|  |  |
| --- | --- |
| A black and white logo  Description automatically generated | **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**  **ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ** |

**КУРСОВ ПРОЕКТ**

**„****Проектиране и реализация на система за контролиран достъп“**

**по „Програмни Среди”**

**Изготвен от**:

**Филип Ивайлов Андонов**

**Фак. №: 121222054, Група: 38**

[fiandonov@tu-sofia.bg](mailto:fiandonov@tu-sofia.bg)

***2024***

***София***

Съдържание

[Съдържание 1](#_Toc197357004)

[Увод 2](#_Toc197357005)

[Изложение 3](#_Toc197357006)

[Архитектура 3](#_Toc197357007)

[Програмен дизайн 5](#_Toc197357008)

[Models слой 5](#_Toc197357009)

[Views слой 5](#_Toc197357010)

[ViewModels слой 6](#_Toc197357011)

[Data слой 6](#_Toc197357012)

[Security слой 7](#_Toc197357013)

[Services слой 7](#_Toc197357014)

[Тестови програми 8](#_Toc197357015)

[Програмна реализация 8](#_Toc197357016)

[Резултати и анализ 9](#_Toc197357017)

[Контролно приложение 9](#_Toc197357018)

[Тестови програми 12](#_Toc197357019)

[Заключение 14](#_Toc197357020)

[Източници 14](#_Toc197357021)

# Увод

В днешно време процесът на урбанизация се засилва все повече. Земеделието се модернизира и човешкият труд в сферата вече е все по-малко търсен. Докато средната възраст в провинцията расте, младите трудоспособни хора се преселват в големите градове и започват работа в офиси и предприятия. Според проучване на Националния статистически институт (1) в Българските градове към 31.12.2020 г. пребивават 72.9% от населението. Това създава нов проблем – сградната сигурност. Служителите и фирмената собственост имат нужда от защита против злонамерени лица. При липса на подходящи мерки е възможен фирмен шпионаж, оронване на престижа на предприятието и/или служителите, изтичане на ценни данни, кражби и саботаж. Протекцията се случва най-често чрез жива охрана, но в голямо предприятие с много служители тя е неефективна и скъпа. Това е причината охранителите да работят в симбиоза със системи за сигурност като видеонаблюдение и системи за контрол на достъпа.

Целта на този курсов проект е да спомогне подобряването на сигурността, чрез проектирането и разработването на гъвкава система, способна да работи с различни модули за физически достъп.

От този проект се очаква да намали цената на охранителната дейност и да повиши нивото на сигурност чрез имплементацията на технологично решение.

# Изложение

## Архитектура

С оглед лесна поддръжка, курсовият проект предвижда гъвкава архитектура. На практика поне четири основни компонента съставят проекта: Система за заключване, Система за прочитане на карта за достъп, MQTT брокер и Система за установяване на контрол над заявките. На фигура 1 е изобразена опростена диаграма на събитията в системата.

A close-up of a white sheet

AI-generated content may be incorrect.

Фигура 1.

Потребител притежава препрограмирана информационен източник, съдържащ уникален идентификационен код. Той доближава устройството до модул, в който има хардуерен четец. Информацията за картата се предават посредством MQTT заявка към брокер на тема *„cardreader/{идентификационен номер на четеца}/request“*. MQTT сървърът е наличен чрез IP/TCP по мрежата на порт 1883. Към същият брокер и тема се свързва и контролното приложение, написано на C#. При получаване на заявка, то проверява достоверността на информацията и дали съответният потребител има право да отключи дадения портал. Ако няма – то с цел сигурност не се подава никакъв сигнал. Ако обаче лицето притежава носител, имащ право да премине, то тогава се пуска обратно MQTT заявка към отключващия механизъм на тема *„actuator/{идентификационен номер на отключващият механизъм}/response“*. Тази заявка се прехвърля през сървъра към съответния свързан механизъм.

Модулите за четене на устройство и за отключване на портала могат да бъдат най-различни: например RFID четец + RaspberryPi, NFC четец с Arduino или четец на магнитна лента с ESP32. За целта на курсовия проект, те са заменени с тестови програми, написани на python, които ще бъдат разгледани в друга глава.

За MQTT сървър е избран Eclipse Mosquitto MQTT брокер [2] заради своята лекота, стабилност и съвместимост с MQTT стандарта, който е подходящ за системи с ограничени ресурси и изисквания за бърза и надеждна комуникация. Той е с отворен код, широко поддържан от общността и лесен за интеграция в IoT среди, където се изисква ниска латентност и минимален трафик. Освен това Mosquitto предлага сигурност чрез TLS криптиране и поддръжка на удостоверяване, което го прави надежден избор за реални приложения, особено в сферата на сигурността.

Контролното приложение е написано на C#. Използва MVVM архитектура (фиг. 2). View слоят се състои от редица страници и прозорци, написани на XAML и C#, използвайки Windows Presentation Foundation (WPF), част от .NET [4] платформата. Той е отговорен само и единствено за репрезентирането на програмата в удобен за потребителя графичен интерфейс.

A diagram of a work flow

AI-generated content may be incorrect.

Фигура 2.

Данните за него се зареждат във ViewModel-а. Той от своя страна черпи всички данни като използва методи на Unit Of Work, които позволяват удобно използване на различни заявки към базата данни. Unit Of Work работи със слоя Repositories(хранилища), който съдържа в себе си самите заявки за моделите, а Access Control Context отговаря за достъпа до базата данни и съхранението на запазените промени в нивото с хранилищата. Models съхранява всеки модел, който се използва в приложението. За база данни е избрана технологията SQLite, защото е много лесна за работа, съхранява данните локално и е напълно безплатна за използване. Тя позволява удобството на всяка релационна база данни, но е значително по-лека за машината на която върви от други нейни еквиваленти като MySQL Server или MariaDB. MQTT Service върви паралелно, като постоянно слуша за нови заявки от MQTT сървъра. Той обработва събитията, свързани с постъпване на заявка за отключване, проверява правата на заявителя чрез Access Evaluator и съхранява успешните опити чрез Unit Of Work. Console Manager служи за принтиране на съобщения в конзолата по време на разработката.

## Програмен дизайн

### Models слой

Слоят за моделите съдържа всички описани модели на данните. Те са четири.

1. User – Това е потребителят картодържател. Запазва се само името му и уникален идентификационен код.
2. Card – Съдържа Id, име на картата, ниво на достъп и картодържател. Един човек може да има повече от една карти, но една карта има само един собственик.
3. CardReader – Съдържа името на локацията на която е поставен, Id и ниво на достъп (нивото на картата трябва да бъде по-голямо или равно, за да бъде допуснато през съответния портал).
4. AccessTime – Събитие на успешно маркиране на карта. Съдържа времето на чекиране, както и участвалите портал и карта.

### Views слой

View слоят отговаря за визуализацията на приложението и потребителския графичен интерфейс. Програмата започва в App.xaml и App.xaml.cs. Там се инициализират инстанциите на Unit Of Work, Access Control Context, Mqtt Service и първия прозорец, който се отваря – MainViewModel, който пък от своя страна зарежда Access Log Page страницата в себе си.

Access Log Page, фигура 3, показва информацията за всеки успешен достъп в табличен вид. В съответния .cs файл е изнесена логиката за извикването на зареждането на данните, опресняването на интерфейса и прехвърлянето в други страници.

Users Page, фигура 5, изобразява информацията за всички потребители, както и активира верига, която изпълнява CRUD заявки върху потребител от базата данни.

User Details Page, фигура 4, показва информацията за един потребител, който е преминал през портал с дадена карта.

Cards Page, фигура 7, позволява на администратора да извършва CRUD заявки над картите, записани в базата данни.

Readers Page, фигура 6, дава възможност за работа с порталите и данните, свързани с тях. От тук могат да се добавят или премахват четци, които ще бъдат поставяни на съответните помещения.

### ViewModels слой

Този слой съдържа само един файл: MainViewModel. Той е отговорен за инициализирането на Unit Of Work, MQTT Service и за зареждането на данните в AccessLogPage (защото това е първата страница, която се появява). Също така обновява интерфейса при постъпване на нова успешна MQTT заявка.

### Data слой

Data слоят се състои то 3 основни вида класове: AccessControlContext, UnitOfWork и репозиторита.

1. AccessControlContext е клас, отговорен за установяването на връзка с базата данни и за създаването ѝ ако не съществува. Също така тук се зареждат и примерни данни.
2. UnitOfWork и IUnitOfWork дефинират репозиторита, чрез които се изпълняват CRUD заявки към базата данни. Също така тук се съдържа целия контекст за работа с базата на едно централизирано място. Този клас и интерфейс се използват за финалното съхранение на данните в базата.
3. Репозиторитата са разпределени в два вида файлове:
   1. GenericRepository и IGenericRepository дефинират общите за всички модели CRUD заявки. Освен тях съществуват и други спомагателни методи като GetAllAsync (връща всички резултати) и GetTrackedOrAttach (позволява работа с дублирани елементи).
   2. Надграждащи конкретния модел репозиторита. Тук се добавят специфични към модела заявки като:
      1. AccessTimeRepository и IAccessTimeRepository – GetLatestAsync, GetLatestWithReaderAsync, GetByCardAsync и др.
      2. CardReaderRepository и ICardReaderRepository – GetByAccessLevelAsync и GetByLocationLikeAsync.
      3. CardRepository и ICardRepository – GetByNumberAsync, GetCardsForUserAsync и GetAllWithOwnerAsync.
      4. UserRepository и IUserRepository – GetByCardNumberAsync и GetUsersWithCardsAsync.

### Security слой

Слоят за сигурност съдържа цялата функционалност, свързана с верификацията на картата и работата с нивата на достъп. В случая е нужен само един клас – AccessEvaluator, който проверява дали дадена карта е с достатъчно ниво на достъп спрямо четеца.

### Services слой

Слоят на услугите съдържа допълнителни класове, които са абстрахират дадена функционалност. Целта му е да поддържа чистата архитектура като позволява на определена логика да бъде енкапсулирана в съответния нужен клас.

Пример за такъв клас е Console Manager, който се грижи за отварянето на конзолата в началото на приложението. Тя се използва за дебъгване и по-лесно демонстриране на функционалността.

Друг пример е MQTTService. Той е отговорен за работата с MQTT сървъра. Класът има достъп до UnitOfWork, MQTT библиотеката на .NET, AccessEvaluator и логика за обновяването на потребителския интерфейс. В конструктора си инициализира всички полета, установява контакт с MQTT брокера (към момента е използвано локално IP и порт 1883) и настройва асинхронни функции, които ще бъдат изпълнявани при постъпване на събитие като нова заявка или загуба на връзка с брокера. Постъпването на заявка се случва като класът слуша на тема *„cardreader/+/request“* („+“ означава всяко ID). При получаване на заявка въпросния идентификационен номер в темата се използва за определяне на изпращача (четеца), а информацията от заявката съдържа уникалния идентификационен код от устройството на потребителя. Този код се сферява посредством AccessEvaluator и ако потребителят има достъп се създава се нов запис (AccessTime), който запазва чрез UnitOfWork. После се подава съобщение до съответния портал на тема „actuator/{идентификационен код на портала}/response“. Това съобщение се обработва от модула и трябва да отвори съответната врата.

## Тестови програми

Ефективната разработка и качествено тестване предполагат създаването на програми, които да симулират работата на устройствата на порталите. За целта са изработени две програми, едната симулираща заявка от четец, а другата получаваща заявка за отключване. Те са написани на Python, защото това е ефективен, познат и добре документиран език, който поддържа работа с MQTT чрез Paho MQTT [3].

Едната програма се казва lock\_publisher.py и се извиква по следния начин:

python lock\_publisher.py <cardreader\_id> <card\_id>

в терминала, където съответно аргументите се сменят със съответните идентификационни номера. Програмата взима тези номера и генерира заявка за MQTT сървъра на локално IP и порт 1883, която бива изпратена на тема „cardreader/{cardreader\_id}/request“ със съдържание идентификационния код на симулирана карта за достъп.

Втората програма се нарича actuator\_subscriber.py и се извиква аналогично:

python actuator\_subscriber.py <actuator\_id>

в терминала, където съответно аргумента се сменя със съответния идентификационен номер на портала. Когато програмата се стартира, тя отваря връзка до същия MQTT сървър като lock\_publisher.py и започва изчакване на заявки за отключване на тема „actuator/{actuator\_id}/response“. При постъпване на нова заявка, тя бива принтирана в терминала.

# Програмна реализация

Целият код може да бъде открит и изтеглен на: <https://github.com/callmeFilip/PS_Course_Work>

# Резултати и анализ

## Контролно приложение

Програмата, която се реализира в курсовата работа, притежава няколко прозореца. На фигура 3 е изобразен главният прозорец.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 3.

Тук може да се види списъкът с всички успешни заявки за отваряне на портал в табличен вид. Колоните на таблицата показват локацията на портала, времето на достъп и идентификационните номера на заявката, картата и портала. При натискане на бутона за детайли излиза прозореца на Фигура 4, който показва информация за потребителя и картата, която е използвал.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 4.

При натискане на бутона за управление на потребителите от главната страница, приложението зарежда страницата, показана на фигура 5. Тук са показани всички потребители на системата и се дава възможност за добавяне на потребител (чрез полето в горния ляв ъгъл и бутона Add), за изтриване на вече съществуващ такъв, както и за връщане назад към основното меню. След натискане на бутона Add, приложението проверява информацията от потребителя и я запазва в базата данни. При маркиране на потребители и натискане на Delete Selected излиза бутон за потвърждаване на изтриването, след което данните биват заличени каскадно от базата данни.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 5.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 6.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 7.

Аналогично са показани и прозорците за редактиране на портали и карти на фигура 6 и 7. Всяка CRUD операция се отразява в графичния интерфейс динамично, отново резултат от MVVM и .NET архитектурата.

## Тестови програми

Фигура 8 показва изпратена заявка за отключване от симулирана ключалка. Тестовата програма изпраща заявка от „четец“ с идентификационен номер 3 и информация за сканиране от карта с идентификационен код 2.

A screen shot of a computer

AI-generated content may be incorrect.

Фигура 8.

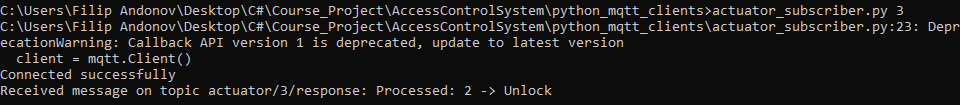
Тази информация бива предадена през MQTT брокера до контролиращата програма, която маркира този опит като успешен и го логва на фигура 9, след което изпраща сигнал за отключване на портал идентификационен код 3.

A black screen with white text

AI-generated content may be incorrect.

Фигура 9.

На фигура 10 се вижда резултатът от втората тестова програма, която симулира ключалка. Тук се вижда получаване на команда за отключване (само на съответната ключалка), която ще се даде само и единствено ако сканираното устройство наистина има достъп.



Фигура 10.

# Заключение

В разглежданата тема „Проектиране и реализация на система за контролиран достъп“ се описва проблем, поставя се цел и се описват предприетите реални стъпки за постигането ѝ. Определена е архитектура, включваща графичен интерфейс, функционални компоненти и тестови програми, позволяващи широкото приложение и навързване на системата. Изготвена е част от използваема система, спомагаща за подобряването на сградното осигуряване.

# Източници

[1] Проучване на Националния статистически институт – https://nsi.bg/sites/default/files/ files/pressreleases/Population2020\_IVGTQG5.pdf

[2] <https://mosquitto.org/>

[3] https://pypi.org/project/paho-mqtt/

[4] https://learn.microsoft.com/en-us/dotnet/