**4. Анализ на получените резултати, приложимост и изводи**

При анализ на една цялостна система за контрол на достъпа може да се разгледат множество характеристики, като по – важните са:

1. Бързодействието на системата
2. Сигурността
3. Капацитета на системата
4. Гъвкавост при интегриране

Това ще даде ясна представа къде са силните и слабите страни на системата и възможните посоки за подобрение на им.

Използван hardware на сървърната част: Intel Core i5-3320M 2.60 GHz (2 ядра), 8 GB RAM, SSD 120 GB, Windows 10

**4.1.** **Бързодействие на системата**

**Сървър**

Направен е тест на сървърната част представляващ две паралелни заявка през няколко секунди в продължение на една минута. Теста има за цел да симулира ситуация в която имаме сграда с два входа през които потребителите се автентикират. Резултат трябва да покаже, че REST API-то не изисква сериозни ресурси и работи с лекота при нормални условия.

Теста е реализиран чрез разработка на конзолно приложение, което изпраща заявка до сървъра.

**Резултата**: средното време за което се обработва 1 заявка докато системата с натоварване от 2 паралелни заявки е 53 ms.

Следва натоварващ тест имащ за цел да покаже, възможността на системата да се справи с по – сериозен обем от потребители. Теста изпраща 200 паралелни заявки до сървъра в продължение на една минута. Теста симулира сценарии в които имаме 200 точки за контрол на достъп, като през всяка една през 1 секунда има потребител.

**Резултат**: средното време обработка на една заявка докато системата е натоварена с 200 паралелни заявки е 88 ms.

Извода, който може да се направи е, че REST API-то изисква минимален ресурс и е подходяща дори и за крайно натоварени обекти.

**Четец**

Направен е тест на бързодействието на четеца. В това се включва времето нужно за прочитане на идентификатора, изпращането на данните до сървъра, връщане на отговор от сървъра – тества се цялостното бързодействие на системата.

За да се отчете времето се ползва millis() функцията, която е част библиотеката на ардуино. Тя връща текущото време от стартирането на ардуиното в милисекунди. Измерва се времето преди прочитане на идентификатора и времето след индикацията за отговор от сървъра и накрая се прави разлика между двете за да се изчисли изминалото време.

**Резултат**: Направени са 10 заявки до сървъра и осреднената стойност е 1500 ms.

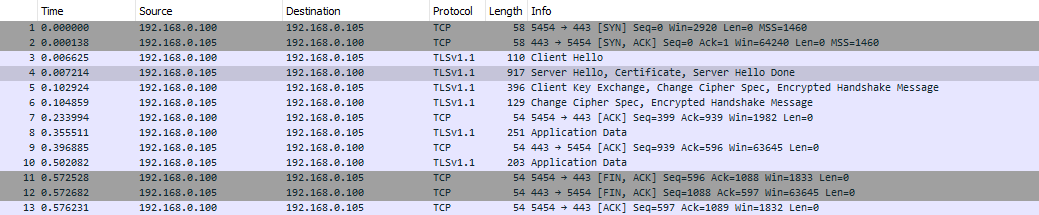
Според изследване направено за определяне на човешкото възприятие от гледна точка на работа с компютър стигат до извода, че забавяне от 100 ms е неуловимо за потребителя, а закъснение от 1000 ms е горната граница в която потребителя не би се разсеял, независимо че ще забележи забавянето. [22]

Времето към което трябва да се стремим е около 1000 ms, от което може да направим извода, че скоростта на четеца е приемлива, но не задоволителна - допълнително усилие в тази посока е желателно. Възможно решение би било усилване на сигнала от антената на четеца с което да се увеличи максималното разстояние на комуникация между четеца и идентификатора.

**4.2. Сигурност**

**Връзка между четец и сървър**

Цялата комуникация между четеца и сървъра се осъществява на базата на HTTPS. След анализ на трафика между четеца и сървъра посредством WireShark, се вижда, че комуникацията е HTTPS и е избрана версия на протокола TLS 1.1.



Фигура 4.1. Снимка на пакетите между четеца и сървъра анализирани чрез WireShark

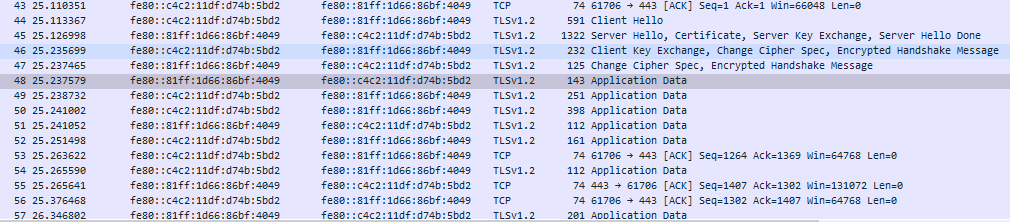
TLS 1.1 стандарта е създаден през 2006 година. Към текуща дата протокола се води остарял, също така водещите browser-и съветват преминаването към последна версия 1.2 на протокола. [23]   
Последната версия на firmware-а на ESP8266 Wi-Fi модула има поддръжка за TLS 1.2.

За да се подсигури сигурна връзка между четеца и сървъра е необходим update до последна версия на firmware-а на ESP8266 Wi-Fi.

**Връзка между website и сървър**

Тук комуникацията също е изцяло на HTTPS, като протокола, който се използва ще зависи пряко от версията на операционната система и browser.

С Windows 10 и FireFox v72.0.1 се използва TLS 1.2.



Фигура 4.2. Снимка на пакетите между browser и сървъра анализирани чрез WireShark

**4.3. Капацитет на системата**

**Брой връзки**

От гледна точка на максимален брой паралелни заявки това зависи пряко от инфраструктурата: сървър и рутер. REST API-то само по себе си няма зададени ограничения.

**Размер на базата данни**

Направен е анализ на максималния размера на един запис в базата за таблиците, които се очаква да нарастват във времето. Това ще ни даде ясна представа за нуждите на системата от към дисково пространство.

За да се направи анализа се използва функция от SQL Server, която предоставя информация за размерите на редовете в дадена таблица

dbcc showcontig (‘{table\_name}’) with tableresults

Трите таблици които се очаква да нарастват са:

[access\_control].[Tags] – пази информация за идентификаторите. Максимален размер за един запис е 112 bytes.

[access\_control].[AccessPoints] – пази информация за точките на достъп. Максимален размер за един запис е 213 bytes.

[stat].[Events] – пази информация за всеки опит за достъп осъществен през дадена точка на достъп и идентификатор. Максимален размер за един запис е 187 bytes.

За да имаме реална постановка при изчислението на размерите ще разгледаме обект с 500 точки за достъп и 50 000 идентификатора. Подобна система би могла да бъде фирма за логистика.

500 x 112 bytes = 54 KB  
50 000 x 213 bytes = 10.65 MB

За да се пресметне размера на последната таблица ще вземем по – реален пример. Размера зависи изцяло от броя опити за проверка на контрол на достъп.   
За пример ще вземем данните за натоварването на метро-станциите в Торино, Италия. Според публична информация за един ден преминат 155 000 пътника, а за година 41 милиона. [24]

41 000 000 x 187 bytes = 7.67 GB на годишна база

В днешно време твърдите дискове варират в размери от 20 GB до 12 TB, като при средностатистическия потребител най – често срещаните варират от 200 GB до 2 TB, а сървърните са най – често над 1 TB. Извода, който може да се направи, е че системата няма сериозни изисквания от към дисково пространство и продължителното използване на системата дори и в крайно натоварени обекти би била покрита от днешните стандарти за твърди дискове.

**4.4. Гъвкавост при интегриране**

REST API-то и website-а са написани на езици за програмиране, които са cross-platform. Това позволява интегрирането им на всички водещи операционни системи като Windows, Linux, Mac OS, Darwin.

**Заключение**

Настоящата дипломна работа разглежда принципите на работа в една система за контрол на достъп с административен модул базирана на RFID технологията посредством Arduino микро-контролер за четец и безжична комуникация към сървърна част, както и цялостно изграждане, и изпробване на системата в реални условия.

Системата е направена с цел висока сигурност, бързодействие и възможност за анализ на данните.

Безжичната комуникация, голямата модулност на Arduino-то, както и използването на широко разпространени езици за програмиране, голямата екосистема под формата на библиотеки и модули, предоставят сериозни удобства при изграждането и поддръжката на четеца.

Минималните изисквания, интеграция върху широк набор от операционни системи и възможността за използване на вече изградени инфраструктурни решения като сървъри и безжични мрежи, улеснява значително интегрирането на REST API-то и административния website.

Съвременни технологии като HTTP, безжични мрежи и RFID ще продължат да бъдат активно развивани и подобрявани, което ги прави правилния избор за изграждането на една такава система.