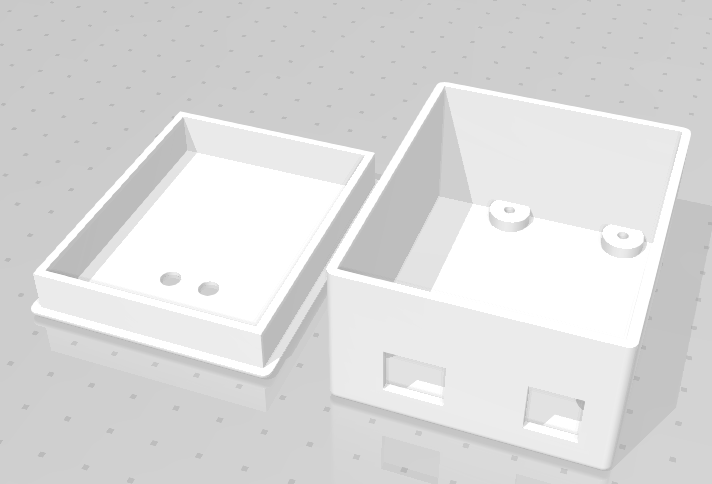


Фигура 3.6. 2D чертеж на капака на кутията гледан от пред

****

Фигура 3.7. 2D чертеж на капака на кутията гледан от горе

3D модела е направен на Autodesk 1234 Design



3.8. Фигура на 3D модел на кутията

**Финален продукт**Направена е с 3D принтер от 3D модела в предишната фигура



Фигура 3.9. Кутия принтирана от 3D принтер

**3.2. REST API**

Основната идея на API-то е да делегира работата с базата и да подготвя данните от базата в подходящ вид. Използва се както от Website-а за да се извършват всички административни дейности, както и от Ардуиното за да се извършва проверката за достъп.  
Имплементирано е на C# 7.3 под .NET CORE 2.1. Използван е ASP NET Core framework-а, която е базирана на Model-View-Controller модела.  
Архитектурно проекта е разбит на пет части: контрол на достъп (AccessControl), администрация (Administration), автентикация (Auth), логове (Log), статистики (Stat).

**AccessControl**  
Отговаря за проверката за контрол на достъп. Тази част се състой само от един контролер със следните функционалности:

|  |
| --- |
| **uri**: /accessControl/api/tags/checkAccess **query**: tagNumber=&accessPointSerialNumber= **HTTP method**: GET **HTTP Status Code:  200 –** когато достъпа е разрешен **401 –** когато достъпа е отказан **404** – когато точката на достъп или четеца не съществуват в базата |

Сравнява нивото на достъп на идентификатора дали е достатъчно за да бъде пропуснат през определена точка на достъп (четец). Това е URI-то към което се обръща ардуиното за да получи информация за контрола на достъп. Всяко изпълнение на тази заявка в базата се генерира събитие, което запазва пълната информация за идентификатора и точката на достъп – впоследствие тези събития се използват за генерирането на справки.

**Administration**Отговаря за всички заявки свързани със сайта за администрация. Кода е разбит в четири контролера: AccessPointController, ExportController, TagsController, UsersController.

*AccessPointController* – дава възможност за работа с точките на достъп (четци)

|  |
| --- |
| **uri**: /administration/api/accessPoint/register **HTTP method**: POST **Request Content-Type**: application/json  **Request Body**: { serialNumber, description, isActive, accessLevel, direction }  **HTTP Status Code:  200 –** когато точка за достъп е добавена в базата **400 –** когато изпратения модел е невалиден |

Регистрира нова точка на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/activate **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато точката за достъп е активирана **404 –** когато точката за достъп не съществува |

Активира неактивна точка на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/deActivate **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато точката за достъп е деактивирана **404 –** когато точката за достъп не съществува |

Деактивира активна точка за достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/accessLevel  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { accessPointId, accessLevel }  **HTTP Status Code:  200 –** когато нивото на достъп е сменено **404 –** когато точката за достъп не съществува |

Променя нивото на достъп за дадена точка на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/update  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id, description, isActive, isDeleted, accessLevel, direction}  **HTTP Status Code:  200 –** когато точката на достъп е променена успешно **404 –** когато точката за достъп не съществува |

Променя оказаните свойства на дадена точка на достъп. Ако някое от свойствата в заявката няма стойност се пренебрегва по време на update-a.

|  |
| --- |
| **uri**: /administration/api/accessPoint/delete  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато точката на достъп е изтрита **404 –** когато точката за достъп не съществува |

Изтрива дадена точка за достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/undelete **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато точката на достъп е възстановена **404 –** когато точката за достъп не съществува |

Възстановява изтрита точка на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/active  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за активните точки на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/inactive  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за всички неактивни точки на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/deleted  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за всички изтрити точки на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/unknown  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за всички непознати точки на достъп

|  |
| --- |
| **uri**: /administration/api/accessPoint/count  **HTTP method**: GET **Response Body**: [{ active, inActive, deleted, unknown }] |

Връща информация за бройките на различните състояния на точките за достъп.

*ExportController* **–** дава възможност за експорт на данните.

|  |
| --- |
| **uri**: /administration/api/export  **HTTP method**: GET **HTTP Response Content**: application/json |

Връща пълна информация за потребителите, точките на достъп, идентификаторите и събитията.

*UsersController* **–** дава възможност за работа администраторски потребители

|  |
| --- |
| **uri**: /administration/api/users/register  **HTTP method**: POST **Request Content-Type**: application/json  **Request Body**: { email, password, roles }  **HTTP Status Code:  200 –** когато потребителя е регистриран успешно **400 –** когато данните в тялото на заявката са невалидни |

Регистрира администраторски потребител.

*TagsController* – дава възможност за работа с идентификаторите.

|  |
| --- |
| **uri**: /administration/api/tags/register  **HTTP method**: POST **Request Content-Type**: application/json  **Request Body**: { number, accessLevel, userName }  **HTTP Status Code:  200 –** когато идентификатора е регистриран успешно **400 –** когато данните в тялото на заявката са невалидни |

Регистрира нов идентификатор по номер, ниво на достъп и потребител на когото принадлежи.

|  |
| --- |
| **uri**: /administration/api/tags/activate  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато идентификатора е активиран успешно **404 –** когато идентификатора не съществува |

Активира неактивен идентификатор.

|  |
| --- |
| **uri**: /administration/api/tags/deactivate  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато идентификатора е деактивиран успешно **404 –** когато идентификатора не съществува |

Деактивира активен идентификатор.

|  |
| --- |
| **uri**: /administration/api/tags/delete  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато идентификатора е изтрит успешно **404 –** когато идентификатора не съществува |

Изтрива идентификатор.

|  |
| --- |
| **uri**: /administration/api/tags/undelete **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато идентификатора е възстановен успешно **404 –** когато идентификатора не съществува |

Възстановява изтрит идентификатор.

|  |
| --- |
| **uri**: /administration/api/tags/accessLevel  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id }  **HTTP Status Code:  200 –** когато нивото на достъп на идентификатора е променено успешно **404 –** когато идентификатора не съществува |

Променя нивото на достъп на даден идентификатор

|  |
| --- |
| **uri**: /administration/api/tags/update  **HTTP method**: PATCH **Request Content-Type**: application/json  **Request Body**: { id, isActive, isDeleted, username, accessLevel }  **HTTP Status Code:  200 –** когато свойствата на идентификатора са променени успешно **404 –** когато идентификатора не съществува |

Променя свойствата на идентификатора с оказаните в тялото на заявката. Ако някое от свойствата в заявката няма стойност се пренебрегва по време на update-а.

|  |
| --- |
| **uri**: /administration/api/tags/active  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за активните идентификатори.

|  |
| --- |
| **uri**: /administration/api/accessPoint/inactive  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за неактивните идентификатори.

|  |
| --- |
| **uri**: /administration/api/accessPoint/unknown  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за непознатите идентификатори.

|  |
| --- |
| **uri**: /administration/api/accessPoint/deleted  **query:** page=&pageSize=  **HTTP method**: GET **Response Body**: [{ page, pageSize, pageCount, count, items }] |

Връща страницирана информация за изтритите идентификатори

|  |
| --- |
| **uri**: /administration/api/accessPoint/count  **HTTP method**: GET **Response Body**: { active, inactive, deleted, unknown } |

Връща информация за бройките на идентификаторите в различните състояния

|  |
| --- |
| **uri**: /administration/api/accessPoint/users  **HTTP method**: GET **Response Body**: [{ id, userName }] |

Връща информация за всички потребители, които притежават идентификатори.

**Auth**Отговаря за всички функционалности свързани с автентикацията на потребителите.

|  |
| --- |
| **uri**: /auth/api/token/generate **HTTP method**: POST **Request Content-Type**: application/json  **Request Body**: { email, password }  **HTTP Status Code:  200 –** JWT е генериран **404 –** когато потребителя във тялото на заявката не съществува |

Генерира JWT по даден емайл адрес и парола на потребител.

**Log**Отговаря за всички функционалности свързани със записването на грешки възникнали при клиента.

|  |
| --- |
| **uri**: /log/api/client/error **HTTP method**: POST **Request Content-Type**: application/json  **Request Body**: { message }  **HTTP Status Code:  200 –** когато грешката е записана успешно |

Записва информация за възникнали грешни при клиента.

**Stat**Отговаря за функционалности свързани с генерирането на справки

|  |
| --- |
| **uri**: /stat/api/user/overview **HTTP method**: GET **Response Body:** {avgEntranceTime, avgExiteTime, avgWorkHourNorm} |

Връща информация средностатистически когато потребителите идват на работа, тръгват от работа и средно с колко се отмества прекараното време на работното място спрямо очаквания осем часовия работен ден.

|  |
| --- |
| **uri**: /stat/api/user/all **HTTP method**: GET **Response Body:** [{ id, username, avgEntranceTime, avgExitTime, avgWorkHourNorm }] |

Връща информация за всеки един потребители - емайл адрес, средностатистически кога идва на работа, средностатистически кога си тръгва от работа и с колко се отмества времето прекарано на работното място спрямо осен часовия работен ден.

|  |
| --- |
| **uri**: /stat/api/user/norm query: userId=&startDate=&endDate= **HTTP method**: GET **Response Body:** [{day, time}] |

Връща информация за средностатистическа информация за отместването на изработените часове спрямо осем часове работен ден за даден потребител за опреден период от време. Информацията е на дневна база.

|  |
| --- |
| **uri**: /stat/api/user/entrance query: userId=&startDate=&endDate= **HTTP method**: GET **Response Body:** [{day, time}] |

Връща информация за средностатистическа информация за това, когато за даден потребител идва на работа. Информацията е на дневна база.

|  |
| --- |
| **uri**: /stat/api/user/exit query: userId=&startDate=&endDate= **HTTP method**: GET **Response Body:** [{day, time}] |

Връща информация за средностатистическа информация за това, когато за даден потребител си тръгва от работа. Информацията е на дневна база.

**3.3. Website**

Изграден е на базата на Angular framework-а. Основната градивна единица е така наречения компонент. Той представлява съвкупност TypeScript файл за дефиниране на логиката, HTML за дефиниране на визуализационната структура и CSS файл за дефиниране на дизайна на резултатния компонент.

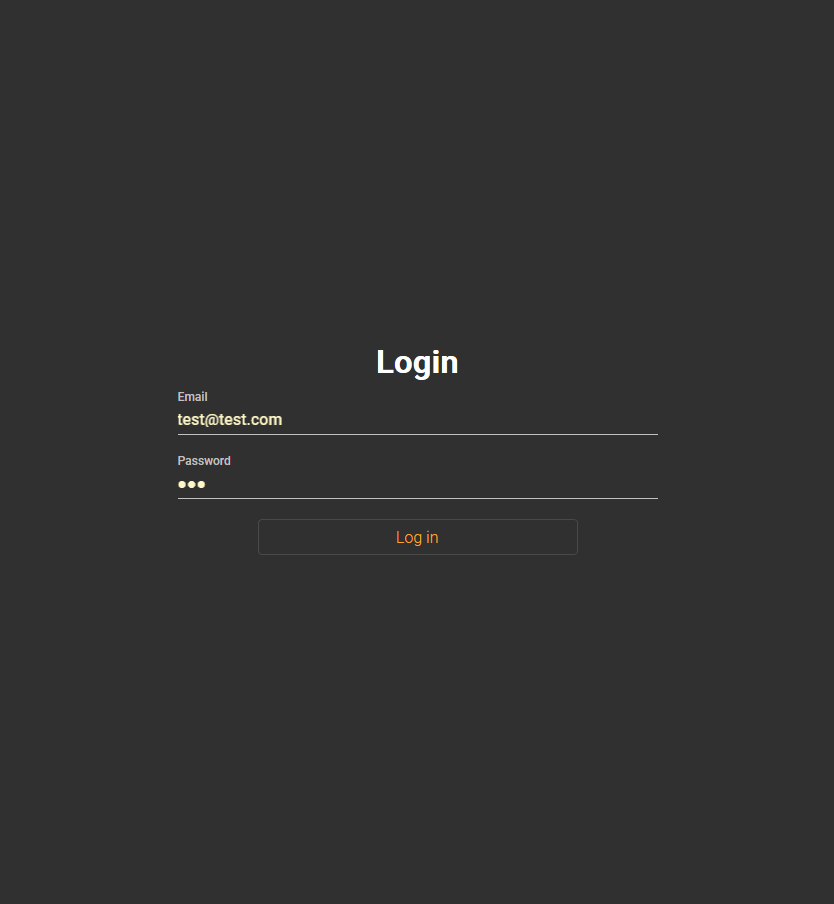
Компонентно базираните framework-ци позволяват по –добро разделение на кода, съответно по – добра изолация, което неминуемо води до по – лесната му поддръжка.

За дизайна е използвана Materialize библиотеката, която дефинира предварително готови компоненти, които се подчиняват на общи дизайна правила. Под компоненти се разбира: бутони, таблици списъци.

Website-а не е публично достъп, но за целта на дипломната работа website-а може да се отвори само от други компютри в същата мрежа в която се намира и сървъра на адрес: <https://www.desktop-sb3j0h0/>.

За самата демонстрация е създаден и администраторски потребител със следните данни: **Емайл**: [test@test.com](mailto:test@test.com), **Парола**: 123.

При първоначално отваряне на сайта в browser ще ви пренасочи към страница за автентикация, чрез въвеждане на потребителско име и парола.



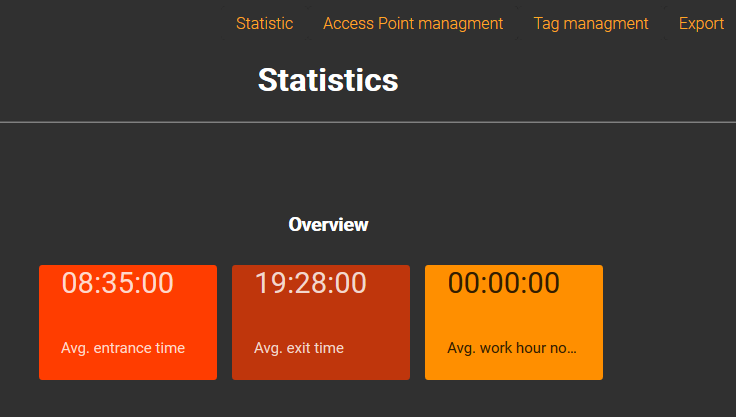
След успешна автентикация, приложението ви пренасочва към началната страница. Генерирания JWT е с валидност от три минути, в рамките на които администратора няма ще бъде третиран като автентикиран при всяко негово действие. След изтичането на тези три минути, системата ще ви пренасочи отново към страницата за въвеждане на потребителско име и парола.

Всяка страница съдържа в горен десен ъгъл навигационно меню. Сайта е разбита на четири основни страници: Statistic, Access Point management, tag management, Export

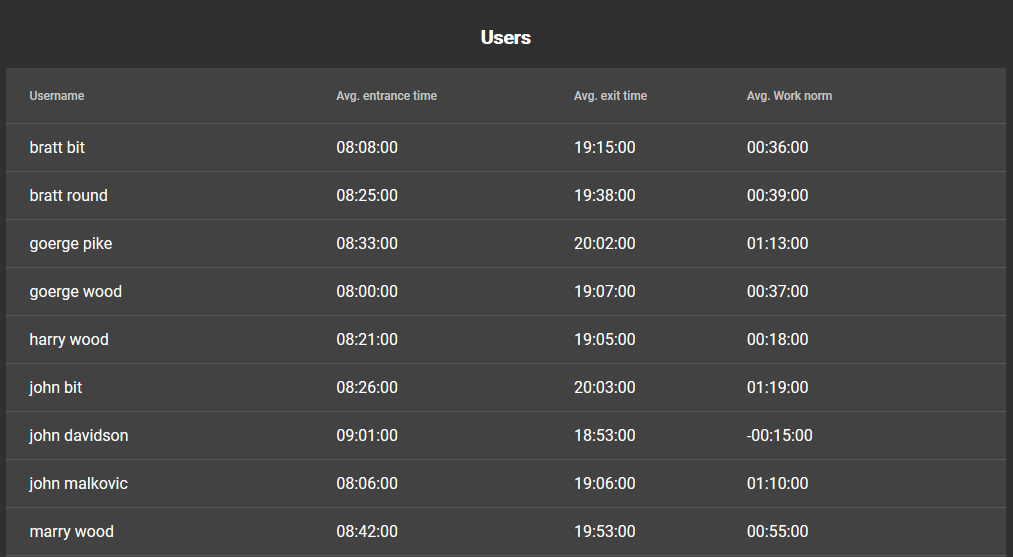


**Statistic**Показва справки свързани с потребителите свързани с това кога идват на работа, кога си тръгват от работа, по колко време прекарват на работа. Съдържа три основни компонента: Overview, Users, Charts

Overview – визуализира информация за средностатистически за всички потребители, кога потребителите идват на работа, кога тръгват и дали осем часовата норма за работния ден е изпълнена.

****

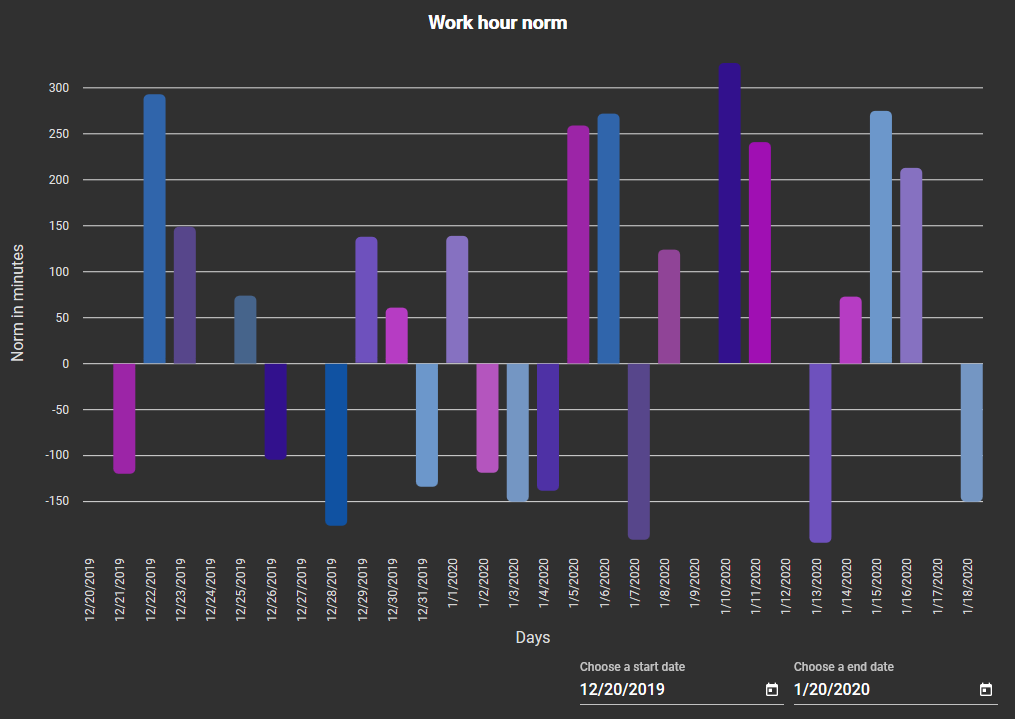
Users **–** визуализира за всички потребители средностатистически кога идва на работа, кога си тръгва от работа и по колко време прекарва, под формата на списък.

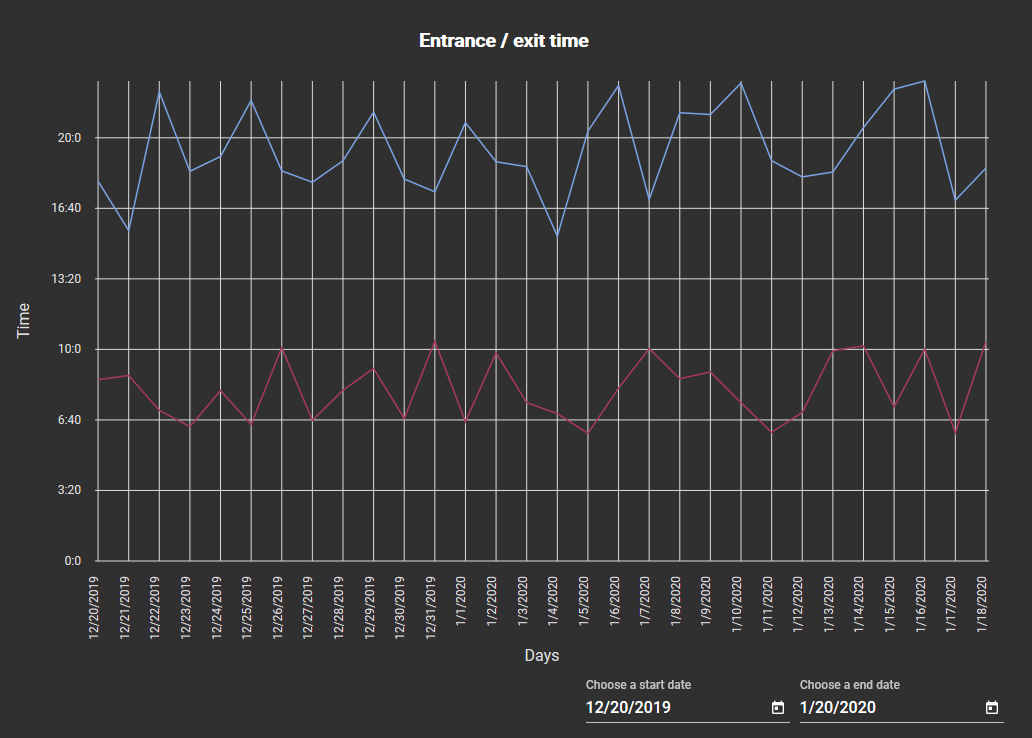


При селектиране на даден потребител от списъка, след него се визуализират две графики, свързани с данните на този потребител, като всяка от тях дава възможност ограничаване на времевия период за които да се генерират.

Първата графика показва времето прекарано на работа спрямо осем часов работен ден под формата на дневна база. Където положителна стойност означава, че потребителя е прекарал повече от осем часа на работа, а отрицателна означава, че не е успял да изкара осем часа на работа.

В сметките се взимат само действително прекараните часове на работното място, т.е. е предвидена възможността за обедна почивка или друг вид краткосрочно отсъствие от работното място.



****Втората графика представлява съпоставяне между часа на идване на работа спрямо часа на тръгване от работа на дневна базата.

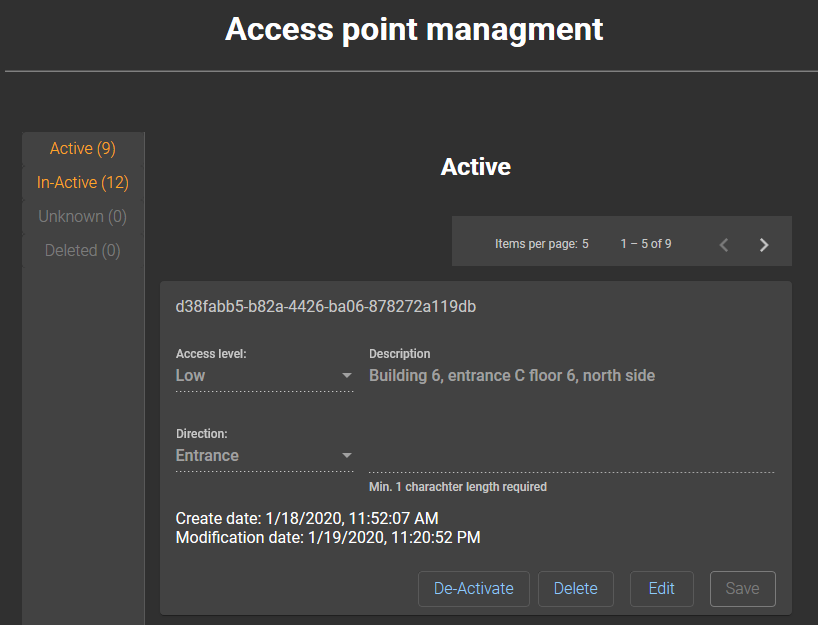
**Access point management**Визуализира списък с точките за достъп. Под точка на достъп се разбира четеца изграден на базата на ардуино. Точките на достъп могат да бъдат в четири различни състояния: Active (активно), In-Active (неактивно), Unknown (непознати за системата), Deleted (изтрити). Различните състояния са достъпни през ляво позиционирано странично меню.

Когато за първи се направи опит за достъп през дадена точка за достъп, тя бива регистрирана от системата като Unknown (непозната), отговорност е на администратора да активира и попълни нейните данни. Това е принципа за регистриране на точка за достъп в системата.

Всяка точка за достъп е представена от следните свойства:   
**Access level** – минимално ниво на достъп нужно на даден потребител за да бъде пропуснат. Възможните стойности са Low, Mid, High   
**Direction** – оказва дали точката на достъп е входна или изходна, това е от значение при генерирането на справките от Statistic частта.  
**Description** – кратко описание за точката на достъп. Тук може да се впише на в коя сграда се намира и на кой етаж.  
**Create Date** – да на създаване на точката за достъп в системата  
**Modification Date** – дата на последна промяна в системата.

Следните операции могат да бъдат изпълнени върху дадена точка за достъп:

**Activate / De-Activate** – активиране или деактивиране на точка за достъп. Не активните точки за достъп не позволяват достъп на нито един потребител, независимо от техните нива на достъп.   
**Delete** – изтрива точка на достъп. Изтритата точка на достъп също не позволява достъп на нито един потребител. Разделението между изтрита и неактивна точка на достъп е чисто административна, а не толкова функционална. Когато дадена точка за достъп се повреди или вече не се ползва би било подходящо тя да бъде изтрита, докато деактивирането е по – подходящо за временно спиране на достъпа.  
**Edit** – позволява редакция на горе изброените свойства.

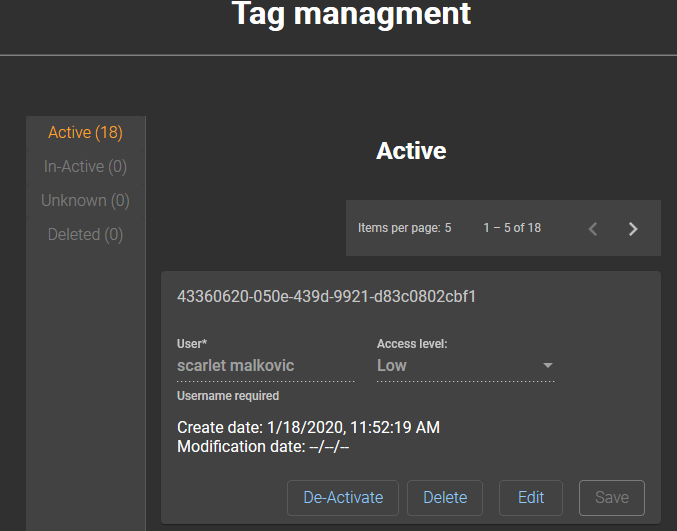
****

**Tag management**Визуализира списък с идентификаторите. Идентификаторите идентично на точките за достъп също имат четири възможни състояния: Active (активни), In-Active (неактивни), Unknown (непознати), Deleted (изтрити). Различните състояния са достъпни през ляво позиционирано странично меню.

Аналогично на точката за достъп, регистрацията на нови идентификатори се случва при първи опит за достъп през определена точка с даден идентификатор. Отговорността е на администратора да попълни данните за идентификатора и да го активира.

Всеки идентификатор притежава следните свойства:  
**User** – името на собственика на идентификатора  
**Access Level** – ниво на достъп на идентификатора  
**Create Date** - дата на създаване на идентификатора в системата **Modification Date** – последна промяна на идентификатора в системата

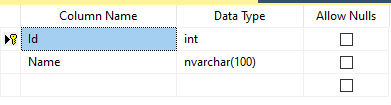
Следните операции могат да бъдат изпълнени за даден идентификатор:  
**Activate / De-Activate** – активиране или деактивиране на идентификатор. Деактивирания идентификатор няма право на достъп през която и да е точна на достъп.  
**Delete** – изтриване на идентификатор  
**Edit** – редактиране на горе посочените свойства на идентификатора

****

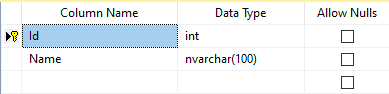
**3.4. База данни**

Използва се SQL Server Express Edition. Базата е разделена на четири логически груби функционалности под формата на схеми:  
access\_control, administration, log, stat, като във всяка схема се съдържат съответните таблици, функции, процедури и тригери.

[**access\_control**].[**AccessLevels**] – таблица съдържаща възможните нива на достъп за идентификаторите и точките на достъп. Биват три вида: Low (ниско), Mid (средно), High (високо)

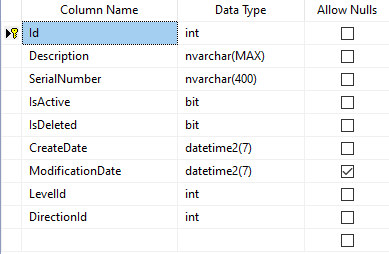
  
Id – уникален пореден номер на нивото на достъп  
Name – името на нивото на достъп (Low, Mid, High)

[**access\_control**].[**Direction**] – таблица съдържаща възможните ориентации на точките на достъп: Entrance (вход), Exit (изход)



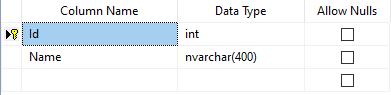
Id – уникален номер на ориентацията на точките на достъп  
Name – име на ориентацията (Entrance, Exit)

[**access\_control**].[**AccessPoints**] – таблица съдържаща информация за точките на достъп.



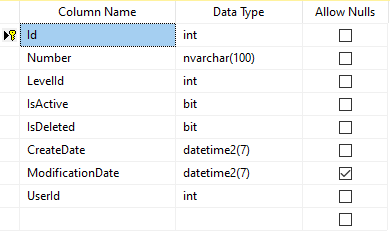
Id – уникален номер на точката на достъп  
Description – описание за точката на достъп  
SerialNumber – уникален номер с който се идентифицира точката на достъп  
IsActive – показва дали точката на достъп е активна  
IsDeleted – показва дали точката на достъп е изтрита  
CreateDate – дата на създаване на записа  
ModificationDate – дата на последна промяна  
LevelId – навигационен ключ към таблицата за ниво на достъп.  
DirectionId – навигационен ключ към таблицата за ориентация

[**access\_control**].[**Users**] – таблица съдържаща информация



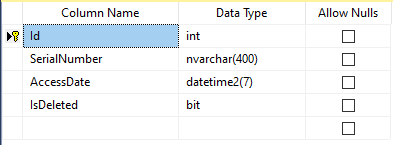
Id – уникален номер на потребителя  
Name – име на потребителя

[**access\_control**].[**Tags**] – таблица съдържаща информация за идентификаторите



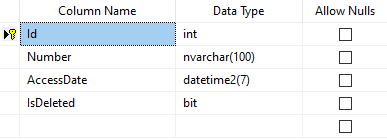
Id – уникален номер на идентификатора  
Number – уникален номер с който се идентифицира идентификатора  
LevelId – навигационен ключ таблицата с нива на достъп  
IsActive – показва дали идентификатора е активен  
IsDeleted – показва дали идентификатора е изтрит  
CreateDate – дата на създаване на записа в базата  
ModificationDate – дата на последна промяна  
UserId – навигационен ключ към таблицата с потребители, регистрирани в системата като притежатели на идентификатори

[**access\_control**].[**UnKnownAccessPoints**] – съдържа информация за непознатите точки на достъп. Това е всяка точка на достъп, която е направила заявка към сървъра, но няма информация за нея.



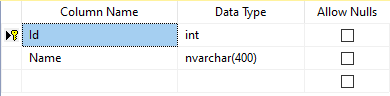
Id – уникален номер на непознатата точка за достъп  
SerialNumber – уникален номер по който се идентифицира  
AccessDate – последна дата към която е правен опит за достъп през непознатата точка за достъп  
IsDeleted – показва дали е изтрита

[**access\_control**].[**UnknownTags**] – съдържа информация за непознатите идентификатори. Това е всеки идентификатор, който е бил използван за опит да се получи достъп без да има информация за него в базата данни.



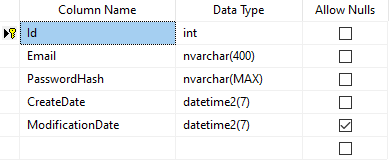
Id – уникален номер на непознатия идентификатор  
AccessDate – последна дата към която е оправен опит за достъп с идентификатора  
IsDeleted – показва дали е изтрит

[**administration**].[**Roles**] – съдържа информация за ролите на администраторските потребители. Ролята на този етап е само една: Admin (администратор)



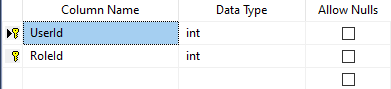
Id – уникален номер на ролята  
Name – име на ролята (Admin)

[**administration**].[**Users**] – съдържа информация за администраторските потребители.



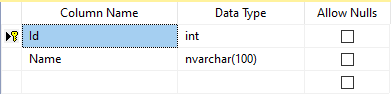
Id – уникален номер на потребителя  
Email – емайл на потребителя  
PasswordHash – hash-рана версия на паролата на потребителя  
CreateDate – дата на създаване на записа  
ModificationDate – дата на последна промяна на записа

[**administration**].[**UsersRoles**] – междинна таблица описваща връзка много към много между потребителите и ролите.



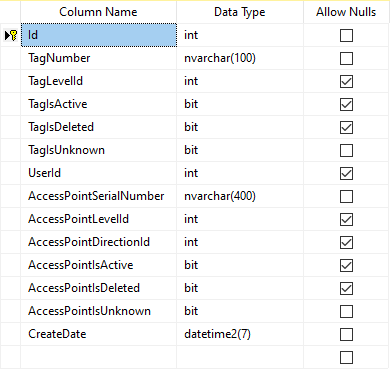
UserId – навигационен ключ към таблицата с потребители  
RoleId – навигационен ключ към таблицата с роли

[**log**].[**Type**] – съдържа информация за видовете логове, които се записват в базата от страна на клиента. Следните типове са налични: Error (грешка)



Id – уникален номер типа  
Name – име на типа (Error)

[**stat**].[**Events**] – съдържа информация за идентификаторите и точките на достъп за конкретен момент за извършен опит на даден потребител да премине през определена точка на достъп. Чрез данните в тази таблица се генерират справките за даден потребител.



Id – уникален номер на събитието  
TagNumber – идентификационен номер на идентификатора  
TagLevelId – ниво на достъп на идентификатора  
TagIsActive – показва дали идентификатора е бил активен  
TagIsDeleted – показва дали идентификатора е бил изтрит  
TagIsUnknown – показва дали идентификатора дали е бил непознат  
UserId – навигационен ключ към таблицата с потребители, които са притежатели на идентификатор  
AccessPointSerialNumber – идентификационен номер на точката за достъп  
AccessPointLevelId – навигационен ключ към таблицата с нива за достъп  
AccessPointDirectionId – навигационен ключ към таблицата за ориентация  
AccessPointIsActive – показва дали точката за достъп е била активна  
AccessPointIsDeleted – показва дали точката за достъп е била изтрит  
AccessPointIsUnknown – показва дали точката за достъп е била позната  
CreateDate – дата на създаване на събитието

**3.6. Изходен код и използван software**

Кода на приложението е публично достъпен на: <https://github.com/vasiloreshenski/RFID>

Използван software:

**Visual Studio 2017 Community Edition** – за разработка на REST API-то  
**Visual Studio Code** – за разработка на website-а  
**Arduino IDE** – за разработка на кода изпълняван от ардуиното  
**SQL Server Expression Edition** – база данни  
**Internet Information Service (IIS)** – web сървъра за обслужване на website-а и REST API-то.  
**Fiddler** – за тестване на заявките по време на разработка на REST API-то.  
**Notepad++** - текстов редактор  
**AutoDesk 123D Design** – за 3D моделиране на кутията  
**Autodesk AutoCAD LT 2020** – за проектиране на 2D схема на кутията  
**WireShark** – за проследяване на TCP/IP трафика към сървъра