|  |  |
| --- | --- |
|  | **Технически университет – София** |

**Курсов Проект**

На тема:

**„Пректиране и разработка   
нa система за онлайн магазин за храна”**

Изпълнена от:

Десислав Андреев Христов

Факултет: ФКСТ

Специалност: КСИ

Факултетен № 121317006

Група: 222

Дата: 17.01.2018 г.

град София

Съдържание

[Въведение 4](#_Toc486841616)

[Глава. 1. Анализ на темата. Цели и задачи. 6](#_Toc486841617)

[1.1. Нуждата от подобно софтуерно решение 6](#_Toc486841618)

[1.2. Изисквания към системата 7](#_Toc486841619)

[1.2.1. Обработване на голямо количество от данни идващи от устройствата (Big Data) 7](#_Toc486841620)

[1.3. Обзор на подобни системи 9](#_Toc486841621)

[1.3.1. OpenHab 9](#_Toc486841622)

[1.3.2 Home assistant 9](#_Toc486841623)

[1.3.3 Stringify 10](#_Toc486841624)

[1.3.4 Loxone 11](#_Toc486841625)

[Глава 2. Обзор на използваните технологии 12](#_Toc486841626)

[2.1. Java EE 12](#_Toc486841627)

[2.2 Apache Maven 13](#_Toc486841628)

[2.3 Jersey 14](#_Toc486841629)

[2.4 Oracle Database 14](#_Toc486841630)

[2.5 Windows Presentation Foundation 15](#_Toc486841631)

[2.6 Ninject 16](#_Toc486841632)

[2.7 Android 16](#_Toc486841633)

[2.8 Google Cloud Messaging Server 17](#_Toc486841634)

[Глава 3. Проектиране на мултиплатформено IoT базирано решение. 18](#_Toc486841635)

[3.1. Проектиране на сървър приложението 18](#_Toc486841636)

[3.1.1 Проектиране на базата от данни (БД) 20](#_Toc486841637)

[3.1.2 Комуникация между сървър приложението и базата от данни 23](#_Toc486841638)

[3.1.3 Архитектурен дизайн на сървър приложението 24](#_Toc486841639)

[3.2 Проектиране на Андроид приложението 27](#_Toc486841640)

[3.2.1 Ахитектурен дизайн на Андроид приложението 27](#_Toc486841641)

[3.2.2 Комуникация между Андроид приложението и сървър приложението 27](#_Toc486841642)

[3.3 Проектиране на приложението за администратори 29](#_Toc486841643)

[3.3.1 Архитектурен дизайн на приложението за администратори 29](#_Toc486841644)

[3.3.2 Комуникация между сървър приложението и приложението за администратори 31](#_Toc486841645)

[Глава 4. Програмна реализация 33](#_Toc486841646)

[4.1. Програмна реализация на сървър приложението 33](#_Toc486841647)

[4.1.1 Реализация на базата от данни 33](#_Toc486841648)

[4.1.2 Реализация на връзката между сървъра и базата от данни. 35](#_Toc486841649)

[4.1.3 Реализация на Услугите 37](#_Toc486841650)

[4.1.4 Реализация на Диспатчара на заявки 39](#_Toc486841651)

[4.1.5 Реализация на шаблона „*Observer*“. 41](#_Toc486841652)

[4.2 Програмна реализация на приложението за администратори 44](#_Toc486841653)

[4.2.1 Реализация на логин на потребител 44](#_Toc486841654)

[4.2.2 Реализация на “Inversion of control” 45](#_Toc486841655)

[4.2.3 Реализация на връзката със сървъра 46](#_Toc486841656)

[4.2.4 Реализация на добавяне на ново правило. 47](#_Toc486841657)

[4.3 Програмна реализация на Андроид приложението 48](#_Toc486841658)

[4.3.1 Реализация на нотификациите 48](#_Toc486841659)

[Глава 5. Ръководство на потребителя 50](#_Toc486841660)

[5.1. Ръководство на приложението за администратори 50](#_Toc486841661)

[Глава 6. Заключение 58](#_Toc486841662)

[Източници 61](#_Toc486841663)

[Приложение 1. Програмен код 63](#_Toc486841664)

# Въведение

Намираме се на прага на технологична революция, която коренно ще промени начина, по който живеем, работим и се отнасяме един към друг. В своя мащаб, обхват и сложност, промяната ще бъде много силно осезаема. Все още не знаем точно как ще се развие, но едно нещо е сигурно: отношението ни към нея трябва да бъде интегрирано и всеобхватно, с участието на всички заинтересовани страни на глобалната система на управление, от публичния и частния сектор до академичните среди и гражданското общество.

Първата индустриална революция е използвала вода и пара, за да механизира производството. Втората употребява електрическата енергия за създаване на масово производство. Третата впряга електрониката и информационните технологии за автоматизиране на производството. Сега четвъртата индустриална революция надгражда предшестващата - цифровата революция, която се формира от средата на миналия век. Тя се характеризира със създаването на технологии, които размиват физическите, цифровите и биологичните граници. Възможностите на милиарди хора, свързани с мобилни устройства, притежаващи безпрецедентна изчислителна мощ, капацитета за съхранение и достъп до знания, са неограничени. И тези възможности ще бъдат умножени от нововъзникващите технологични пробиви в области като изкуствения интелект, роботиката, интернет на нещата, автономните превозни средства, 3-D печата, нанотехнологиите, биотехнологиите, материалознанието, съхраняване на енергията, както и квантовите компютри.

Ние всички знаем какво е интернет и го използваме по-някакъв начин. Хиляди компании вече са разработили свои уеб-сайтове, мобилни приложения, приложения за настолен компютър и сървърни приложения за да продават своите продукти, за да извършват дадени услуги или просто за да улесняват своите клиенти. Във всеки един момент, в интернет се разпространява огромно количество информация, която е изпратена от едни и е предназначена за други хора. Може да се, каже че „интернета е на хората“. И ето тук е момента да се запитаме: „Защо интернета е само на хората?“, Защо не развиваме още нашата интернет мрежа?“. Истината е, че вече съществуват много идеи и концепции точно в тази насока. В момента сме изправени пред нова технологична революция наречена „Интернет на нещата“. От името става ясно, че към интернет мрежата ще бъдат включени устройства, който ще изпращат, приемат и обработват информация и по този начин ще комуникират помежду си без намесата на хората.

Това е концепция за компютърна мрежа от физически обекти (устройства, превозни средства, сгради и други предмети), притежаващи вградени електронни устройства за взаимодействие помежду си или с външната среда. Тази концепция разглежда организацията на такива мрежи като явление, способно да преустрои икономическите и обществени процеси така, че да изключи необходимостта от участие на човека в част от действията и операциите. Концепцията е формулирана през 1999 г. като разширение на приложението на радиочестотна идентификация при взаимодействието на физически обекти един с друг и с околната среда. Започвайки от 2010-те години, концепцията „Интернет на вещите“ се изпълва с многообразно технологично съдържание и се внедряват нови практически решения и тя се смята за възходяща тенденция в информационните технологии, преди всичко благодарение на повсеместното разпространение на безпроводните мрежи, появяването на облачните изчисления, развитието на технологиите за междумашинно взаимодействие, началото на активния преход към IPv6 и усвояването на програмно-конфигурируемите мрежи.

Интернет на нещата (IoT) е концепция, много по-различна от интернета от вчера. Въпреки че технически погледнато това е един и същ интернет, процесите и функционалността на свързаните устройства, които съществуват в него не са типичните, с които сме свикнали да работим. Много от тях нямат дисплей за визуализация, други изпълняват комплекс от различи дейности и задачи, а трети пък служат само за предоставяне на данни за или до един конкретен обект.

Всъщност интернет на нещата включва почти всичко, което може да си представи човек, от автомобили и къщи до комплексни инсталации и процеси. Широко се дискутира дори и концепцията за т.н. “умен прах”, който ще събира и предоставя информация за всичко, в досег с въздуха около нас.

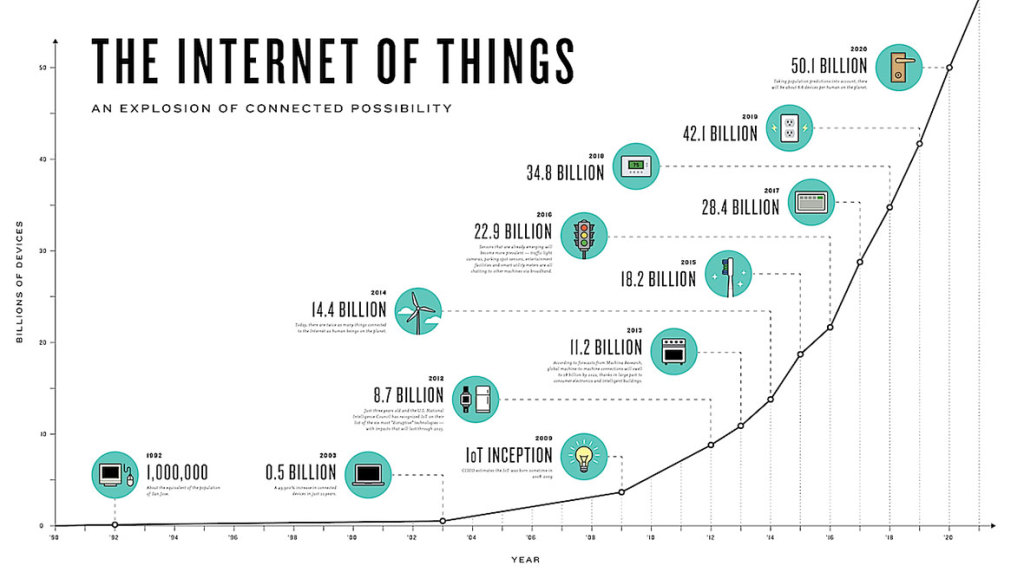
На интернет на нещата (IoT) трябва да се гледа като на платформа за развитие, с изключително разнообразен обхват и функционалност. Това разнообразие е още по-ясно изразено при преглед на широкия спектър от платформи, навлезли вече на пазара, на които може да се базира разработването на редица IoT проекти. „Интернет на нещата“ ще промени заобикалящия ни свят. Едно от нещата, който ще засегне значително е нашия дом и как различните устройства в него изпълняват свойте функции. Всички устройства в нашия дом ще бъдат свързани и ще могат да комуникират помежду си без наша намеса. Естествено за да се постигне това съществуват още много проблеми, който трябва се решат. Като за начало човечеството, например, все още не е готово да се довери на тази нова технология. Хората трябва да могат да управляват всички тези устройства и да оставят само най-незначимите решения да бъдат взимани без тяхна намеса. Също така съществуват и много проблеми свързани с интеграцията на тези устройства и начина, по който те да си комуникират.

Проектът „Хоуми“ се стреми да намери решение на точно тези проблеми. Решението предоставя опция както за конфигуриране на някои действия, така и изпълнението на някои от тях автоматично. Системата се състои от сървър приложение, приложение за настолен компютър и мобилно приложение. Сървър приложението се стреми да улеснява комуникацията между различните устройства и да даде възможност на потребителя да конфигурира решенията, които да бъдат взимани от устройствата. Системата предоставя прегледен потребителски интерфейс и също така предоставя мобилно приложение за операционната система Андроид.

# Глава. 1. Анализ на темата. Цели и задачи.

## 1.1. Нуждата от подобно софтуерно решение

Вследствие на големия брой нови устройства, които се появяват на пазара във всеки един момент, „интернет на нещата” ще наложи сериозно обмисляне на подходите за управление на данни. Ръстът на мащабите на „интернет на нещата” ще повлече след себе си възникването на огромно количество данни, които трябва да бъдат обработвани и анализирани в реално време. Според резултатите от най-новите изследвания, до 2030 г. към интернет ще бъдат свързани около 50 млрд. различни датчици, устройства и друго оборудване. [1]



*Фигура 1. Диаграма на всички устройтва, които се очаква да бъдат свъразани към интернет мрежата до 2030 година.[11]*

Тези цифри показват, че ще има огромна нужда от софтуерни решения, които да свързват всички тези устройства, за да може да се извлече някаква полза от тях. Можем само да си представим колко ценна информация, може да носи всяко едно от тези устройства. Нека за момент погледнем нещата не толкова глобално, а се ограничим само до нашия дом. И сега да си представим как всички наши домашни уреди притежават толкова много функционалности и как всички те работят с операционни ситеми и са свързани към нашата интернет мрежа. Но ние нямаме никаква полза от това, защото те все пак си остават домашни уреди и ние ги използваме по абсолютно същия начин. Ето тук идва нуждата от подобен софтуер. Благодарение на него те няма да са вече просто уреди, които включваме и изключваме. Заедно, те вече могат да работят за нас и за нашия домашен уют.

## 1.2. Изисквания към системата

Целта на тази система е да свърже всички „умни“ устройства в един дом с цел управление и автоматизация. Ситемата трябва да може да намира и да се свързва с всички устройства в локалната мрежа, а също така да може да се добавят и мобилни устройства, за да може потребителят да получава известия от системата. Цялата концепция на системата е да обединява и свързва всички устройства в дома и да позволява лесна и бърза комуникация между тях. Също така ще предоставя различни опции за конфигуриране на системата, които ще се вземат с най-голям приоритет, когато системата взема решение самостоятелно. Основният компонент на системата трябва да бъде централният сървър, където ще бъде имплементирана бизнес логиката. Тя ще бъде отговорна за вземането на всички решения, които зависят от системата. За нея е необходимо да се работи с база от данни, където да се съхранява списък със свързани устройства и различни конфигурации на системата за даден потребител. Базата от данни е много важна за функционирането на цялата система, затова тя трябва да бъде много надежна. За последните няколко години операционната система на Google – Android, си извоюва мястото на най-популярна мобилна операционна система в света. Тя се използвана основно при смартфони и таблети, като в последните години нараства и приложението й при смарт телевизори, часовници и коли. Като се oчаква да бъде изпозлвана и за много домашни уреди в бъдеще. Поради настрастващия брой на устройствата с операционна система Android необходимо да се имплементира начин за комуникация с всички устройства с тази операционна система. Първоначалната версия на системата ще поддържа само един протокол за комуникация и това е HTTP и само устройства с операционна система Android. И следователно могат да бъдат свързани само устройства, които могат да приемат и изпращат данни по този протокол.

### 1.2.1. Обработване на голямо количество от данни идващи от устройствата (Big Data)

Голямото количество от данни в днешни дни е глобален проблем в сферата на информационните технологии.[2] С всяка изминала година количеството данни, което трябва да бъде обработено, анализирано и съхранявано след това, се увеличава главоломно. Още по-голям проблем е, когато тези данни трябва да бъдат обработени и анализирани в реално време. Това е и един от най-големите проблеми пред новата технология „Интернет на нещата“. За реализацията на тази система, този проблем също е налице, макар и не в толкова голям мащаб. Очаква се свързаните устройства към системата да не са толкова много на брой и получаването и обработката на данни в реално време да не бъде толкова голям проблем. Все пак това е въпрос, на който трябва да се отдели нужното време и да се имлементира добро решение, което да се справи с голямото количество данни. Системата трябва да бъде готова да приема това количество от данни и да може да вземе правилното решение на база тези данни в реално време.

Една от характерните черти на хората е това, че те трудно свикват с новите неща, особено ако не ги разбират напълно. В сферата на технологиите това винаги е представлявало голям проблем, с който не всички успяват да се справят. Има хиляди примери за компании с иновативни идеи, които не успяват да се реализират на пазара, въпреки че продуктите, които са представили са били много добри и иновативни. Главната причина за провала на тези компании с блестящи и иновативни идеи е, че клиентите не са били готови да приемат дадения продукт и той е бил извън техните разбирания в дадения момент. След известно време тези продукти са се наложили на пазара, естествено от други компании. И понеже тази концепция („Интернет на нещата“) е сравнително нова, е нужно да се предостави решение което да не „плаши“ потенциалните клиенти и те да могат да разберат ползата от него. В момента много малка част от хората са готови да оставят устройствата в дома си да комуникират помежду си и да вземат самостоятелно решения, от които може да зависи сигурността на съответния дом. Още повече, когато тази система е напълно нова и не се е наложила на пазара. А както добре знаем от много примери в нашата история, в света на технологиите винаги е възможно нещо да се обърка. В този ред на мисли, за да се внедри успешно системата на пазара, е нужно тя да притежава адмистративен панел, от който клиентите да имат възможност да конфигурират какво могат да правят различните устройства и какви решения имат право да вземат самостоятелно.

Също така е нужно да има опция за конфигуриране на даден брой събития от клиента и те да се стартират и изпълняват автоматично без намеса от негова страна. Примери за това са събития от вида на: „Когато се събудя“, „Когато не съм вкъщи“, „Когато съм вкъщи“, „Когато се задейства алармата“. Потребителят трябва да може да конфигурира кои устройства да бъдат задействани и какво точно да направят те.

Мобилно приложение

Както вече стана ясно системата трябва да притежава мобилно приложение за операционната система Android. Идеята на това приложение е да изпраща съобщения с актуална информация за състоянието на дадено устройство. Като например ако е инсталирано на климатик, той ще изпраща състояние за текущата температура. Това приложение ще работи на заден план и ще представлява нещо като средство за комуникация с главния сървър. Но за да бъде системата в крак с новите теденции, то също така трябва да работи като средство за известяване на клиента, но само тогава когато е инсталирано на мобилен телефон. Така той ще получава известия за главните действия, които се извършват в момента в дома му, когато той е извън него. Също така, ще получава известия при възникнали проблеми с някой от уредите или ако някоя от алармите за сигурност или датчиците за пожар са били задействани. Известията трябва да носят значима информация за клиента и не бива да бъдат досадни.

Сигурност

Съществува още един голям проблем, освен голямото количество от данни пред глобалното реализирането на „Интернет на нещата“ и това е – сигурността.[3] Все още не са решени проблемите със сигурността и това е един от най-големите фактори, които забавят реализирането на тази технология и всяват колебания в бъдещите потребители. Сегашните технологии за сигурност не покриват критериите за сигурност на тази нова технология и в момента се разработват и тестват нови решения за сигурност в конкретност за тази концепция. Това ще бъде едно от най-трудните задачи пред реализирането на проекта „Хоуми“. Но без съмнение, трябва да се отдели много време в решаването на този проблем. А като се има предвид, че този проблем все още не решен глобално, това ще остане извън рамките на първата версия на системата и ще се търси решение на проблема в следващите версии на продукта.

## 1.3. Обзор на подобни системи

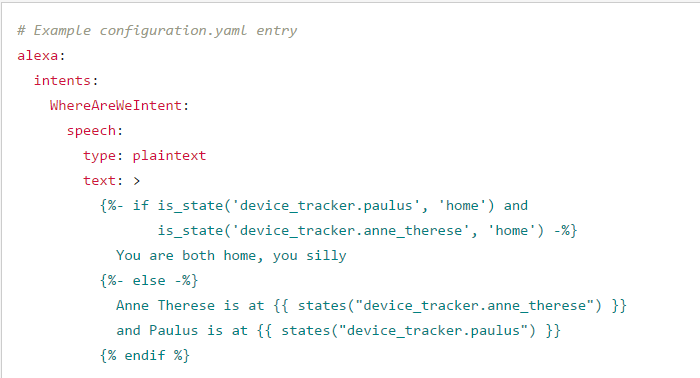
### 1.3.1. OpenHab

Платформата позволява интегриране на много устройства и контролери свързани с автоматизацията на домашните уреди. С помощта на тази платформа потребителите могат да изграждат „умни” домове и офиси. За целта са налични средства за интегриране на множество устройства, както и възможност за изграждане на логиката на взаимодействие между тях. OpenHAB предоставя наготово няколко разновидности на потребителския интерфейс за браузър и мобилни устройства, чрез който лесно може да се упражнява контрол и мониторинг на устройствата. По-рано тази година излезе нова версия на платформата – openHAB 2.0, базирана на проекта Eclipse SmartHome, който цели да ускори представянето и управлението на повече устройства в контекста на „умните” домове. Платформата идва с готови интеграции за над 130 различни устройства, технологии и протоколи, както и с добавки за подобряване на потребителското изживяване. Почти половината от наличните адаптации в openHAB 2.0 използват новата и значително подобрена структура на Eclipse SmartHome. Тя поддържа автоматично откриване и конфигуриране на нови устройства директно през графичния интерфейс. Едно от основните нововъведения е възможността на потребителите да виждат наличните поддържани устройства в своята мрежа и да ги добавят към системата си само чрез едно движение. Вече са налице и механизми за дефиниране на правила, които позволяват на потребителя да зададе условия за взаимна работа на устройствата.

Някои примери за такива правила са запалване на осветлението при засичане на човек в дома, сваляне на щори при силна слънчева светлина, както и контрол на устройства само чрез жестикулации или говор. Колкото и добре да звучи всичко това, тази система има един голям недостатък. И той е, че е прекалено сложен за средностатистическия потребител без никакви познания в областта. Да конфигурира такава система е почти невъзможно за обикновения потребител и би отнело доста време дори на човек, който има значителен опит в областта на информационните технологии, понеже е нужна допълнителна имплементация, за да се конфигурира системата за конкретния потребител.

### 1.3.2 Home assistant

Home assistant е платформа за наблюдение, автоматизиране и контрол на различни устройства в дома. Тази система е с отворен код, а главния сървър е разработен на Python и комуникира през Websocket с интерфейса, който е изграден върху Polymer. Също така приложението предоставя опция за контол на всяко едно от устройствата, като не се пазят данни на клиентите в никакви облачни системи. Налице са и механизми за дефиниране на правила, които позволяват на потребителя да зададе условия за взаимна работа на устройствата. Приложението предлага и опция за контрол на устройствата чрез Google Home или Alexa.



*Фигура 2. Част от Конфигурационен файл на системата „Home assistant“.[12]*

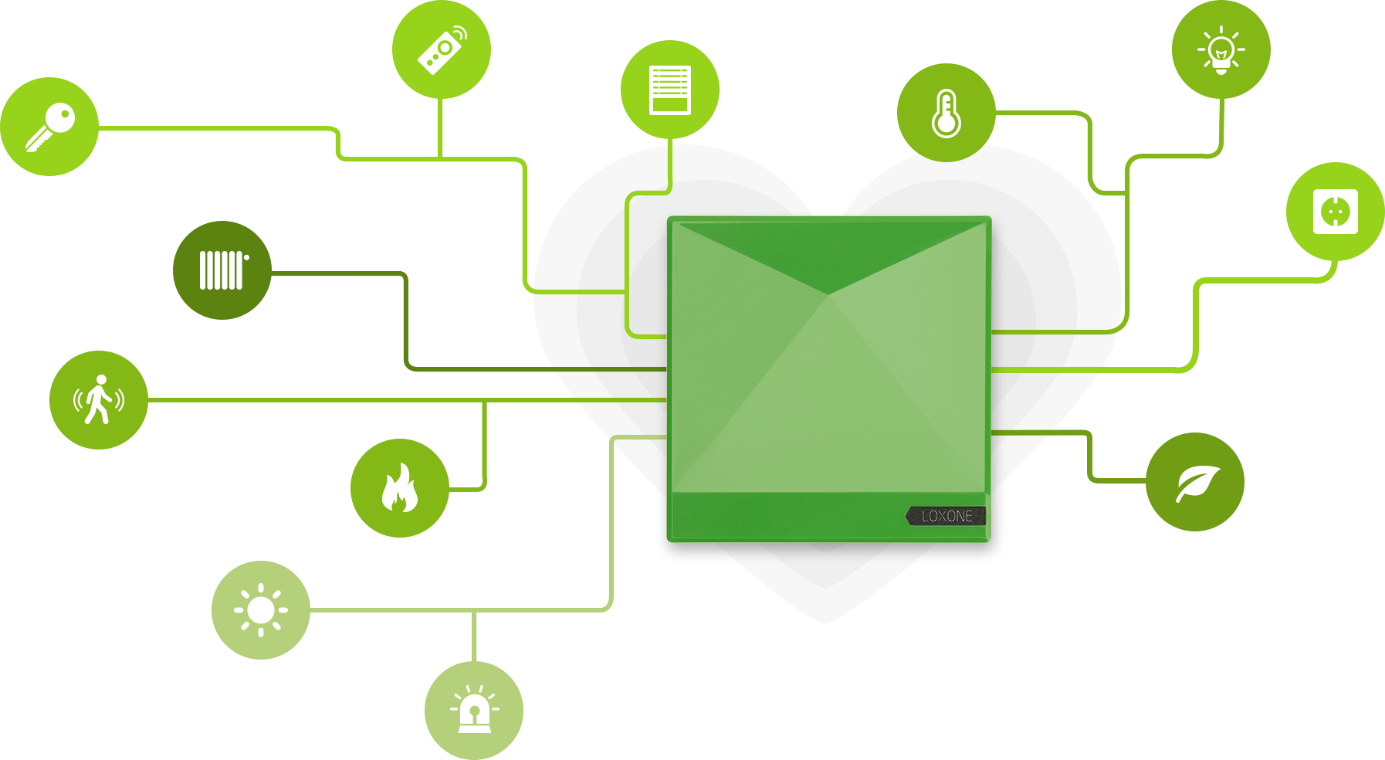
На фигурата по-горе е показана само малка част от един конфигурационен файл за свързване на системата към “Alexa” (чийто производител е известната компания Amazon). Един огромен недостатък, както може да забележите, е че тази система също изисква доста познания и опит в сферата на програмирането, за да може да се конфигурират успешно различните устройства.

### 1.3.3 Stringify

Stringify е мобилно приложение, чрез което можете да се свържете с всички „умни“ устройства в дома и да ги управлявате. Едно от големите предимства на това приложение са така наречените „потоци“. Това е предварително вградена в системата верига от събития, които се изпълнят по-определен ред. Те разбира се могат да се редактират от потребителя и също така потребителя може да създава свои собствени „потоци“. Приложението се предлага и за iOS операционна система. Приложението предлага и опция за контрол на устройствата чрез Google Home или Alexa. Това приложение предоставя много и то полезни функционалности, без нужда от излишно конфигуриране, но е много ниско оценено в Google Play от потребителите на Android.

### 1.3.4 Loxone

Това е пример за система, която е завършена от начало до края. Това е системата, която най-много се доближава до концепцията за „умен” дом. Тя позволява да се автоматизират и контролират всички свързани към системата устройства.



*Фигура 3. Схема на главния сървър на Loxone.[13]*

Тази система притежава главен сървър, който представлява сърцето на системата. Той позволява комуникацията между различните устройства и системи. Loxone е компания с дългогодишен опит, която разработва своя хардуер и софтуер за цялата система. Вследствие на това, може да се направи заключение, че Loxone правят невъзможно интегрирането на други устройства към тяхната система. Като изключим този факт, това е една от най-надежните системи на пазара до момента, естествено и много скъпа за един средностатистически потребител.

# Глава 2. Обзор на използваните технологии

## 2.1. Java EE

Описание на технологията

Java  Enterprise Edition e изчислителна платформа, използвана най-често за създаването на големи проекти в Java общността. Платформата предоставя API среда за изпълнение, за разработване и управление на бизнес софтуер, включително мрежови и уеб услуги в голям мащаб, които са сигурни и надеждни. Java EE е разширеният вариант на Java Platform Standard Edition (Java Standard Edition ). Осигурява API за обектно-релационните планирания, многослойни архитектури и уеб услуги. В платформата се включват главно дизайнерски компоненти, работещи на сървър. Софтуерът за Java EE е написан основно на езика Java. Платформата подчертава конвенцията над конфигурацията и поясненията. XML може да се използва за презаписване на нотациите или да се отклони от началните настройки на платформата. Java EE се дефинира чрез нейните спецификации. Java Community Process спецификациите гласят, че разработчиците на софтуер, които допринасят за развитието на Java EE трябва да отговарят на изискванията за съответствие, за да декларират продуктите си.

Java EE включва много на брой API спецификации – RMI, e-mail, JMS, web services, XML, и други, и дефинира критерии за координирането им. Java EE също поддържа някои спецификации, присъщи единствено за компонентите на Java EE. Те включват Enterprise JavaBeans, connectors, servlets, JavaServer Pages и други технологии за Web услуги. Това позволява на разработчиците на софтуер да създават корпоративни приложения (enterprise applications), които са преносими и мащабируеми.

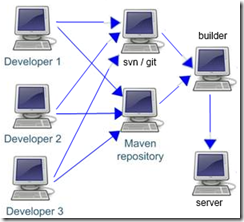
Защо използвам тази технология?

Главната прична поради която използвам тази технология е, защото тя е много надежна и лесно може да се интегрира с други утвърдени технологии.[4] Сървърът за апликации на Java EE предлага транзакции, сигурност, едновременност и управление на компонентите, които обслужва, и това позволява на разработчиците да се концентрират върху бизнес логиката на компонентите без да се губи излишно време. Също така в по-нова версия на системата може да се добави и JMS (Java Messaging Service), който би предоставил още много нови възможности за системата.

## 2.2 Apache Maven

Описание на технологията

Maven е автоматизиран инструмент за билдване на проекти, използван основно за проекти написани на Java. Maven е проект, чиято цел най – общо казано е да улесни писането и компилирането на проекти на Java. Целта е да има универсално средство за билдване на проекта, така че когато добавяш нови елементи към проекта, да е лесно и удобно за нов build. Конфигурацията за билда на проекта (т.е. нещо като build script) се намира във файл „project object model” (POM.xml). Едно от предимствата на Maven е конфигурацията на repository. В други подобни технологии трябва да се описва всяка една библиотека, от къде се взима, докато при Maven, има две хранилища – едно локално, което се намира на нашия компютър, и едно централно.  
И когато се компилира нещо за първи път и в локалното хранилище няма необходимите библиотеки, те се вземат от централното – в повечето случаи –автоматично. Свалят се необходимите библиотеки и проекта се компилира. При следващо компилиране, вече имате библиотеките, готови на разположение.

[](http://gutsev.com/wp-content/uploads/2015/01/Maven_central_repository.png)

*Фигура 4. Примерна конфигурация на хранилищата в Maven.[14]*

Компанията може така да си конфигурира хранилищата, че когато някой поиска да се включи в проекта, е нужно само да изтегли проекта от системата за контрол на версиите (SVN, GIT или др.), след това да стартира билда на проекта и всички артефакти се изтеглят при него автоматично. За да се направи това, организацията прави едно централно хранилище с всички необходими библиотеки (артефакти). На локалната станция на всеки от програмистите, Maven е настроен, да гледа в локалното repository и ако  там няма артефакти  –  се обръща към централното хранилище  на организацията. По този начин, дори да има нова версия на библиотека, тя се поставя в централното repository и се разпространява автоматично. Ако има артефакти,които не са публично достъпни, например на Oracle, не е нужно тези артефакти да се дават ръчно на всеки, а е необходимо само да се сложат в централното хранилище.

Защо използвам тази технология?

Мисля че обясненията по-горе са достатъчни, но накратко ще изкажа отново моите причини. Основните причини, поради които ползвам Maven, са защото улеснява много работата при разработка и пести страшно много време в разрешаване на зависимости към други библиотеки. И веднъж конфигуриран pom.xml файла е много удобен и позволява на хората които ще се включат към разработването на проекта да започнат работа моментално.

## 2.3 Jersey

Описание на технологията

Jersey е технология с отворен код и служи за реализирането на RESTful приложения. Първоначалната версия е на технологията е пусната на пазара през 2007 и до сега тази технология се е наложила на пазара като една от най-надежните. Целта на тази технология е да предостави набор от приложни програмни интерфейси (API), които да опростят разработването на уеб услуги по RESTful дизайн.

Защо използвам тази технология?

Освен че тази технология е актуална, надежна и популярна, тя също има три основни предимства пред другите подобни технологии: първата е, че предоставя стандартизирана нотация, с която лесно и бързо може да се разработи уеб услуга по RESTful дизайн. Втората е, че предоставя приложни програмни интерфейси (API) и за приложенията от тип „клиент“, които лесно да комуникират с дадена уеб услуга. И трето, предоставя библиотеки, с които лесно може да се интегрира със Spring, Guice, Apache Abdera и други.

## 2.4 Oracle Database

Описание на технологията

Oracle Database е база от данни, продукт на големия софуерен гигант „Oracle“. Базата от данни представлява колекция от логически свързани данни в конкретна предметна област, които са структурирани по определен начин. В първоначалния смисъл на понятието, използван в компютърната индустрия, базата от данни се състои от записи, подредени систематично така, че компютърна програма да може да извлича информация по зададени критерии. Системите за управление на базите от данни създават, обработват и поддържат точно определени структури от данни. Съществуват три типа организация на данните и връзките между тях (наричани модели на бази от данни): йерархичен (на англ. hierarchical model), мрежови (network model) и релационен (relational model), като така се различават йерархични бази от данни, мрежови бази от данни и релационни бази от данни. Oracle Database спада към третия тип база от данни - релационни. Релационният модел е най-популярен, при него данните се организират в таблици, между които се осъществяват връзки/съотношения (т.н. релации). Информатиката познава още 2 вида модели на бази от данни, различни от основните 3, а именно многомерен/многопространствен (multidimensional model) и обектен (object model). Таблиците се състоят от редове и колони. Понякога редовете се наричат записи, а колоните - полета.

Защо използвам тази технология?

Oracle Database е доминиращата на пазара система за управление на бази данни (DBMS), която се използва от големи и малки организации в цял свят.[5] И главната и единствена причина, поради която ползвам тази технология е, че тя е най-надежната релациона база от данни на пазара до момента.

## 2.5 Windows Presentation Foundation

Описание на технологията

Windows Presentation Foundation (WPF) е изцяло нова графична система за разработване на приложения за Windows операционна сиситема, която осигурява едновременно на разработчици и дизайнери – унифицирана платформа за създаване на богати медия приложения на базата на XAML езика. WPF е проектиран за .NET платформата, повлиян е от модерните технологии HTML и Flash и е хардуерно ускорен.

WPF е графична подсистема за рендериране на потребителски интерфейс в Windows-базирани приложения, която ползва XAML. Езикът е представен през 2006 г. заедно с WPF (Windows Presentation Foundation) в .NET 3.0. Основните предимства на XAML са, че разделя UI от бизнес логиката и че позволява едновременно да се работи върху логиката и интерфейса на една програма с различни инструменти (Visual Studio и Microsoft Expression Blend).

Елементите на XAML са директно свързани с Common Language Runtime (CLR) обекти, а атрибутите – с Common Language Runtime (CLR) свойства и събития върху тези обекти. XAML файловете могат да бъдат редактирани с инструменти за визуален дизайн като Microsoft Expression Blend и Microsoft Visual Studio, както и със стандартни текстови редактори, кодови редактори като XAMLPad или графични редактори като Vector Architect.

Защо използвам тази технология?

Изполвам тази технология защото тя е една от най-добрите, лесна и надеждна технология за разработка на приложения за операционната система Windows. А също така, едно от предимстватана на WPF е, че използва XAML. Всички графични елементи в XAML са Direct3D апликации (използват видео картата за рендериране), което води до висококачествен и разнообразен потребителски интерфейс. Освен това графиките са вектор-базирани, което позволява оразмеряване на обектите без загуби в качеството им. XAML поддържа множество от функции, като анимации с таймери или свързани с действия на потребителя, видео и аудио във всички формати, които операционната система поддържа, както и връзки между тези функции.

## 2.6 Ninject

Описание на технологията

Използваният в текущото приложение контейнер е Ninject. Това е лека платформа за инжекция на зависимости, предназначена за ползване от .NET приложения. Спомага разделянето на приложенията на слабо свързани, силно кохезни модули, съединени отново по изключително гъвкав начин. Обикновено се намира в собствено assembly, така че приложението не се обвързва със зависимости към Ninject. Тази платформа е една от първите по рода си, предлагащи контекстно разрешаване на зависимости – на база на рънтайм контекста, бил той http или друг, може да се опишат случаи с различни имплементации.

Защо използвам тази технология?

В едно приложение винаги е добре да се използва някаква технология за инжекция на зависимости. В .NET средите това е една от най-използваните и лесно конфигурируеми технологи за тази цел. А най-голямата полза от използването на тази технология е, че спомага разделянето на приложенията на слабо свързани, силно кохезни модули.

## 2.7 Android

Описание на технологията

Android е операционна система на Google Inc. за мобилни устройства. Тя е създадена, поддържана и развивана първоначално от Android Inc., която е купена от Google Inc. през 2005. Операционната система Android използва в основата си модифицирана версия на ядрото на Linux.

За развитието на Android се грижат голям брой софтуерни разработчици, които създават така наречените „apps“ (Applications) – малки приложения, които разширяват функционалността на системата. Приложенията могат да бъдат сваляни от различни сайтове в Интернет или от големи онлайн магазини като Goolge Play - магазинът на Google. По данни към януари 2011 г. за Android има над 200 000 приложения. Приложенията се пишат предимно на Java, Python или Ruby. За разработването на приложения за операционната система Android се изпозлава Andoid SDK.

Android SDK (Software development kit) представлява набор от средства

и иструменти за разработване на приложения за платформата Android. Това множество от програмни средства предоставя на програмиста API библиотеки и инструменти, които да облекчат и улеснят изграждането, тестването и отстраняването на грешки в написаните от него приложения. Android SDK е предназначена за вграждане и използване като част от средата за програмиране (IDE), която разработчиците използват за писането и компилирането на технически код.

Защо използвам тази технология?

Отговора на този въпрос е очевиден. За последното тримесечие на 2010 г. Android вече е определена като най-продаваната платформа за смартфони в света.[6] Популярността на системата непрекъснато расте, и през второто тримесечие на 2014 г. например 86% от продадените смартфони са с инсталирана операционната система Android. По последни данни от март 2017 година, Android вече е най-използваната операционна система, като води пред Microsoft Corporation и тяхната операционна система Windows с нищожна разлика от 0,02%. [7]

## 2.8 Google Cloud Messaging Server

Описание на технологията

Google Cloud Messaging e облачна услуга предоставяна от Google. Това е технология за изпращане на съобщения между различни платформи, която позволява надеждно да се изпращат съобщения в реално време, без никакви разходи. С тази технология могат да се изпращат съобщения от тип известия и съобщения, които съдържат някакви данни, които се обработват от приложението.

Едно от главните предимства от услугата Google Cloud Messaging е, че позволява масово да се изпращат съобщения до всички клиентски приложения и да се организира между тях обмен на данни. Обслужването на потребителските профили в приложенията може да става с помощта на технологията Google Authentication.

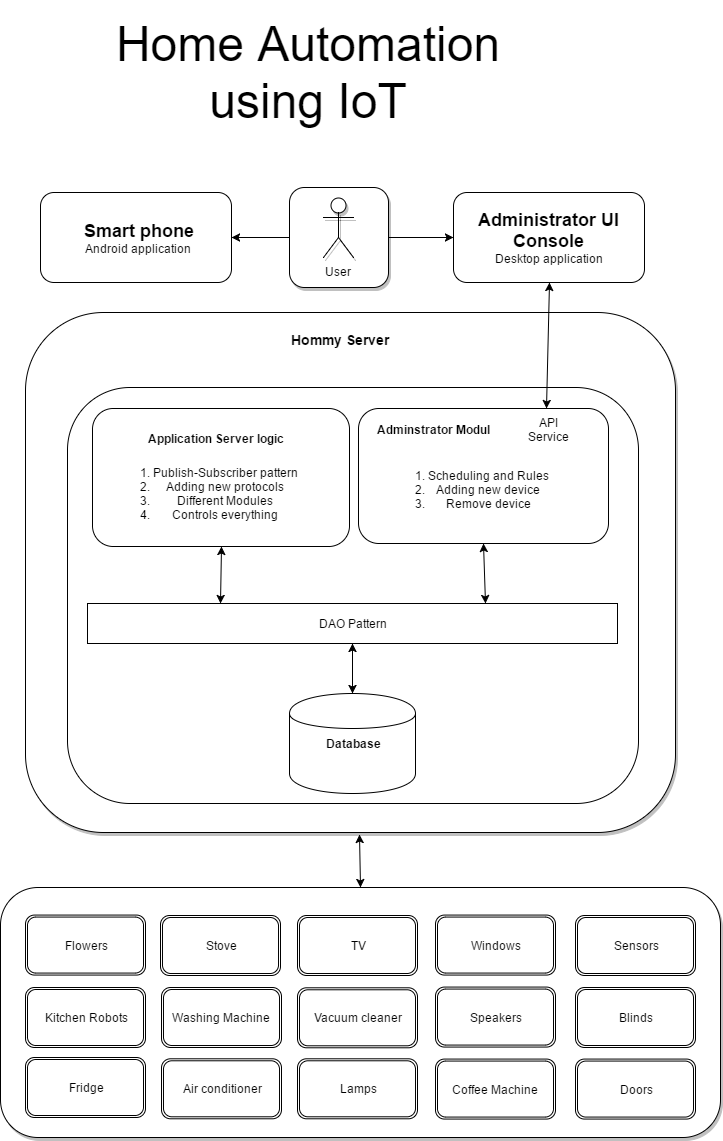
Защо използвам тази технология?

Причината да изпозлвам тази технология е защото тя е единствената толкова добра, надежна и лесна за използване технология в момента на пазара. Ето и няколко статистически данни които доказват това. По данни на Google към 2015 година 60 процента от топ приложенията в Google Play използват Google Cloud Messaging.[8] Също така на ден се изпращат 17 млрд. съобщения през Google Cloud Messaging. Като по последни данни тези цифри се увеличават с всеки изминал ден.

# Глава 3. Проектиране на мултиплатформено IoT базирано решение.

## 3.1. Проектиране на сървър приложението

Крайните потребители на тази система ще са всички хора, които притежават „умни“ устройства в своя дом. Системата трябва да бъде бърза и надеждна. Също така трябва да притежава красив и опростен потребителски интерфейс и да бъде лесно конфигурируема.



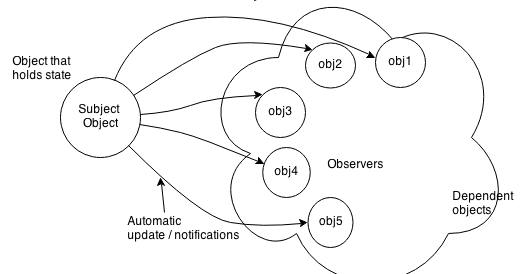
Фигура 5. Архитектурен модел на цялата система “Хоуми“

Нужно е да се направи много добра и лесна за разширяване софтуерна архитектура. Също така системата трябва да бъде лесна за поддържане. На фигура 5 може да видите цялостната архитектура на системата.

Най-голямата и най – важната част е централният сървър. Той ще приема и обработва заявките от всички устройства и ще изпраща команди към тях. Системата ще имплементира Data Access Object (DAO) шаблона за връзка с базата от данни. Този шаблон се използва за отделяне на начина на достъп до базата от данни (ниско ниво) от бизнес логиката на програмата (високо ниво). Този шаблон се състои от три основни компонента.

* Интерфейси – те дефинират набор от стандартни операции, които да се изпълняват от конкретните класове.
* Конкретни класове – те имплементират методите от горния интерфейс. Този клас е отговорен за вземането на данните от базата.
* Модели – Тези обекти са обикновени класове (POJO обект), които дефинират стандартни аксесори за достъп и модификация (getter и setter методи).

Едно от главните изисквания на системата е тя да изпраща съобщения към много устройства, за да се постигне това е нужно да реализира добър и гъвкав начин, по който да се разбере към кои точно устройства да се изпрати дадено съобщение. Също така този софтуер е нужно да бъде разбираем от разработчиците, които ще го надграждат и поддържат. За тази цел е нужно да се имплементира поведенчески шаблон Наблюдател (на английски: Observer).



*Фигура 6. Схема на работа на шаблона Наблюдател (Observer)*

Различните обекти от схемата представляват всички свързани устройства, които ще бъдат абонирани за различни теми. И така, когато има промяна по някоя тема, всички устройства, които са абонирани за нея ще получат известие за тази промяна, разбира се, ако сървърът сметне за необходимо, на база всички конфигурации от страна на потребителя. Например ако има промяна по темата температура, сървърът ще обработи и анализира тази промяна и ще разбере към кои устройства да изпрати команди или известия точно по този начин.

### 3.1.1 Проектиране на базата от данни (БД)

Базите данни предлагат много високо ниво на достъпност, в което

промяна, добавяне или изтриване на нови елементи става много лесно и

бързо.Това се случва чрез създаване на „n” на брой таблици, които могат да

комуникират помежду си и да подават данните си на други програми.

За целите на нашата система е необходима наистина много надежна и бърза база от данни. Поради точно тази причина за реализирането на базата от данни е избрана Oracle Database 12с. И понеже обработката на данните трябва да се случва наистина бързо, още на ниво база от данни е нужно да се добавят индекси на колоните в таблиците, където се очаква да има много записи и ще бъде нужно тези записи да се достъпват много бързо. Най-често индекси се поставят на колоните, където се записва идентификационният номер за всеки запис в таблицата. Така търсенето в тази таблица е сравнително по-бързо. Цялата тази оптимизация се случва в самата база от данни, като всяка една база от данни оптимизира търсенето в таблиците по свои алгоритми.

За да може системата да притежава приложение за администратори, които да извършват някакви конфигурации по системата, е нужно да съхраняваме техните данни в базата. За целта е нужно да се направи таблица с четири колони. Първата колона ще съхранява идентификационен номер на потребителя. Втората колона е за потребителското име, а третата е за паролата на потребителя. Добре е паролата на потребителя да се съхранява в хеширан формат, за да бъде трудна за разбиране. И последната колона ще се съхранява имейла на потребителя.

За съхраняване на всички свързани устройства към системата е нужно да се направи таблица в базата от данни. Тази таблица е нужно да има четири колони, една за идентификационен номер на устройството, втора за името на устройството, още една колона за протокола, по който комуникира това устройство и последната колона е за “authentication token” за това устройство. По принцип не е добра идея “authentication token” да се пази в базата от данни, но в този случай е по-добре да се реализира така, защото това ще доведе до повишаване на бързото действието на системата. За последната колона е важно да се предвиди дължината на “authentication token”, защото той може да се увеличи след време.

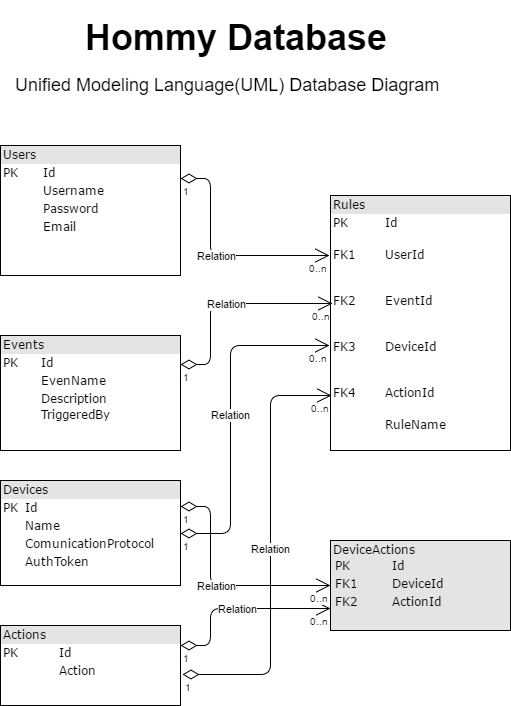
Нужно е да се направи и таблица за съхраняването на всички събития, за които системата ще следи. Ще са нужни отново четири колони за създаването на тази таблица. Първата както обикновено ще съхранява идентификационния номер на събитието. Втората колона ще съдържа името на събитието. Третата ще съхранява кратко описание за това събитие. А последната колона ще съхранява името на действието което ще задейства това събитие. Като например температура, така всеки път когато има промяна по тази тема, системата ще проверява за това събитие и дали е нужно да го изпълни.

Още едно от нещата, които трябва да се съхраняват в базата от данни са действията, които могат да извършват различните устройства. Всяко едно устройство може да извършва различни действия, но има и много действия, които се припокриват от всички устройства, като включване и изключване. Затова е нужно да се направи една таблица за съхранение на действия, които могат да изпълняват различните устройства. Причината да е отделна таблица е защото всички устройства могат да бъдат включвани и изключвани и не е нужно да се пази тази информация стотици пъти, а само веднъж. Тази таблица ще има само две колони. Една за идентификационен номер на действието и друга за името на самото действие. Но, за да разбере системата кои действия може да изпълнява дадено устройство, е нужно да се направи свързваща таблица между таблицата за устройства и таблицата за действия. Тази таблица ще се състои от три колкони. Първата колона ще съхранява идентификационен номер за този запис, който ще е уникално число. Втората ще съхранява идентификационния номер на устройството, а третата колона ще съхранява идентификационен номер на действието. Така системата разбира много лесно за дадено устройство с някакъв идентификационен номер - кои действия могат се да изпълняват от него.

И последната най-голяма таблица, която е нужно да се реализира е таблицата за правилата. Тя ще се състои от шест колони. Първата колона отново е за за идентификационен номер на правилото. Втората колона съхранява идентификационния номер на потребителя, който е създал това правило. Третата колона съхранява идентификационния номер на събитието, което ще задейства това правило. Четвъртата колона ще съхранява идентификационния номер на устройството към което трябва да се прати команда. В следващата колона се съхранява идентификационния номер на действието, което трябва да изпълни устройството. И в последната колона се съхранява името на правилото.

За всички таблици, които съдържат колона с идентификационен номер, трябва в тази колона да се съдържат само уникални номера. За тази цел е нужно всеки път, когато нов запис постъпи в дадена таблица да се записва с нов уникални идентификационен номер. Най-лесно може да се постигне това в Oracle Database чрез създаването на последователности (sequence) за всяка таблица.

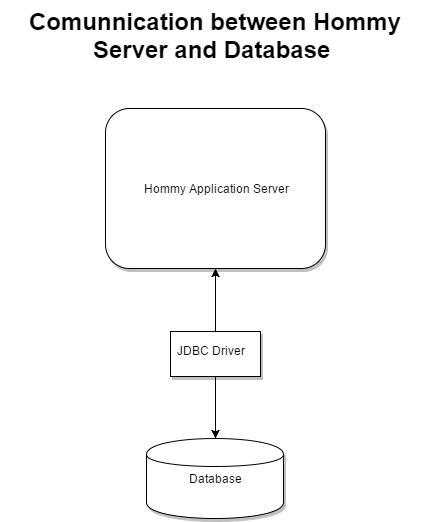
Последователностите генерират ново число, което е по-голямо от предишното генерирано число. А, за да се разбере кога да се генерира ново число, е нужно да се имплементират тригери. Тригерите изпълняват някакво действие в базата от данни, когато настъпи някакво събитие, като добавянето на нов ред в дадена таблица. Това, което трябва да се направи, е да се задейства тригер, който да увеличи последователност (sequence) от дадена таблица, когато се добавя нов запис в тази таблица. По този начин ще се гарантират уникални стойности за дадена колона, която съхранява идентификационен номер. На следващата фигура ще намерите схема на релациите (ER диаграма) между таблиците в базата от данни



*Фигура 7. Схема на релациите между таблиците в базата от данни*

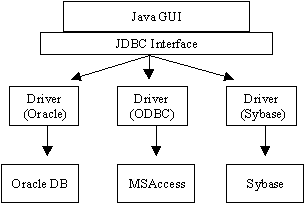
### 3.1.2 Комуникация между сървър приложението и базата от данни

За да се реализира добър архитектурен дизайн за комуникацията с базата от данни, е препоръчително използването на шаблона Data Access Object (DAO). Това ще доведе до абстракция от начина, по който сървър приложението комуникира с базата и ще улесни добавянето на нова база от данни която да се поддържа от системата.



*Фигура 8. Схемата за комуникацията между базата от данни и системата*

На фигурата по-горе е изобразена схемата за комуникацията между базата от данни и системата. Връзката се осъществява благодарение на JDBC драйвер. JDBC (Java Database Connectivity) на Java Enterprise е първият междуплатформен програмен интерфейс за използване на бази от данни. Може да се каже, че JDBC е първият стандартизиран метод за интегриране на Java с базите от данни. В проектирането на JDBC са използвани основни абстракции и методи от ODBC. Идеята за базиране на JDBC върху ODBC идва от това, че ODBC е популярен сред независимите разпространители на софтуер и реализирането и използването на JDBC ще бъде по-лесно за хора, работили с ODBC. Най-общо казано архитектурата на едно приложение, използващо JDBC може да бъде видяна на фигурата, по-долу:

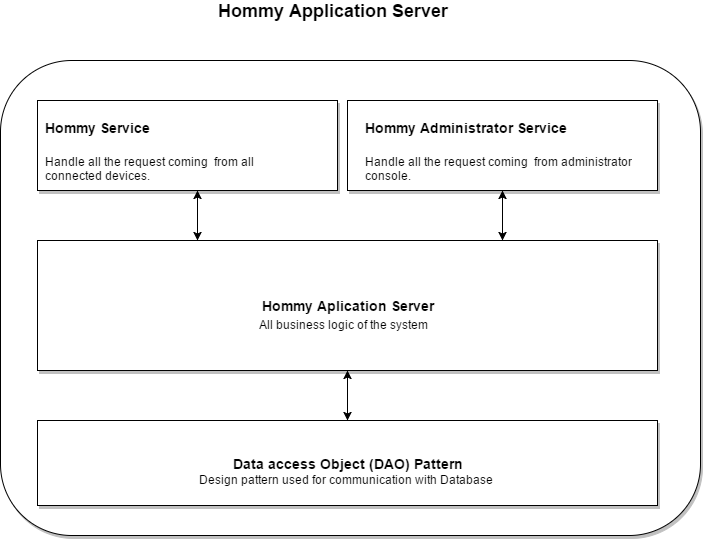


*Фигура 9. Архитектура на приложение, използващо JDBC.[15]*

На фигурата може да забележите, че благодарение на тази технология, добавянето на нова база от данни, с която да работи приложението е много лесно. При проектирането на един софтуер е от основно значение системата да бъде лесно надграждана, чрез добавянето на нови функционалности. Подържането на друга база от данни също спада в тази категория, така че JDBC ни позволява тази гъвкавост.

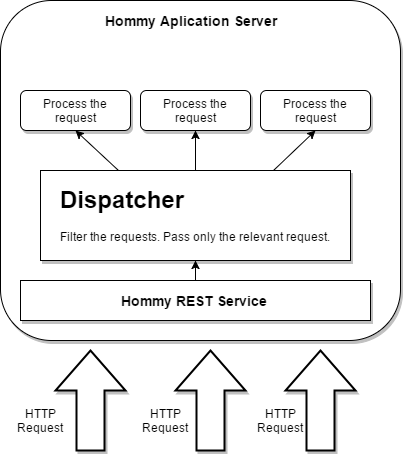
### 3.1.3 Архитектурен дизайн на сървър приложението

Съществуват две главни функционалности за реализиране на сървър приложението. Нужно е да се имплементират две услуги, които да работят с другите компоненти на системата. Едната услуга е за комуникация с приложението за адмистратори. Тази услуга трябва да предоставя набор от стандартни операции, като добавяне и изтриване на събития, устройства и правила. Също така, то трябва да предоставя и операции за извличане на данни от базата и изпращането им до приложението за администратори. Другата услуга, която е нужно да се реализира ще служи за приемане на данни от всички свързани устройства към системата. За да могат всички устройства да комуникират със сървър приложението е нужно да се установи единен формат на съобщенията. Форматът на съобщенията може да бъде XML или JSON, като вече все по-предпочитан и използван е JSON форматът. Съобщенията, които се изпращат към услугата трябва да носят достатъчна информация за това кой и каква информация изпраща. Нужно е съобщението да съдържа информация за това, за коя тема се изпраща съобщението. Също така и нова и стара стойност, ако е налична тази информация. И текст с причината за изпращане на това съобщение. Ето и един пример за съобщение направен в разбираем за човека вариант - устройството климатик увеличи стайната температурата от 23 на 25 градуса. След получаване на това съобщение, сървър приложението трябва да прецени дали е нужно да извърши някакво действие и ако да, какво е необходимо да направи точно. Това се решава от бизнес логиката на приложенито, която благодарение на имплемантацията на шаблона Data Access Object (DAO) използва абстрактен API модел за достъп до базата от данни, както можете да видите на фигура 10.



*Фигура 10. Архитектура на сървър приложението*

Изискванията към сървър приложението на системата са доста високи. Едно от най-големите изисквания към системата е тя да бъде достъпна. Това означава, че тя трябва да бъде на разположение и устройствата да могат да комуникират с нея, а ако по-някаква причина тя не е достъпна, е нужно потребителят да получи известие за това. Друго голямо изискване е системата да успява да получи, обработи и да изпълни дадените инструкции на всички устройства. Тук не можем да говорим за голямо количество от данни, понеже тя ще комуникира единствено със свързаните утройства към системата, а те не би трябвало да са много. Това е така за момента, но след време тези данни ще се увеличават значително, трябва системата да бъде готова и за това. За тази цел е необходимо да се проектира решение за бърза обработка на данните, които идват непрестанно от всички устройства.



*Фигура 11. Архитектура на Диспетчера.*

На фигурата по-горе е показано решение за този проблем. В горната схема е показана работата на диспетчера. Той проверява дали дадената заявка е валидна, след това проверява дали заявката идва от познато устройство на системата и също така следи за хакерски атаки от типа - отказ на услуга (DoS атака). Най-общо казано диспечера филтрира всички заявки, които се приемат от системата и така оставя само валидните заявки да се обработят от системата.

Следващото по-приоритет изискване към системата е тя да може да бъде лесно разширявана, и тук се няма предвид просто добавянете на нови функционалности. Системи от този тип е препоръчително да се изпълняват на някоя облачна система и потребителя да може да я достъпва през своя компютър чрез потребителско име и парола. Това естествено веднага повдига въпроса със сигурноста, който е един от най-големите проблеми на “Интернет на нещата” (IoT). Проблем, който спира развитието му. Но на по-късен етап, той ще бъде решен и мигрирането към облачните системи трябва да бъде лесно и бързо постижимо.

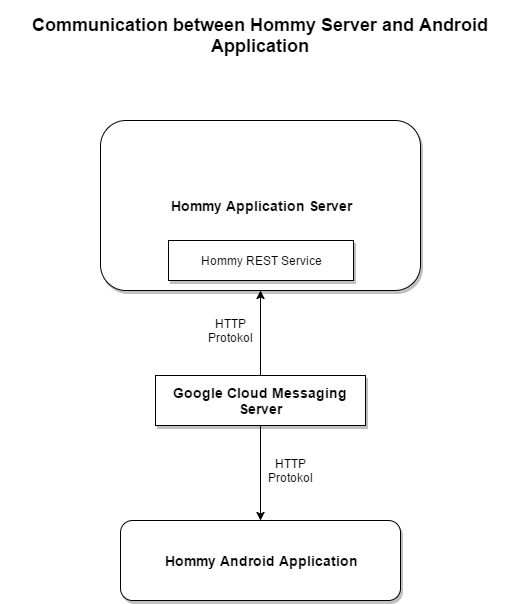
## 3.2 Проектиране на Андроид приложението

### 3.2.1 Ахитектурен дизайн на Андроид приложението

Архитектурният дизайн за приложенето трябва да се придържа към стандартите за проектиране на Android приложения. Нужно е да се създадат нужните Активити за работа на приложението и да се добавят нужните UI елементи към layout файла на сътветните Активита. Активитито е екранът, на който се изобразяват всичките видими елементи, чрез които потребителят комуникира с приложението и обратното. Всички класове, xml файлове и ресурси да бъдат поставени в правилните папки.

### 3.2.2 Комуникация между Андроид приложението и сървър приложението

Главните изисквания към системата са две, първото е тя да известява потребителя за всичко важно, което се случва в неговия дом, и второто е да може да изпраща съобщения до сървър приложението. В свят, където всички притежават мобилен телефон с операционна система, известяването на потребителя е лесно да се постигне именно чрез мобилните телефони. Поради тази причина е небходимо да се имплементира мобилно приложение, което да приема съобщения, които са важни и е нобходимо потребителя да бъде уведомен за тях. И понеже по-голямата част от света използва мобилните телефони с операционна система Android, приложението трябва да бъде реализирано първо за Android. Комуникацията между сървър приложението и мобилното приложение трябва да бъде много добра и винаги да се получава обратен отговор. Това означава, че ако сървър приложението изпрати съобщение до мобилното приложение, то второто трябва да получи обратен отговор дали съобщението е било получено успешно. Ако то не се изпратило успешно, сървърът може да пробва да изпрати съобщение и ако то отново не се изпрати, сървърът да бъде наясно, че това устройство не може да се достъпи в момента.



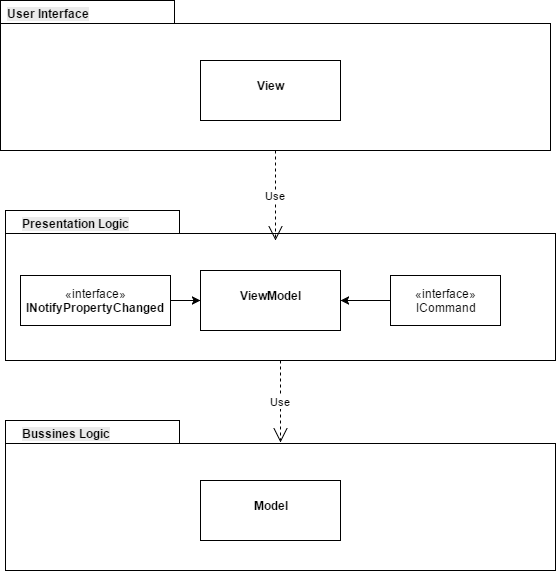
*Фигура 12. Комуникация между сървър приложението и Android приложението*

За да се подсигури добрата комуникация, е препоръчително да се използва Google Cloud Messaging (Firebase). С използването на тази технология се повишава сигурността и надежността на системата. Чрез тази технология, лесно можем да верифицираме дали дадено съобщение е било изпратено успешно. Android приложението ще изпраща съобщения към Google Cloud Messaging (Firebase), а то своя стана ще изпраща съобщението до сървър приложението на системата. За верификация на устройствата се използва “authentication token”, който се запаметява в базата от данни на системата, когато устройството се свързва за първи път към системата. Обратният сценарий на комуникация (от сървър приложението към Android приложението) ще се извършва по същия начин. Също така Google Cloud Messaging поддържа и XMPP протокол, което е голям плус за нашата система, ако след време се добави поддръжката и на този протокол.

## 3.3 Проектиране на приложението за администратори

### 3.3.1 Архитектурен дизайн на приложението за администратори

Един от големите плюсове на тази система е възможността тя да бъде конфигурирана спрямо нуждите на клиента. И тази конфигурация да се изпълнява не от някой външен човек, а от самия потребител. Точно това отличава тази система от всички други на пазара в момента. Тя притежава интуитивен дизайн и крайният потребител може да конфигурира основните начини на работа на системата. Естествено, това, както плюсове, носи и големи рискове със себе си. Възможно е потребителят да обърка доста неща и системата да не работи правилно, но това е грешка на потребителя и ако преди това той прочете внимателно упътването за употреба, би трябвало да няма никакви проблеми с използването на системата. За да бъде налице тази функционалност трябва да се имплементира потребителски интерфейс за администратори. Достъп до този интерфейс ще имат само лица, които могат да конфигурират системата, така наречените администратори на системата. Ето един пример: в едно семейство член на семейството е закупил продукта и регистрира себе си като администратор, така само той има право да конфигурира системата, а всички останали членове на семейството, гости, роднини или съседи нямат достъп до системата.В следващите версии на приложението може да се добави опция за регистриране и друг член на семейство като администратор.

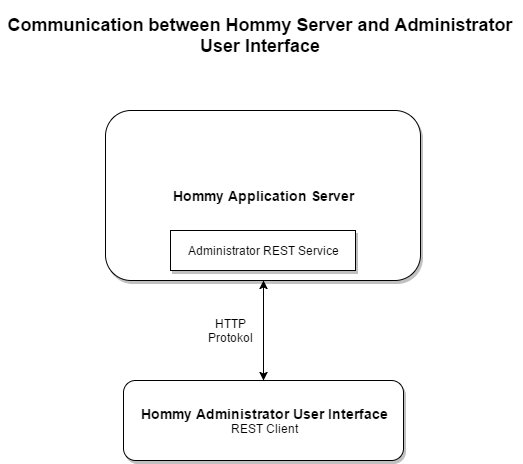


Фигура 13. Архитектура на шаблона Model-View-ViewModel (MVVM).

За реализацията на приложението за администратори е избрана операционата система Windows. Така всеки клиент ще може да инсталира приложенето на личния си компютър. Широко предпочитана технология за разработване на приложения за Windows операционната система е Windows Presentation Foundation (WPF). За проектирането и реализирането на Windows Presentation Foundation (WPF) е широко прието да се изпозлва архитектурния модел - Model-View-ViewModel (MVVM).

Model-View-ViewModel (MVVM)  е архитектурен шаблон за дизайн в програмирането основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение. На фигура 11 може да видите схема на архитектурния шаблона Model-View-ViewModel (MVVM).[9]

### 3.3.2 Комуникация между сървър приложението и приложението за администратори



*Фигура 14. Комуникация между сървър приложението и Администраторския панел.*

Комуникацията между сървър приложението и потребителския интерфейс се извършва през HTTP протокол. В този сценарий сървър приложението и потребителския интерфейс изпълняват така наречената сървър – клиент архитектура.

Моделът клиент/сървър е в основата на днешните разпределителни системи. Той е отговор на ограниченията типични за традиционния модел на централизирани ресурси, където главният компютър предоставя достъп до бази от данни чрез множество терминали. Клиент/сървър моделът е отговор и на архитектурата на файловите сървъри в локални мрежи, където системи комуникират с файл сървър, неразполагащ с мощен процесор. Клиентът е програмата, очакваща да получи услуга от друга програма. Той взаимодейства с потребителя чрез клавиатурата, дисплея или друго входно/изходно устройство. Клиентът няма директни отговорности към достъпа до данни. Той само изпраща заявки до сървъра и показва върнатите резултати на екрана. Ето защо клиентската машина може да бъде оптимизирана за своята работа. Например, тя няма да има нужда от голямо дисково пространство, за чиято сметка може да подобри възможностите на графичните си устройства. Сървърът е системата, която предоставя набор от услуги на клиента. Той има за задача да приема и обработва клиентските заявки и да връща отговора обратно. Клиентът и сървърът могат да са на един и същ компютър или на различни компютри, свързани в мрежа. Мрежата прави възможна отдалечената клиент/сървър комуникация. Този тип архитектура ни позволява по-голяма гъвкавост, защото на по-късен етап може да се дабавят още клиенти, които да ползва същия сървър. Потребителският интерфейс за администратори е нужно да има опции за логване на администратор, добавяне и изтриване на устройства, събития и правила. За по-голяма сигурност на потребителите техните пароли трябва да бъдат хеширани и записвани така в базата от данни. А когато потребителя се опита да се логне в системата, тя трябва да хешира паролата и да я сравни с хешираната парола в базата от данни, и ако са еднакви, тогава да извърши успешна автентикация.

# Глава 4. Програмна реализация

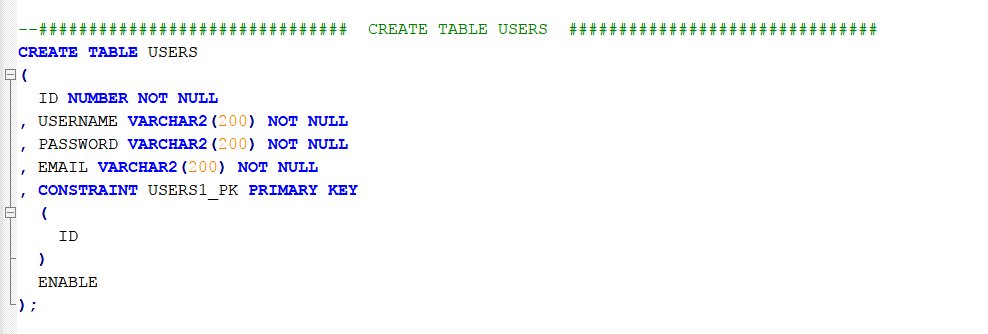
## 4.1. Програмна реализация на сървър приложението

### 4.1.1 Реализация на базата от данни

За реализацията на базата от данни е използвана Oracle Database. За създаването на базата от данни са имплементирани скриптове, които се изпълняват преди инсталирането на програмата. За написването на тези скриптове се използва езика PL-SQL. Този език е разработен от Oracle Corporation  и е внедрен във всички продукти на Oracle за управление на база от данни. За реализацията на базата от данни на тази система са създадени таблици, индекси, тригери, последователности (sequence). Индексите се добавят към избрани колони от таблицата, като тяхната цел е да кешират данните от тази колона, за да повишат търсенето точно от тази колона. Тригерите спомагат за запазването на интегритет на базата данни. Активират се при настъпване на определено събитие върху таблицата, с която са асоциирани, като това събитие е свързано с промяна на данните в нея. Тоест, тригерът се „активира“ (fire), когато настъпи събитие от тип  INSERT, UPDATE, DELETE- това са трите операции, при които се променят данните в една таблица. Тригерите много приличат на съхранени процедури, но за разлика от тях не могат да бъдат извиквани собственоръчно или пък да се извикват един друг.

„Sequence“ в базата от данни са нещо като числови редици (или казано по друг начин - аритметична прогресия), които увеличават своята стойност (най-често с една единица) всеки път, когато бъдат извикани с метода nextval.

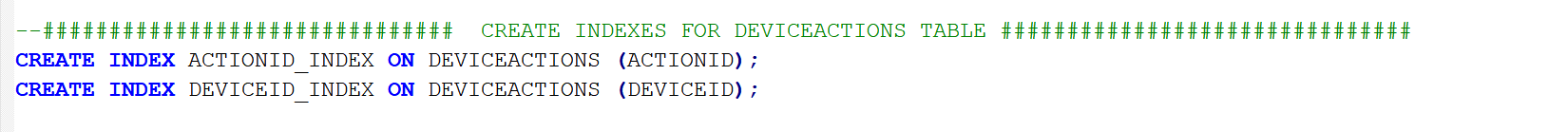
Методи за създаване на таблици:



*Фигура 15. PL-SQL скрипт който създава таблица Users в базата от данни.*

Тази таблица служи за съхраняване на администраторите на системата или по-просто казано - потребителите. Всеки адмистратор се идентифицира чрез идентификационния номер, който се увеличава автоматично при добавяне на нов администратор. Колоната Id e цяло число и също така е primary key за тази таблица. Другите колони съхраняват текстове и съответно са маркирани като VARCHAR, и не приемат нулеви стойности. Реализацията на другите таблици е имплементирана по-същия начин.

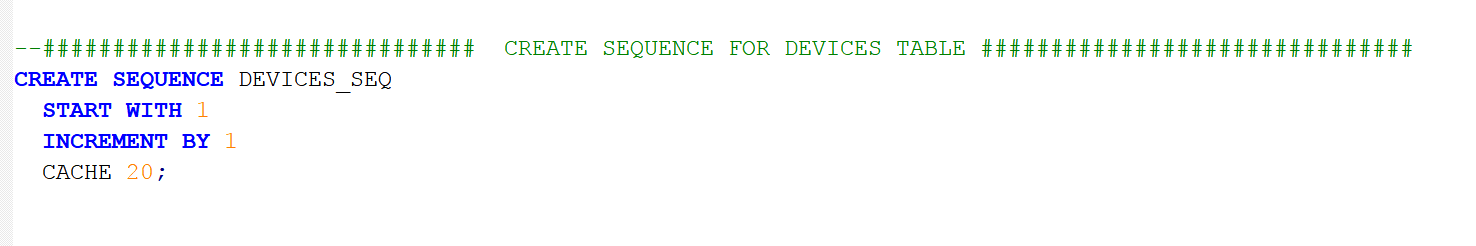
Методи за създаване на индекси:



*Фигура 16. PL-SQL скрипт който създава индекси за DeviceActions таблицата в базата от данни.*

Индексите спомагат за по-бързо търсене в базата от данни. Едно от главните изисквания към системата е тя да бъде бърза. Това е начин за постигане на бързо действие още на ниво база от данни.

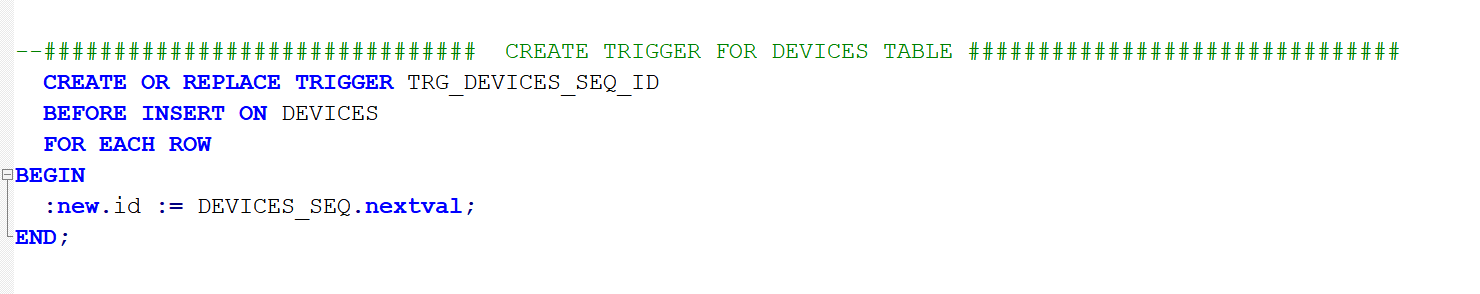
Методи за създаване на последователности (Sequences):



*Фигура 17. PL-SQL скрипт който създава Sequence за Devices таблицата в базата от данни.*

По този начин се създава Sequence за генериране на последователни числа, които се използват за идентификатори в таблиците. Така всеки път, когато се добавя ново устройство ще му се присвоява идентификационен номер, който ще бъде уникален в дадената таблица. Целта е всички записи в таблицата да имат уникален номер, чрез който да могат да бъдат идентифицирани от системата, когато е необходимо.

Методи за създаване на тригери:

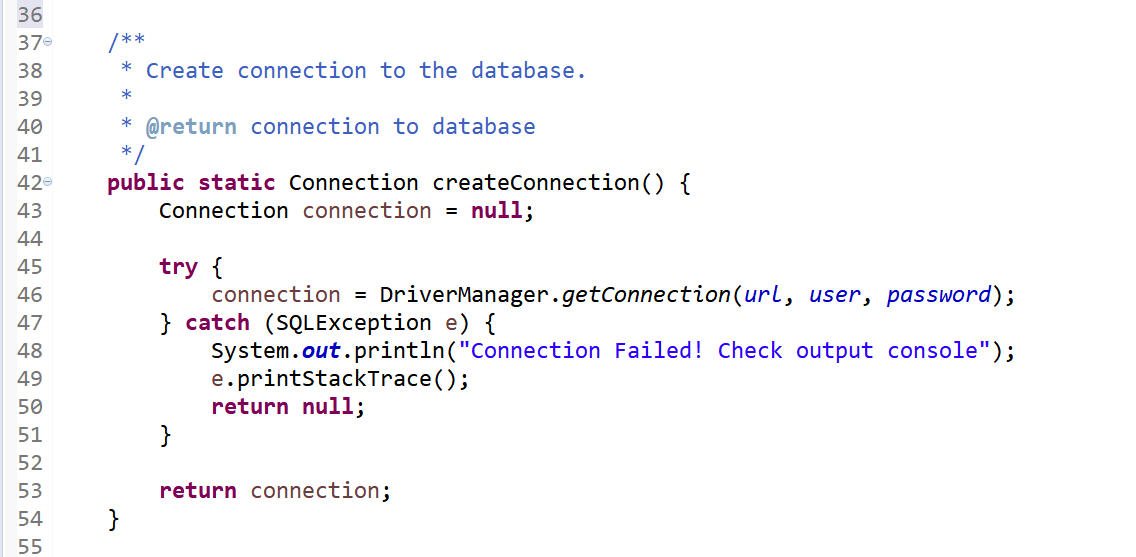


*Фигура 18. PL-SQL скрипт който създава тригер за Devices таблицата в базата от данни.*

По този начин се създават тригери в базата от данни. Тяхното предназначение е да се задействат при някакво събитие и да изпълнят някаква команда. Имплементираните тригери в системата служат за увеличаване на последователността с идентификационни номера за дадена таблица, те се задействат при добавяне на нови записи в таблицата.

### 4.1.2 Реализация на връзката между сървъра и базата от данни.

Връзката между сървъра и базата от данни се осъществява посредством JDBC driver. За да използваме драйвера, е необходимо той да бъде инициализиран. След това драйвера ни предоставя класа DriverManager със статични методи, които да използваме за различните операции. За инистанцирането на драйвъра и за създаването на връзка към базата от данни е имплементиран клас DatabaseManager, който инистанцира драйвера и предоставя метод за получаване на връзка към базата от данни (може да бъде видян на фигура 18).



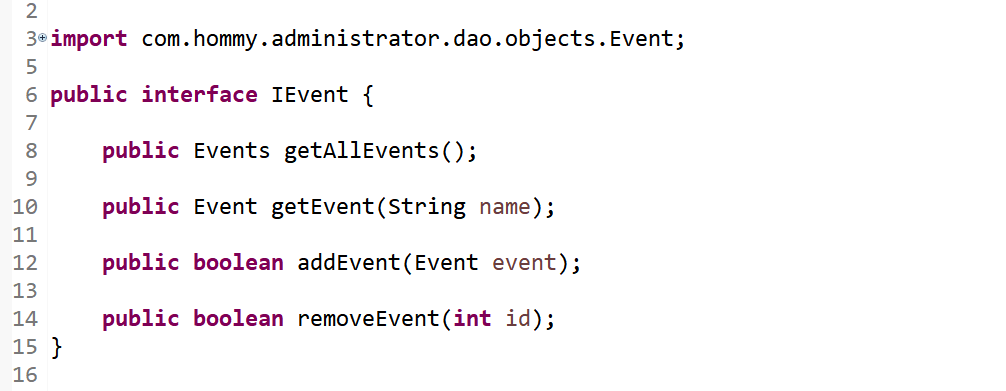
*Фигура 19. Метод от класа DatabaseManager, за получавана на връзка към базата от данни*

Едно от изискванията при проектирането беше да се реализира шаблона

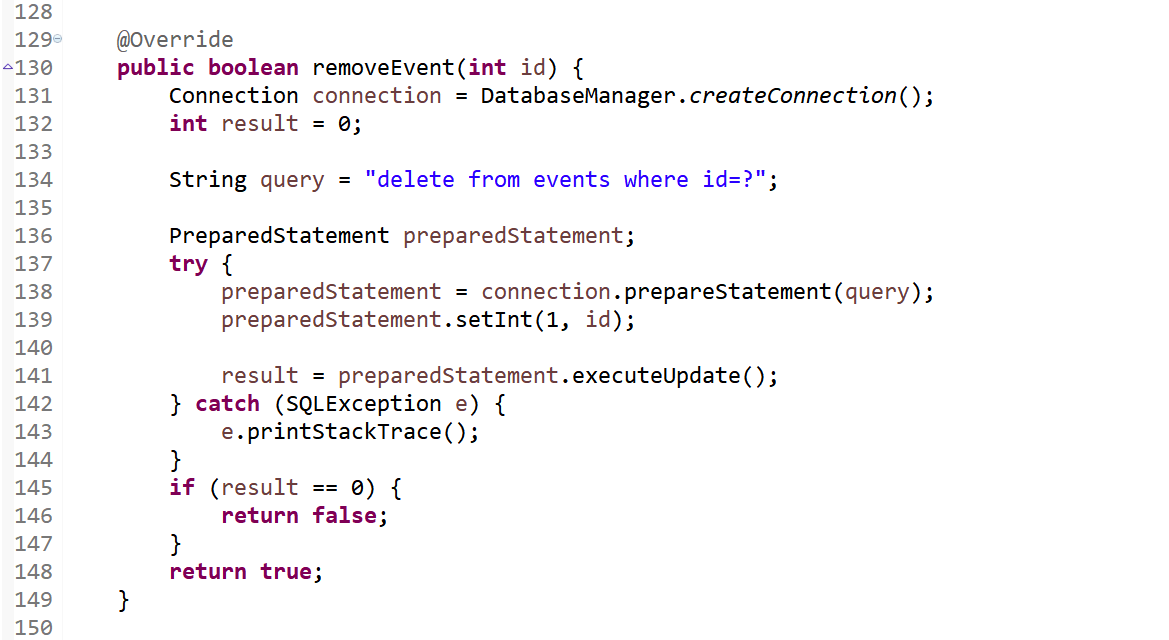
Data Access Object Pattern (DAO). На следващите три фигури може да се види реализацията на този шаблон, като за примера съм избрал имплементацията на събитията. На първата фигура е показан интерфейса IEvent, който дефинира набор от стандартни операции, които да се изпълняват за събитията. Като например вземане на всички събития от базата, вземана на някое конкретно събитие, добавяне и изтриване на събитие.

На следващата фигура е показан конкретния клас EventsOperation, който имплементира интерфейса IEvent. Той предоставя имплементация за всички методи от интерфейса. И на последената фигура е изобразен класа Event . Той

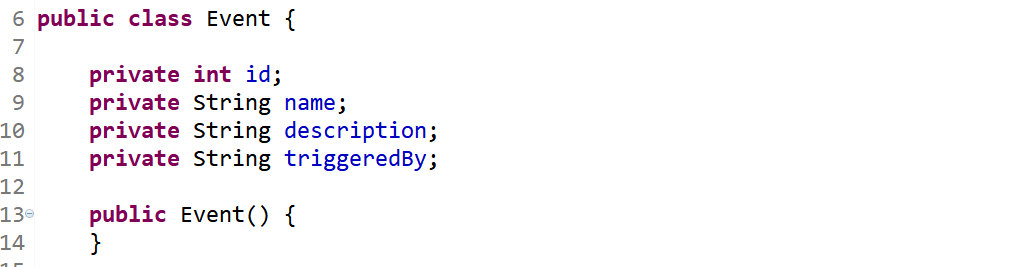
представлява обикновен клас (POJO обект), който дефинира стандартни аксесори за достъп и модификация (getter и setter методи).



*Фигура 20. Интерфейс IEvent*



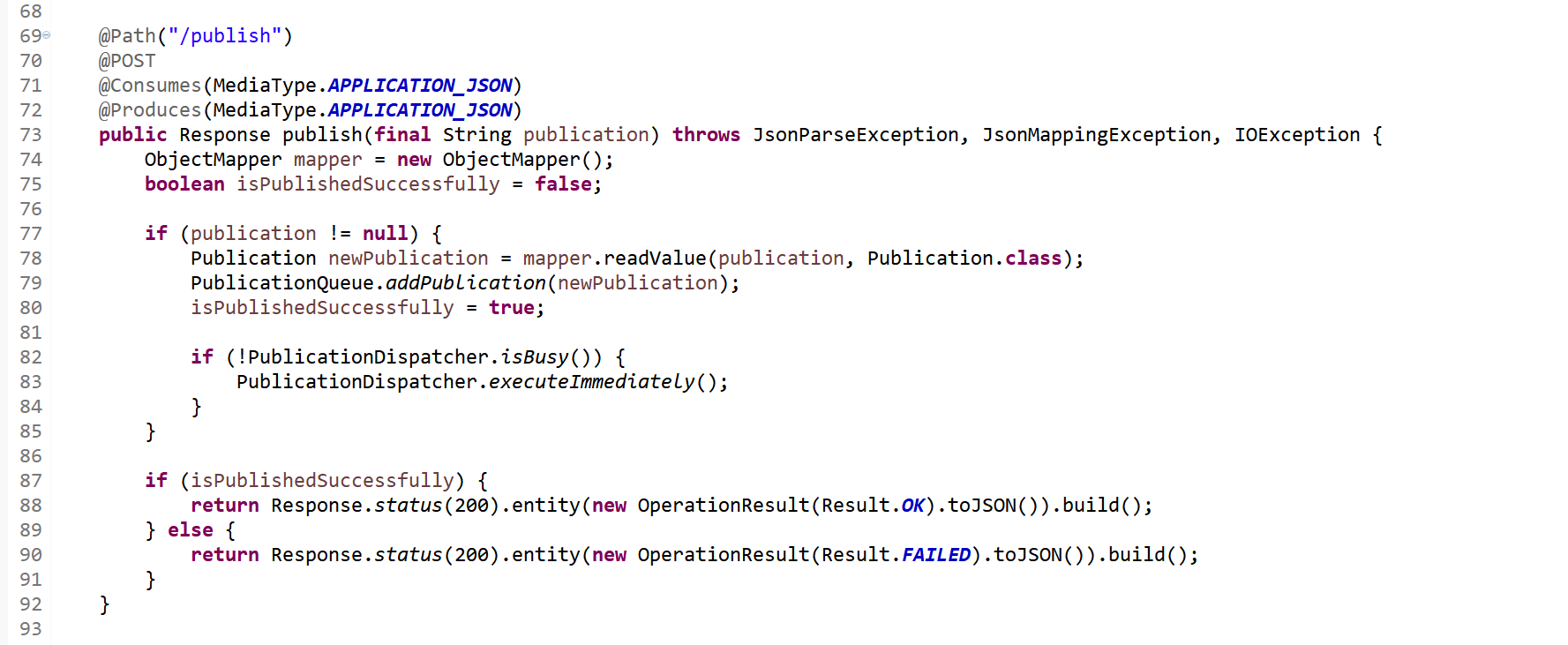
*Фигура 21. Метод от класа EventsOperation, който служи за изтриване на събитие от базата от данни.*



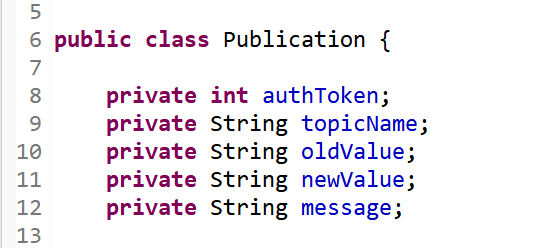
*Фигура 22. Модел на класа Event.*

### 4.1.3 Реализация на Услугите

За реализирането на услугите е използвана Java Jersey библиотека. Тази библиотека предоставя различни нотации, чрез които се задава пътя до метода, типа на метода (дали е POST или GET), какъв формат на данните приема и какъв връща.



*Фигура 23. Метод от класа HommyService, служи за приемане на съобщения от устройствата.*



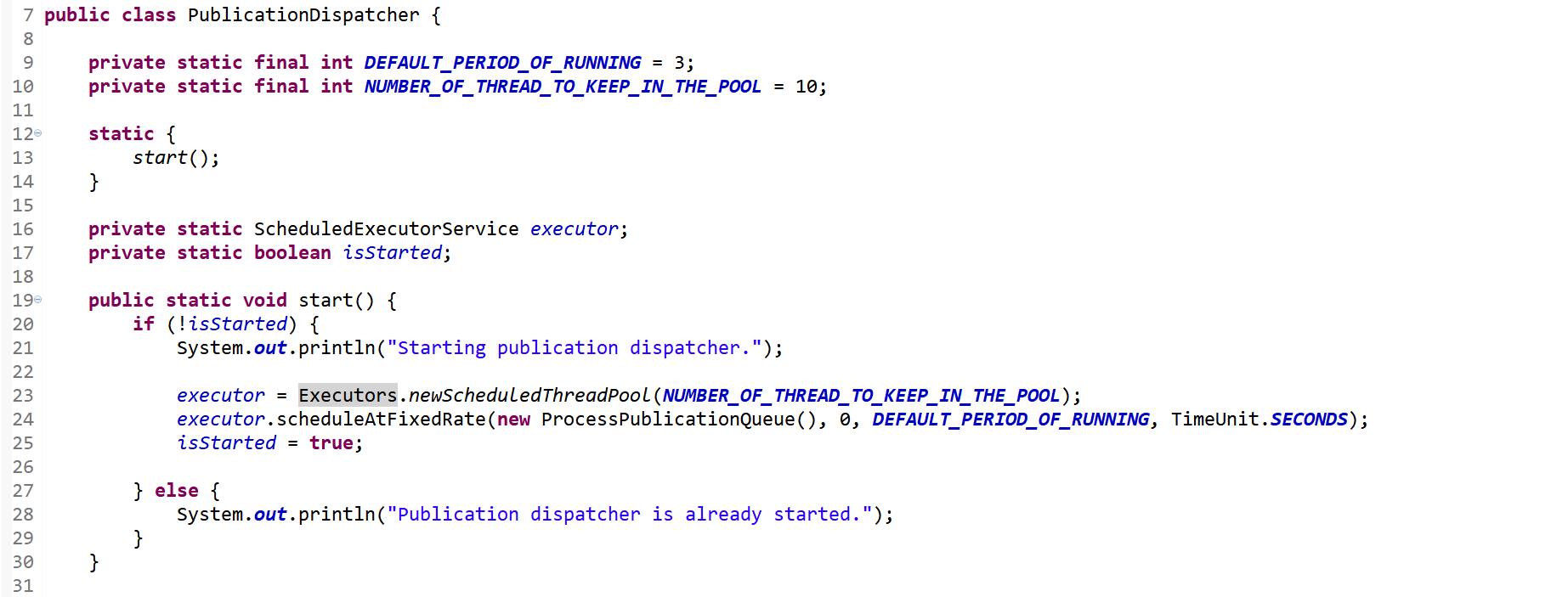
*Фигура 24. Полетата които съдържа класа Publication.*



*Фигура 25. JSON формат на съобщенията, които се приемат от сървъра.*

Тази услуга приема всички заявки, които идват от свързаните устройства. Понеже заявката се приема в JSON формат, първо е необходимо да се преобразува в разбираем за системата обект и след това да се обработи. Първо се проверява дали дадената заявка е валидна и след това я преобразуваме в обект от тип Publication (ще бъде обяснено по подробно в следващата глава). След като я преобразуваме я добавяме към опашката със заявки и ако добавянето е било успешно, връщаме позитивен отговор. Това е направено с цел оптимизация на системата и приемане на заявки максимално бързо. Идеята е системата да не губи време в обработката на някоя заявка и така да пропусне получаването на заявки, които идват от другите устройства.

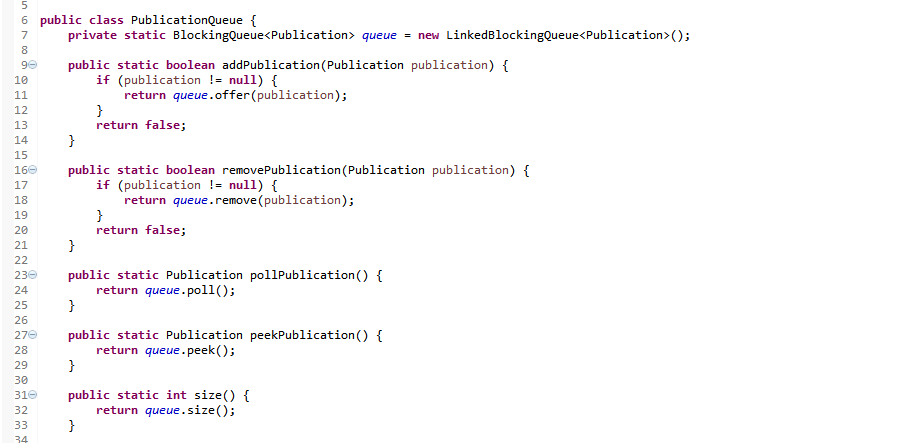
### 4.1.4 Реализация на Диспатчара на заявки



*Фигура 26. Метод от класа PublicationDispatchеr, който стартира диспатчера.*

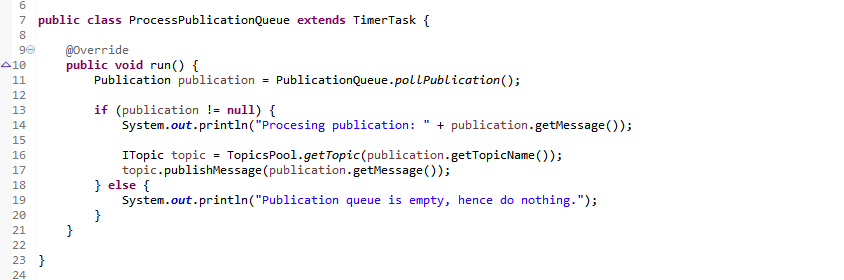
Когато се стартира системата, диспатчера започва своята работа. На фигура 20 може да се разгледа имлементацията на метода за стартиране на диспатчера. Класа ScheduledExecutorService  служи за настройване на задача, която да се изпълнява през определен период от време. Тази задача в този случай е ProcessPublicationQueue и тя се изпълнява през три секунди.

Класа ScheduledExecutorService е наличен в новата версия на Java, а сървър приложението работи с най-новата до момента версия на Java (Java 8). Класа PublicationDispatchеr съдържа още няколко метода, които са stop, isBusy и executeImmediately. Първият метод служи за спиране на диспатчера, вторият проверява колко зает е той и третият метод служи за изпълнение на някаква задача моментално, въпреки другите задачи.



*Фигура 27. Клас PublicationQueue.*

За оптимална работа на системата е имплементирана структура от данни „опашка“, в която се съхраняват всички заявки, които трябва да бъдат обработени. Те трябва да се обработят в реда, в който са постъпили затова се използва точно тази структура от данни. Използва се класа LinkedBlockingQueue, защото е безопасен за използване в многонишкова среда. Така, когато се получи нова заявка, тя се добавя в опашката и след това диспатчера я взима и пуска за изпълнение. В класа PublicationQueue са имплементирани стандартните методи за този тип структура от данни, като add, remove, pull, peek и size. Първият и вторият метод добавят и изтриват публикация от опашката. Разликата между метода pull и peek е, че първият взима елемента който е най-отпред и го изтрива, а другият метод само взима елемента от опашката, без да го изтрива. Методът сизе връща броя на елементите в опашката.



*Фигура 28. Клас ProcessPublicationQueue.*

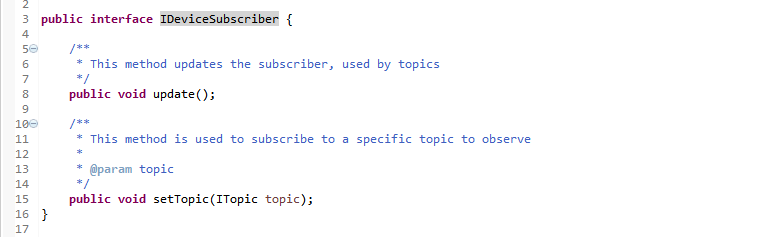
Класа ProcessPublicationQueue наследява класа TimerTask и пренаписва метода run. Това е кодът, който изпълнява следващата заявка от опашката. Метода pull връща следващата наред за изпълнение публикация от опашката, ако опашката е празна, методът pull връща null. След това се проверява дали обектът от тип Publication е null, ако е null метода run приключва своята работа, ако не е null публикацията се обработва и изпълнява. Понеже системата имплементира архитектурния шаблон „Publish - subsribe“, се взема темата на съобщението и се публикува към всички устройства, който са се абонирали за нея.

### 4.1.5 Реализация на шаблона „*Observer*“.

Наблюдател (на английски: Observer) е поведенчески шаблон за дизайн, който се използва в обектно-ориентираното програмиране. Използва се в случаите, при които група от обекти (наблюдатели ) след регистрацията им към регистриращ обект биват оповестявани за промени в неговото състояние.

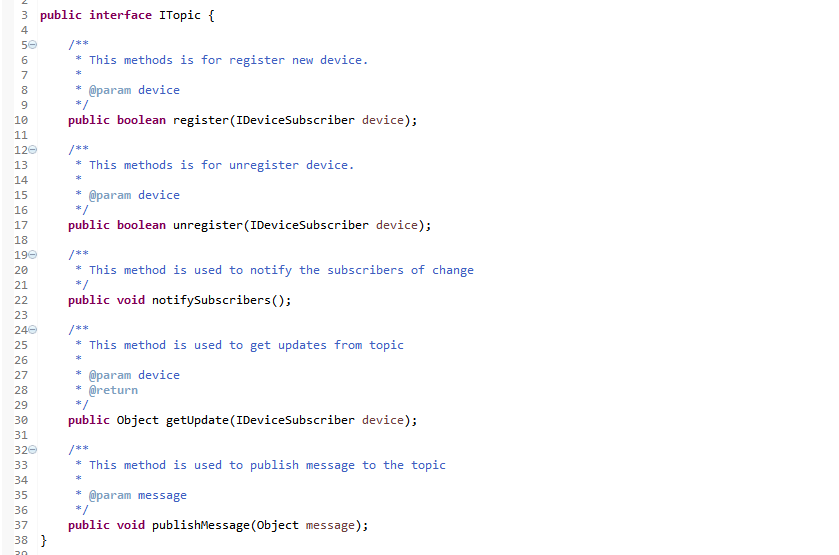
Този шаблон позволява да се имплементират сложни операции, които да извършват устройствата и кода да бъде подреден и лесно разбираем за разработчиците. Благодарение на това, че кода е имплементиран по този шаблон е по-лесно да се реши кои устройства ще бъдат заинтересовани от дадено събитие и да се определи на кои устройства точно да се изпрати съобщение. В системата са имплементирани различни теми (Локация, Време, Температура и т.н). И различните устройства се абонират само за темите, за които те се интересуват. Така, например, когато има някаква промяна по темата температура, само някои от устройствата биват уведомявани. Така се спестява много време от излишно изпращане на съобщение към някое устройство, при положение, че то не може да извърши никаква промяна по тази тема. Например будилника няма нужда да получава съобщение, че температурата в къщата е спаднала.

Имплементирането на този шаблон оставя място за разработването на още много идеи и то по-един по-лесен, елегантен и гъвкав начин.



*Фигура 29. Интерфейс IDeviceSubscriber.*

Това е интерфейсът, който имплементират всички класове от тип Device (всички устройства). Първият метод служи за оповестяване на устройствата, които са абонирани за дадена тема. Вторият метод служи за абониране на устройствата към дадена тема.



*Фигура 30. Интерфейс ITopic.*

Интефейсът ITopic се имплементира от всички класове от тип тема. Методът register служи за абониране на ново устройство към темата, а методът unregister служи за отписване на дадено устройство от темата. Методът notifySubscribers служи за оповестяване на всички устройства, които са се абонирали към темата, че има някаква промяна по тази тема. Методът getUpdate служи за вземане на ново съобщение от темата, а publishMessage се използва за публикуване на ново съобщение.

## 4.2 Програмна реализация на приложението за администратори

### 4.2.1 Реализация на логин на потребител

Логването на потребителя в системата се извършва на няколко стъпки. След като потребителя въведе своите данни за вход, приложението първо прави валидация на въведените данни, след това хешира паролата и създава HTTP заявка към сървъра. Сървърът проверява за потребител с въведените данни и изпраща отговор на клиента приложението. Ако такъв потребител е намерен се създава нов обект от тип HommyUser, ако не съществува такъв потребител приложението връща кратко и ясно съобщение на клиента. Методът за хеширане на паролата може да видите на следващата фигура.

public static string HashPassword(SecureString password)

{

var data = System.Text.Encoding.ASCII.GetBytes(DecryptPassword(password));

data = new System.Security.Cryptography.SHA256Managed().ComputeHash(data);

return System.Text.Encoding.ASCII.GetString(data);

}

*Листинг 1. Метод от класа SecurityUtil, който хешира паролата на потребителя.*

След успешно логване на потребителя неговите данни се зареждат в класа LoggedInUser. Този клас имплементира шаблона Singleton. Този шаблон е подходящ, когато точно един обект е нужен за координацията на други обекти в системата. В системата има само един логнат потребител през цялото време и от където и да бъде извикан обекта от тип HommyUser е нужно всеки път да се върне един и същ обект. Имплементирането на този шаблон пести от ресурсите на системата и решава много други проблеми, когато потребителят редактира своите данни. Този шаблон трябва да се имплементира много добре, защото може да доведе до трудни за откриване грешки в програмата. Един от недостатъците на този шаблон е, че в многонишкова среда е възможно две или повече нишки да създадат обект едновременно. Решението на този проблем е да се синхронизира (ограничи) достъпа до метода, който създава нов обект само от една нишка в даден момент. Реализация на класа LoggedInUser може да видите на следващата фигура.

public sealed class LoggedInUser

{

private static readonly object syncLock = new object();

private static HommyUser loggedInUser;

public static void Init(HommyUser user)

{

if (user != null)

{

loggedInUser = user;

}

}

public static HommyUser GetLoggedInUser()

{

lock (syncLock)

{

return loggedInUser;

}

}

public static void LogoutUser()

{

lock (syncLock)

{

loggedInUser = null;

}

}

}

*Листинг 2. Класа LoggedInUser.*

### 4.2.2 Реализация на “Inversion of control”

public class NinjectConfig

{

public static IKernel Container { get; private set; }

public static void ConfigureContainer()

{

Container = new StandardKernel();

Container.Bind<IUserService>().To<UserService>().InTransientScope();

Container.Bind<IEventService>().To<EventService>().InTransientScope();

Container.Bind<IDeviceService>().To<DeviceService>().InTransientScope();

Container.Bind<IRulesService>().To<RulesService>().InTransientScope();

}

}

*Листинг 3. Ninject конфигурация на приложението за администратори.*

На тази фигура е показана Ninject конфигурацията на приложението. В обекта Container от тип IКernel се свързват( Bind) Интерфейсите и класовете които имплементират тези интерфейси.

private MyDevicesViewModel()

{

service = NinjectConfig.Container.Get<IDeviceService>();

viewHistoryCommand = new RelayCommand(ViewSelectedHistory);

Init();

}

*Листинг 4. Начин на използване на Ninject.*

По този начин от всеки един клас в приложението може се използва обекта, който имплементира IUserService интерфейса без да е нужно да се разбира кой точно е той. Така Ninject спомага за разделянето на приложението на слабо свързани, силно кохезни модули, съединени отново по изключително гъвкав начин.

### 4.2.3 Реализация на връзката със сървъра

За да се изпрати заявка към сървър приложението е необходимо да се създате обект от тип HttpClient. След това се създава заявката, която е от тип HttpRequestMessage, като в конструктура се посочва типа на заявката в този случай е GET, и адреса, на който да се изпрати. И най-накрая заявката се изпраща и се чака отговор от сървър приложението. За по-добра работа на приложението отговорът от сървър приложението се изчаква асинхронно. Асинхронната комуникация е двупосочна комуникация, по време на която има забавяне между изпращането и получаването на съобщенията. По този начин приложението ще продължи работа без да се блокира потребителският интерфейс и същевременно на заден план ще следи за отговор от сървър приложението. На фигурата е показан метода за извличане на всички свързани устройства от сървър приложението.

public class DeviceService : IDeviceService

{

public async Task<string> GetAllDevices()

{

HttpClient http = new HttpClient();

var myRequest = new HttpRequestMessage(HttpMethod.Get, WebService.URIAddress + "devices");

var response = await http.SendAsync(myRequest);

return await response.Content.ReadAsStringAsync();

}

}

*Листинг 5. Метод за извличане на всички свързани устройства от сървър приложението.*

### 4.2.4 Реализация на добавяне на ново правило.

public class AddRuleViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private ICommand addRuleCommand;

private IRulesService RuleService;

private IDeviceService DeviceService;

private static List<Event> events = new List<Event>();

private static List<Device> devices = new List<Device>();

private static DeviceAction selectedDeviceActions = new DeviceAction();

private Rule rule = new Rule();

public AddRuleViewModel()

{

RuleService = NinjectConfig.Container.Get<IRulesService>();

DeviceService = NinjectConfig.Container.Get<IDeviceService>();

addRuleCommand = new RelayCommand(AddRule);

Init();

}

*Листинг 6. Част от класа AddRuleViewModel.*

Едно от изискванията за архитектурен дизайн на приложението за администратор беше да се използва шаблона Model–view–viewmodel (MVVM). На фигурата по-горе е показан част от класа AddRuleViewModel. Както може да забележите той имплементира метода INotifyPropertyChanged и също така има поле addRuleCommand от тип ICommand. Този клас играе ролята на ViewModel в шаблона Model–view–viewmodel (MVVM), а на следващата фигура може да видите част от xaml кода на View файла. Ролята на модел играе класа RulesService, който се инжектва благодарение на Ninject (може да се види на фигура 30).

<UserControl.DataContext>

<local:AddRuleViewModel/>

</UserControl.DataContext>

*Листинг 7. Част от xaml кода на AddRuleView.xaml.*

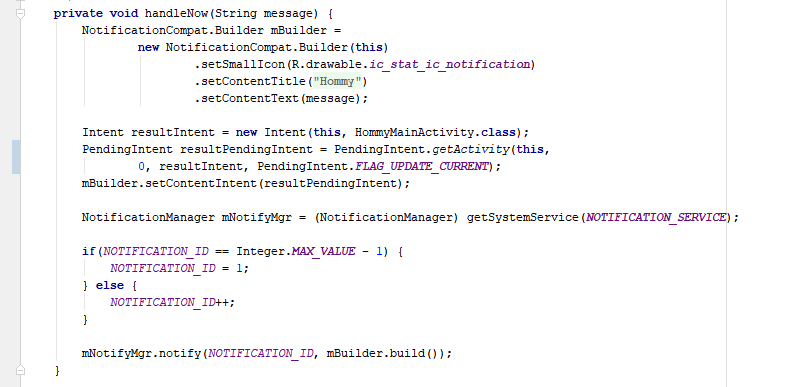
На тази фигура е показана част от xaml кода на AddRuleView.xaml. Както може да забележите той се свързва към класа AddRuleViewModel, по този начин всички графични елементи могат да се свържат с полетата от класа AddRuleViewModel и да извличат и да запазват данни от тях.

## 4.3 Програмна реализация на Андроид приложението

Към момента имплементацията на Андроид приложениете не е финализирана. Разполага се с прототип, изпълняващ една от основните изисквания на приложението, а именно известяване на потребителя за важни събития, които се случват в неговия дом. На следващата подтема може да се разгледа реализацията на това изискване.

### 4.3.1 Реализация на нотификациите

Съобщенията, които се изпращат между сървър приложението и мобилното приложение минават през Google Cloud Messaging. След като се приеме съобщението от сървър приложението, първо се проверява дали е валидно и след това се подава като параметър на метода handleNow, който е показан на следващата фигура. Предназначението на този метод е да създаде нова нотификация и да покаже на потребителя, под формата на нотификация. За да се създаде нотификация се използва NotificationCompat.Builder. Също така е нужно да се добави икона, заглавие за нотификацията и съобщение. След това е необходимо да се създаде нов обект от тип Intent, с два параметъра. Като първи параметър се подава текущото Активити, а другия параматър е активитито, което искаме да се изпълни. Следващата стъпка е да създадем обект от тип NotificationManager. След което, за да се покаже нотификацията на потребителя е нужно да се извика метода notify на обекта от тип NotificationManager, който създадохме в предишната стъпка.



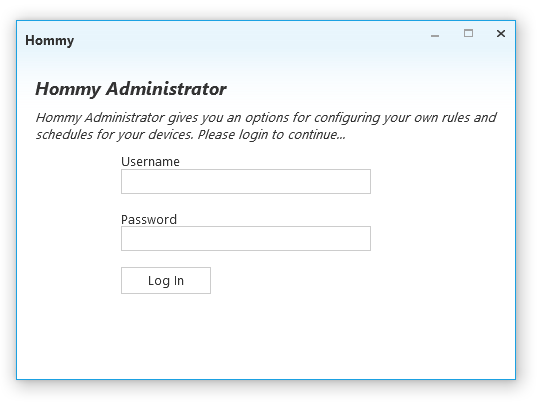
*Фигура 31. Метод който създава нова нотификация и я показва на потребителя.*

Разработката на настоящата дипломна работа се съсредоточава върху направата на сървър приложението, приложението за администратори и цялостната архитектура на тази IoT система. Затова разработката на мобилното приложение остава второстепенна задача за целите на тази дипломна работа и ще бъде доразвито на следващ етап. За да се довърши мобилното приложение докрай, следва да се имплементира логика за следене на основните сензори на устройството. Като например сензора за локализиране на текущото местоположение на мобилните устройства. Така, когато има промяна на текущото местоположение на потребителя (например: когато е извън дома си) ще се изпрати съобщение към сървър приложението, а той от своя страна ще предприеме необходимите действия. За да се увеличават функционалностите на цялата системата, е необходимо да се развива и поддържа непрестанно мобилното приложение. Защото като се добавя поддръжка за следене на повече сензори в устройствата, толкова повече се увеличват и възможностите за конфигуриране, а с тях и ползата от системата. Затова довършването и поддръжката на мобилното приложение е важно за функционирането на цялата система.

# Глава 5. Ръководство на потребителя

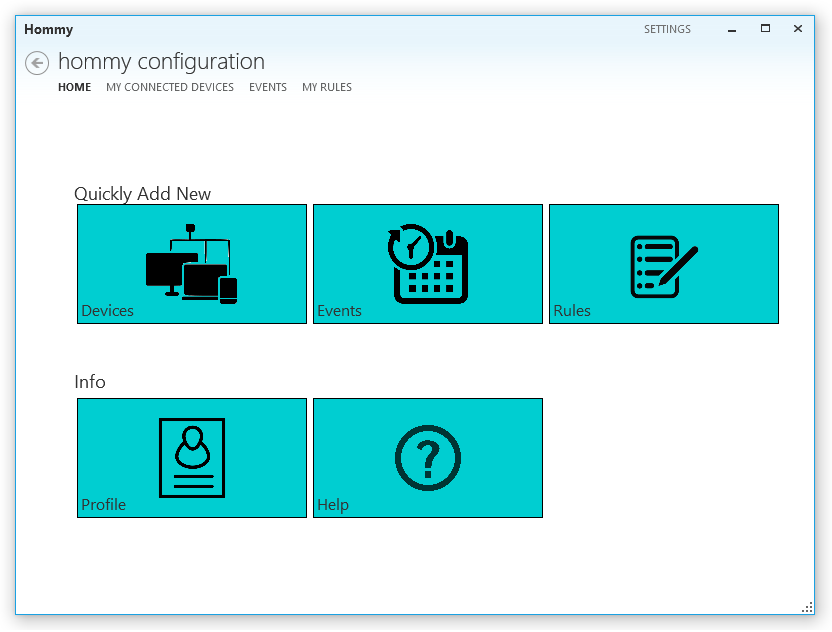
## 5.1. Ръководство на приложението за администратори

За реализацията на потребителския интерфейс е използванa технологията Modern UI, която е с отворен код. Това е отделна библиотека за разработка на красив, интуитивен и функционален потребителски интерфейс за WPF приложения. Тя се инсталира като отделен пакет от NuGet, която представлява онлайн платформа за управление на библиотеки.



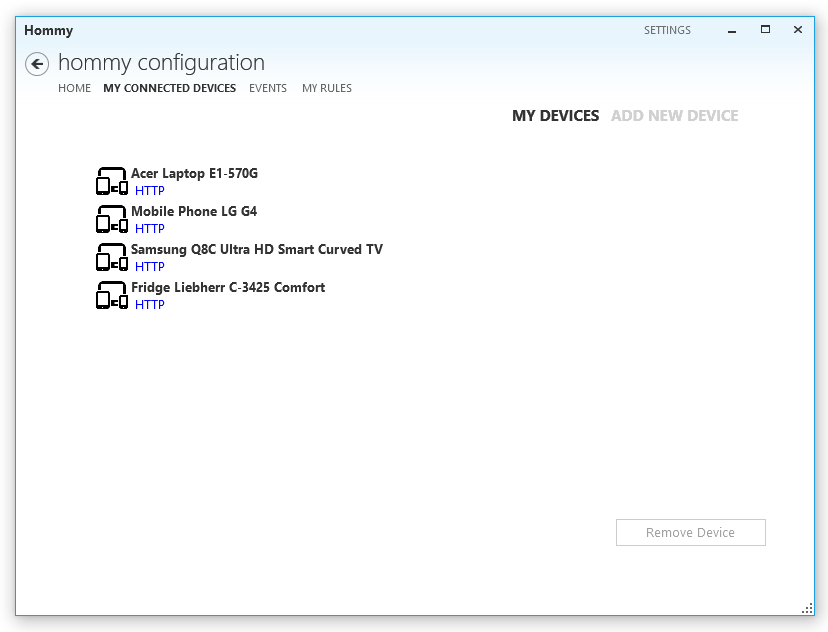
*Фигура 32. Логин прозорец на приложението*

При стартиране на приложението за администратори първо се появява логин прозореца. Като тук е необходимо потребителят да въведе своите данни за вход в приложението.



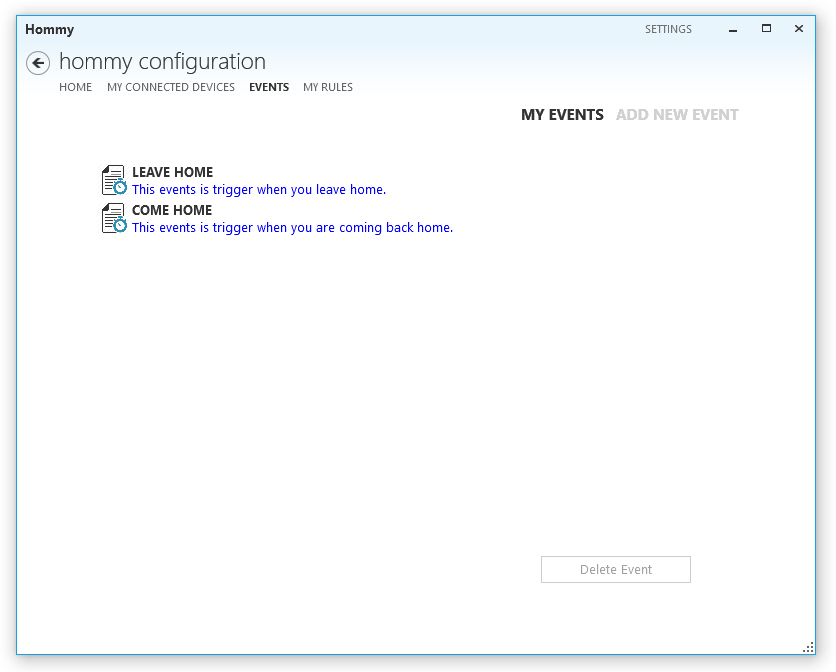
*Фигура 33. Начален екран на прозорец на приложението*

На фигура 33 е показан началния екран на приложението. От тук той има кратки пътища за добавяне на ново устройство, за създаване на ново събитие, за създаване на ново правило. Също така той има директен достъп до своя профил и до упътване за използване на програмата. От горната част на прозореца може да намерите различните табове. От този прозорец може да навигирате директно до прозореца с всички свързани устройства към системата, до прозореца с всички събития или до прозореца с всички правилно създадени от вас. Също така може да забележите, че от горния десен ъгъл



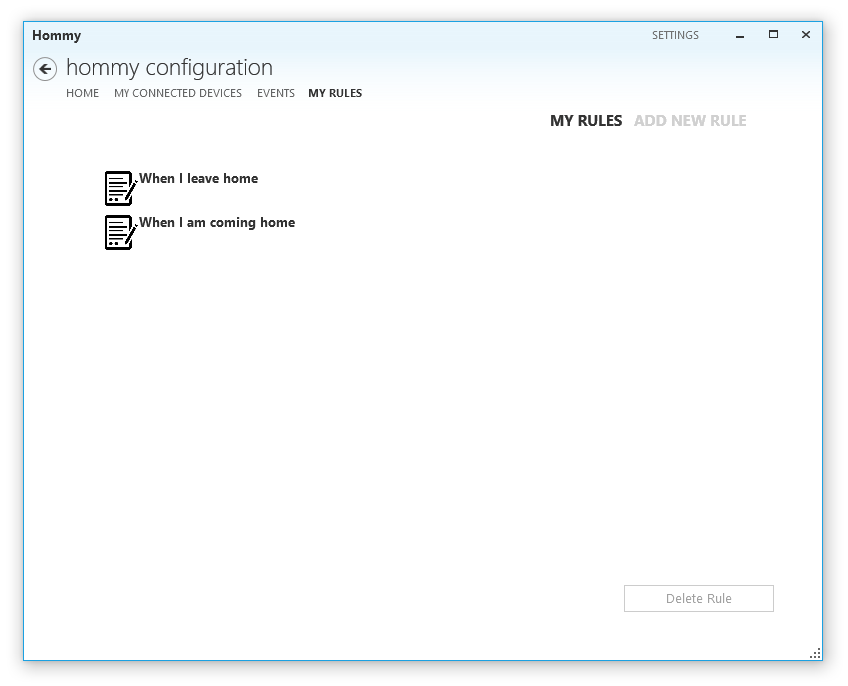
*Фигура 34. Прозорец с всички свързани устройства към системата.*

На фигурата е показан прозореца с всички свързани устройства към системата, както и техния протокол за комуникация. Потребителят може да добавя нови устройства към системата, които да бъдат контролирани от нея. От този прозорец потребителят може да изтрива устройства. Както може да забележите - бутонът за изтриване на устройства не е активен, той ще бъде активен, когато потребителят избере устройство, което иска да изтрие. Менюто за навигация отгоре не изчезва, така че потребителят през цялото време притежава директен достъп до всички други прозорци на системата. Както и достъп до настройките на приложението. От този прозорец потребителят лесно може да навигира до страницата за добавяне на ново устройство.



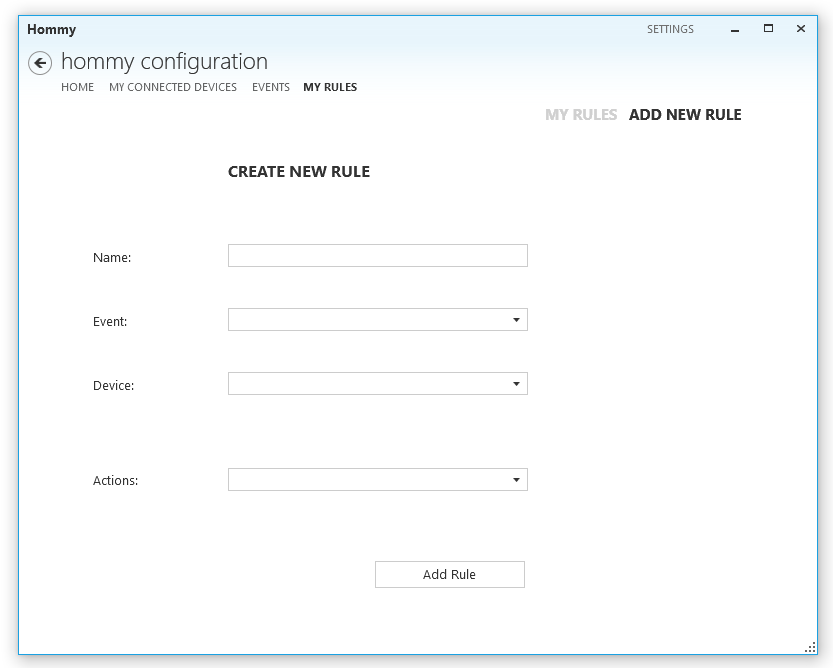
*Фигура 35. Прозорец с всички събития на потребителя.*

Потребителят може да добавя свои събития към системата. На този прозорец той вижда всички събития, които съществуват в системата. Също така се виждат и описанията на тези събития и се разбира кога се задействат те. От този прозорец потребителят може да изтрива събития. Аналогично и тук бутона за изтриване на събитие не е активен, той ще бъде активен, когато потребителят избере събитие, което иска да изтрие. От този прозорец потребителят лесно може да навигира до страницата за добавяне на ново събитие. Необходимо е да се създаде събитие преди да се създаде правило, защото правилата се изпълняват, когато е настъпило някакво събитие.



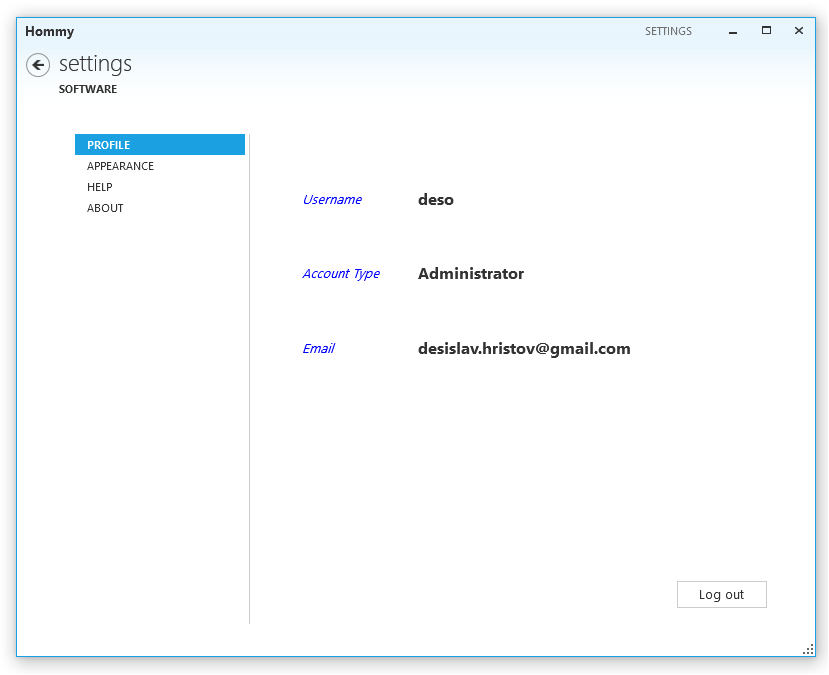
*Фигура 36. Прозорец с всички правила на потребителя.*

Потребителят може да добавя свои правила, които да се изпълняват при настъпване на някакво събитие. На този прозорец той вижда всички правила които е добавил в системата. Също така се виждат и описанията на тези събития и се разбира кога се задействат те. От този прозорец потребителят може да изтрива събития. Както при другите прозорци, така и тук бутонът за изтриване на правило не е активен, той ще бъде активен, когато потребителят избере правило, което иска да изтрие. От този прозорец потребителят лесно може да навигира до страницата за добавяне на ново правило.



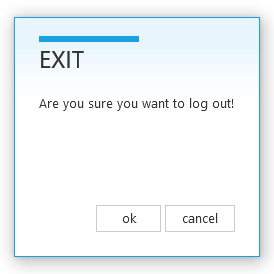
*Фигура 37. Прозорец от където потребителят добавя ново правило.*

От този прозорец потребителят може да добави ново правило в системата, което да се изпълни при настъпване на избраното събитие. За добавянето на ново правило е необходимо да се изпълнят четири стъпки. Първа стъпка е избирането на подходящо и говоримо име за това правило. Като например „включване на телевизора когато се прибера“. Следващата стъпка е да се избере събитие. Защото това правило ще се изпълни когато настъпи събитието, като например „Когато се прибера у нас“. Затова е необходимо да се създаде някакво събитие преди да се създаде правило. Освен това програмата ще предостави опция за избиране на събитие, което вече съществува в системата. Стъпка номер три е да се избере устройство, което да извърши някакво действие. Както в стъпка номер две така и тук, потребителят е необходимо да е добавил ново устройство преди това. Програмата отново ще предостави опция за избиране на устройство, само ако то вече съществува в системата. И последната стъпка е избиране на някакво действие, което да изпълни устройството. Тук опциите ще зависят от устройството, което е било избрано в предишната стъпка. След това е необходимо само да се натисне бутона за добавяне на ново устройство. И то вече може да бъде видяно от прозореца с всички правила на потребителя, показана на фигура 14.



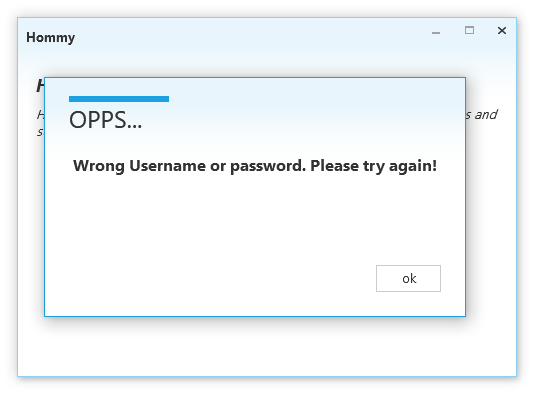
*Фигура 38. Прозорец за настройки на приложението.*

На тази фигура е показан прозореца за настройки на приложението. От тук потребителят има достъп до своя профил, също така може и да се излезе от програмата чрез бутона „Log out“.



*Фигура 39. Прозорец за потвърждаване на изхода от програмата*

След натискането на този бутон се появява нов прозорец за потвърждаване на изхода от програмата. И след потвърждение, чрез натискане на бутона „ОК“ потребителят може да излезе от програмата успешно. При натискане на опцията „Appearance“ oт прозорец за настройки на приложението може да се промени цвета на облика на потребителския интерфейс, като съществуват около 20 цвята, от които потребителят може да избира. Също така от този прозорец може да се избере опцията „Help“, където има подробни инструкции за ползване на програмата, ако потребителят има нужда.



*Фигура 40. Пример на обратна връзка в случай на неправилно въведени потребителско име или парола.*

На тази фигура е даден пример за красив и елегантен начин за известяване на потребителя при възникване на някаква грешка. Приложението притежава добър и абстрактен начин за прихващане на изключения и тяхното показване на крайния потребител. На него се предоставя съобщение, което носи кратка и ясна информация за проблема. Също така се предоставят и няколко опции за изпълнение от потребителя, които са различни в зависимост от проблема.

# Глава 6. Заключение

Към момента разработката на сървър приложението е завършена, както и разработката на приложението за адмистратори. Като те бяха разработени според всички поставени цели и софтуерни архитектури. Двете приложения отговарят на поставените изисквания и изпълняват коректно своите задачи и сценарии. Всички поставени цели бяха осъществени, с използването на едни от най-надеждните и актуални технологии на пазара към момента. Следващата стъпка е да се довърши изцяло андроид приложението, като можем да кажем, че то е на 50% реализирано. В следващата подтема може да разгледате и всички планувани бъдещи подобрения и нови функционалности на системата.

Цялата концепция с „Интернет на нещата“ и по специално - автоматизацията на дома е много иновативна. Тя се променя постояно, и на пазара непрестанно излизат нови решения, които променят начина по-който ние възприемаме тази иновативна концепция. Този проект има за цел да улесни всичко потребители и да допренесе за цялостното развитието на тази идея в бъдеще. Дали ще постигне тази цели, е въпрос само на време за да разберем.

**Бъдещо развитие на системата**

Сама по себе си тази система е доста голяма, защото се състои от много компоненти, които работят заедно. Това е необходимо, защото се работи с много устройства, които работят на различни операционни системи. Основната и най-трудна задача за разработване от сървър приложението беше да се реализира добра софтуерна архитектура, която да обработва и анализира получените данни и да взема решение на база на тях. Също така да се справя с голямото количество данни и да бъде лесна за надграждане в бъдеще. Тази част вече е изпълнена, благодарение на използването на правилните софтуерни шаблони. В следващите версии на приложението е добре да бъдат имплементирани следните функционалности:

* Най-важната функционалност, която е необходимо да се имплементира в следващата версия на системата е - да се подобри сигурността на системата . За да се постигне това е необходимо да се имплементира решение, което да подсигури изпращането на данни и да се предотвратят потенциални хакерски атаки. За момента едно добро решение на проблема изглежда да е протокола HTTPS. Технически HTTPS не е самостоятелен протокол, а резултат от поставянето на протокола за пренос на хипертекст (HTTP) върху SSL/TLS протокол и по този начин се защитава стандартната HTTP връзка. Основната цел на тази комбинация е да се осигури защитена връзка и сигурност при преноса в интернет мрежата. Така съобщенията между устройствата и сървъра ще бъдат криптирани, и по-този начин ще се гарантира по-сигурен канал за комуникация. Има твърдения, че дори този протокол не е достатъчен, за да подсигури канала за комуникацията при „Интернет на нещата“, но за момента все още не измислен по добър начин. [10]
* Една от най-големите и значими идеи, които системата ще е добре да притежава е изкуствен интелект. Така няма да се налага на потребителя да конфигурира системата със своите изисквания, а тя сама ще се учи директно от потребителите. Тя ще свиква с всички ежедневни навици на потребителите и ще бъде възможно да извършва по-точни и по-значими за тях действия. Също така ще може да се съобразява с всички останали членове на семейството, а не просто да се ръководи от зададени правила и събития.
* Много важна функционалност за следващата версия на системата е да се добави поддръжка на нови протоколи за комуникация. Въпреки че с всяка изминала година на пазара се появяват нови домашни уреди, които работят с операционна система (най-популярната за момента е Android), все още не всеки средностатистически потребител може да си позволи да закупи подобни устройства. В този ред на мисли, за да се увеличи целевата група от потребители за тази система е необходимо да се добавят нови протоколи за комуникация с устройствата. Един от най-известните и използвани протоколи за комуникация в домашни условия е Z-wave. Той е безжично предаван протокол, проектиран специално за автоматизация на дома. Той се предава посредством нискочестотни радио вълни между устройствата и центалния контролер.
* Въпреки, че Z-wave е много широко използван протокол за комуникация при системите за управление на дома, съществува и друг протокол, който набира популярност в последните години. MQTT е съкращение от MQ Telemetry Transport. Той представлява лек протокол от типа publish/subscribe за комуникация между устройства, изграден върху TCP/IP. Този протокол също поддържа възможност за криптиране на предаваните данни с TLS/SSL.
* В следващите версии е добре да се добавят (създадат) приложения за операционната система iOS. Taкa хората, които използват тази операционна система, ще могат да свържат своите устройства към системата. Като това ще доведе до увеличаване на броя на клиентите и до увеличаване на базата от поддържани устройства от системата.

Бъдещо развитие на приложението за адмистратори

Приложението за администратор беше разработено от начало до край. Като то изпълнява основния набор от функционалности, които бяха зададени при проектирането на приложението. Едно от нещата, което е добре да притежава приложението за администратори е да бъде възможно да се добавят нови администратори към системата. Това ще улесни значително използването на цялата системата.

Бъдещо развитие на мобилното приложение

Текущата версия на приложението притежава само най-важните за работа функционалности. Една от основните функционалности, която трябва да се развие в следващите версии на приложението е добавянето на повече действия, които да се изпълняват от устройствата. Колкото повече действия се поддържат от устройствата толкова повече се увелича ползата от целия продукт. Като под това се има предвид сценариите, по които се използва цялата система. Друга функционалност, която ще се разработи в следващите версии на приложението е да се предоставя възможност за интеракция с потребителя. Тоест, когато система не е сигурна дали да извърши някакво действие, да изпрати запитване към потребителя и той да може да отговори с положителен или отрицателен отговор. Тази опция отново ще увеличи значително възможностите на системата и нейното използване.

# Източници

[1] Прогнози за всички свързани устройства към интернет мрежата.

<http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated>

[2] Статия за проблема на „Интернет на нещата“ с голямото количество данни.

<https://www.oreilly.com/ideas/the-internet-of-things-has-four-big-data-problems>

[3] Статия за проблема на „Интернет на нещата“ със сигурността.

<http://www.ioti.com/security/10-biggest-iot-security-vulnerabilities>

[4] Описания на предимствата на JAVA EE.

<https://www.java-tips.org/java-ee-tips-100042/17-enterprise-java-beans/1933-what-are-the-main-benefits-of-the-j2ee-platform.html>

[5] Статия за топ 10 софтуери за база от данни.

<http://www.itcareersuccess.com/tech/database.htm>

[6] Статия за най-популярната операционна система за мобилни устройства.

<http://www.pcmag.com/article2/0,2817,2370259,00.asp>

[7] Статия за най-използваната операционна система за 2017 година.

<https://www.weforum.org/agenda/2017/04/the-world-s-most-used-operating-system-you-might-be-surprised/>

[8] Статия за използването на Google Cloud Messaging.

<http://www.zdnet.com/article/io-2013-more-than-half-of-apps-in-google-play-now-use-cloud-messaging/>

[9] Обяснениe за ползата от архитектурния дизайн MVVM

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>

[10] Обяснения защо HTTPS и SSL не са толкова сигурни колкото мислим.

<https://www.sott.net/article/275524-Why-HTTPS-and-SSL-are-not-as-secure-as-you-think>

[11] Статия за всички устройтва, които се очаква да бъдат свъразани към интернет мрежата.

<https://www.t-ink.com/blog/2015/11/16/is-it-time-to-stop-overthinking-the-internet-of-things/>

[12] Част от Конфигурационен файл на системата „Home assistant“

<https://home-assistant.io/components/alexa/>

[13] Статия за главния сървър на Loxone.

<https://www.loxone.com/enen/products/miniserver-extensions/>

[14] Статия за примерна конфигурация на Maven

<http://gutsev.com/install-maven-configurate-repository/>

[15] Статия за използването на база от данни и JDBC.

<http://www-it.fmi.uni-sofia.bg/courses/WDB/db.htm>

# Приложение 1. Програмен код

**Приложението за администратори**

**Rules**

AddRuleViewModel.cs

using Hospital.Models;

using NInjectConfigProject;

using Personal.Health.Care.DesktopApp.Utills;

using Personal.Health.Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

using Ninject;

using System.Runtime.CompilerServices;

using System.Windows;

using Personal.Health.Care.DesktopApp.Pages.Views;

using Personal.Health.Models;

using Personal.Health.Care.DesktopApp.Model;

using Personal.Health.Services.ServiceInterfaces;

using Newtonsoft.Json;

namespace Personal.Health.Care.DesktopApp.ViewModels

{

public class AddRuleViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private ICommand addRuleCommand;

private IRulesService RuleService;

private IDeviceService DeviceService;

private static List<Event> events = new List<Event>();

private static List<Device> devices = new List<Device>();

private static DeviceAction selectedDeviceActions = new DeviceAction();

private Rule rule = new Rule();

public AddRuleViewModel()

{

RuleService = NinjectConfig.Container.Get<IRulesService>();

DeviceService = NinjectConfig.Container.Get<IDeviceService>();

addRuleCommand = new RelayCommand(AddRule);

Init();

}

private async void Init()

{

string response = await NinjectConfig.Container.Get<IEventService>().GetAllEvents();

EventsCollection allEvents = JsonConvert.DeserializeObject<EventsCollection>(response);

AllEvents = allEvents.Events;

string allDevicesFromServer = await NinjectConfig.Container.Get<IDeviceService>().GetAllDevices();

DevicesCollection allDevices = JsonConvert.DeserializeObject<DevicesCollection>(allDevicesFromServer);

AllDevices = allDevices.Devices;

}

#region Properties

public Rule Rule

{

get { return rule; }

set { rule = value; NotifyPropertyChanged(); }

}

public List<Event> AllEvents

{

get { return events; }

set { events = value; NotifyPropertyChanged(); }

}

public DeviceAction SelectedDeviceActions

{

get { return selectedDeviceActions; }

set { selectedDeviceActions = value; NotifyPropertyChanged(); }

}

public List<Device> AllDevices

{

get { return devices; }

set { devices = value; NotifyPropertyChanged(); }

}

public ICommand AddRuleCommand

{

get { return addRuleCommand; }

set { addRuleCommand = value; }

}

#endregion

#region INotifyPropertyChanged

private void NotifyPropertyChanged([CallerMemberName] string propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

#endregion

#region Add Rule Code

public void AddRule(Object obj)

{

RuleDao simpleRule = new RuleDao();

simpleRule.userId = LoggedInUser.GetLoggedInUser().Id;

simpleRule.eventId = Rule.Event.id;

simpleRule.deviceId = Rule.Device.Id;

simpleRule.actionId = SelectedDeviceActions.Id;

simpleRule.ruleName = Rule.Name;

if (simpleRule != null)

{

RuleService.addNewRule(simpleRule);

Rule = new Rule();

RulesViewModel.GetInstance().LoadRules();

System.Windows.Threading.Dispatcher.CurrentDispatcher.Invoke((Action)(() =>

{

Messenger.ShowMessage("Result", "Rule created successfully");

}));

}

}

#endregion

}

}

AddRuleOptionView.xaml

<UserControl x:Class="Personal.Health.Care.DesktopApp.Pages.Options.VisitationOptions.AddVisitationOption"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit"

xmlns:xcdg="http://schemas.xceed.com/wpf/xaml/datagrid"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Personal.Health.Care.DesktopApp.ViewModels"

xmlns:mui="http://firstfloorsoftware.com/ModernUI"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<UserControl.DataContext>

<local:AddRuleViewModel/>

</UserControl.DataContext>

<Grid Style="{StaticResource ContentRoot}">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition />

<RowDefinition Height="2\*"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="4\*"/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.Column="1" Text="CREATE NEW RULE" Style="{StaticResource Heading2}" />

<Label Grid.Row="1" Grid.Column="0" Content="Name:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120"/>

<TextBox Grid.Row="1" Grid.Column="1" Text="{Binding Rule.Name, Mode=TwoWay}" HorizontalAlignment="Left" VerticalAlignment="Center" Width="300"/>

<Label Grid.Row="2" Grid.Column="0" Content="Event:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120"/>

<ComboBox Grid.Row="2" Grid.Column="1" ItemsSource="{Binding AllEvents}" SelectedItem="{Binding Rule.Event, Mode=TwoWay}" HorizontalAlignment="Left" VerticalAlignment="Center" Width="300">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding Path=name}" />

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<Label Grid.Row="3" Grid.Column="0" Content="Device:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120"/>

<ComboBox Grid.Row="3" Grid.Column="1" ItemsSource="{Binding AllDevices}" SelectedItem="{Binding Rule.Device, Mode=TwoWay}" HorizontalAlignment="Left" VerticalAlignment="Center" Width="300">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding Path=Name}" />

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<Label Grid.Row="4" Grid.Column="0" Content="Actions:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120"/>

<ComboBox Grid.Row="4" Grid.Column="1" ItemsSource="{Binding Rule.Device.Actions}" SelectedItem="{Binding SelectedDeviceActions}" HorizontalAlignment="Left" VerticalAlignment="Center" Width="300">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding Path=action}" />

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<StackPanel Grid.Row="5" Grid.Column="1" Orientation="Horizontal" >

<Button Content="Add Rule" Command="{Binding AddRuleCommand}" HorizontalAlignment="Right" VerticalAlignment="Center" Width="150" Margin="147,0,0,0"/>

</StackPanel>

</Grid>

</UserControl>

RulesViewModel.cs

using NInjectConfigProject;

using Personal.Health.Models;

using Personal.Health.Services.ServiceInterfaces;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Ninject;

using System.Runtime.CompilerServices;

using Personal.Health.Care.DesktopApp.Utills;

using System.Windows.Input;

using Personal.Health.Care.DesktopApp.Pages.Views;

using Hospital.Models;

using Personal.Health.Care.DesktopApp.Model;

using Personal.Health.Services;

using System.Windows;

using Newtonsoft.Json;

namespace Personal.Health.Care.DesktopApp.ViewModels

{

public class RulesViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private static RulesViewModel instance;

private List<Rule> rules;

private IRulesService service;

private IEventService visitationService;

private Rule selectedRule;

private Boolean isSelected;

#region Constructor

public RulesViewModel()

{

service = NinjectConfig.Container.Get<IRulesService>();

visitationService = NinjectConfig.Container.Get<IEventService>();

LoadRules();

}

public static RulesViewModel GetInstance()

{

if (instance == null)

{

instance = new RulesViewModel();

}

return instance;

}

#endregion

#region Properties

public List<Rule> AllRules { get { return rules; } set { rules = value; NotifyPropertyChanged(); } }

public Rule SelectedRule { get { return selectedRule; } set { HasSelectedRule = true; selectedRule = value; NotifyPropertyChanged(); } }

public Boolean HasSelectedRule { get { return isSelected; } set { isSelected = value; NotifyPropertyChanged(); } }

#endregion

#region INotifyPropertyChanged

private void NotifyPropertyChanged([CallerMemberName] string propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

#endregion

#region Load Rules Code

public async void LoadRules()

{

string response = await service.getRulesNameForUser(1);

RulesCollection ReturnedRules = JsonConvert.DeserializeObject<RulesCollection>(response);

AllRules = ReturnedRules.Rules;

}

#endregion

}

}

IRulesService.cs

using Personal.Health.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Personal.Health.Services.ServiceInterfaces

{

public interface IRulesService

{

Task<string> getRulesNameForUser(int userId);

Task addNewRule(RuleDao rule);

}

}

RulesService.cs

using Newtonsoft.Json;

using Personal.Health.Models;

using Personal.Health.Services.ServiceInterfaces;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

namespace Personal.Health.Services.Impl.ServiceImpl

{

public class RulesService : IRulesService

{

public async Task<string> getRulesNameForUser(int userId)

{

HttpClient http = new HttpClient();

var myRequest = new HttpRequestMessage(HttpMethod.Get, WebService.URIAddress + "rules?userId=" + userId);

var response = await http.SendAsync(myRequest);

return await response.Content.ReadAsStringAsync();

}

public async Task addNewRule(RuleDao rule)

{

var httpWebRequest = (HttpWebRequest)WebRequest.Create(WebService.URIAddress + "rules/add");

httpWebRequest.ContentType = "application/json";

httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))

{

string json = JsonConvert.SerializeObject(rule);

streamWriter.Write(json);

streamWriter.Flush();

streamWriter.Close();

}

var httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();

String result;

using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

{

result = streamReader.ReadToEnd();

}

OperationResult OperationResult = JsonConvert.DeserializeObject<OperationResult>(result);

}

}

}

**Events**

AddEventOption.xaml

<UserControl x:Class="Personal.Health.Care.DesktopApp.Pages.Options.TemplateOptions.AddTemplateOption"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Personal.Health.Care.DesktopApp.ViewModels"

xmlns:mui="http://firstfloorsoftware.com/ModernUI"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<UserControl.DataContext>

<local:AddEventViewModel/>

</UserControl.DataContext>

<Grid Style="{StaticResource ContentRoot}">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition Height="2\*"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="4\*"/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.Column="1" Text="CREATE NEW EVENT" Style="{StaticResource Heading2}" />

<Label Grid.Row="1" Grid.Column="0" Content="Event Name:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="150"/>

<TextBox Grid.Row="1" Grid.Column="1" Text="{Binding EventObject.name, Mode=TwoWay}" HorizontalAlignment="Left" VerticalAlignment="Center" Width="300"/>

<Label Grid.Row="2" Grid.Column="0" Content="Triggered By:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120"/>

<ComboBox Grid.Row="2" Grid.Column="1" ItemsSource="{Binding Triggers}" SelectedItem="{Binding Trigger}" HorizontalAlignment="Left" VerticalAlignment="Center" Width="300">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding Path=Name}" />

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

<Label Grid.Row="3" Grid.Column="0" Content="Description:" HorizontalAlignment="Left" VerticalAlignment="Center" Width="120"/>

<TextBox Grid.Row="3" Grid.Column="1" Text="{Binding EventObject.description, Mode=TwoWay}" Height="50" TextWrapping="Wrap" HorizontalAlignment="Left" VerticalAlignment="Center" Width="537"/>

<Button Grid.Row="4" Grid.Column="1" Content="Add Event" Command="{Binding AddEventCommand}" HorizontalAlignment="Right" VerticalAlignment="Top" Width="120"/>

</Grid>

</UserControl>

AddEventViewModel.cs

using Hospital.Models;

using NInjectConfigProject;

using Personal.Health.Models;

using Personal.Health.Services;

using Personal.Health.Services.ServiceInterfaces;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

using Ninject;

using Personal.Health.Care.DesktopApp.Utills;

using System.Runtime.CompilerServices;

using System.Windows;

using Personal.Health.Care.DesktopApp.Model;

namespace Personal.Health.Care.DesktopApp.ViewModels

{

public class AddEventViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private List<HommyEventTrigger> triggers;

private ICommand addEventCommand;

private Event eventObject;

private HommyEventTrigger triger;

private IEventService service;

public AddEventViewModel()

{

eventObject = new Event();

Triggers = new List<HommyEventTrigger>();

Triggers.Add(new HommyEventTrigger("TIME"));

Triggers.Add(new HommyEventTrigger("LOCATION"));

Triggers.Add(new HommyEventTrigger("TEMPERATURE"));

service = NinjectConfig.Container.Get<IEventService>();

addEventCommand = new RelayCommand(AddEvent);

}

#region Properties

public Event EventObject

{

get { return eventObject; }

set { eventObject = value; NotifyPropertyChanged(); }

}

public HommyEventTrigger Trigger

{

get { return triger; }

set { triger = value; }

}

public List<HommyEventTrigger> Triggers

{

get { return triggers; }

set { triggers = value; }

}

public ICommand AddEventCommand

{

get { return addEventCommand; }

set { addEventCommand = value; }

}

#endregion

#region INotifyPropertyChanged

private void NotifyPropertyChanged([CallerMemberName] string propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

#endregion

#region Add Template Code

public void AddEvent(Object obj)

{

Event newEvent = new Event();

newEvent.name = eventObject.name;

newEvent.description = eventObject.description;

newEvent.triggeredBy = Trigger.Name;

EventObject = new Event();

Trigger = new HommyEventTrigger();

service.addNewEvent(newEvent);

EventsViewModel.GetInstance().LoadEvents();

System.Windows.Threading.Dispatcher.CurrentDispatcher.Invoke((Action)(() =>

{

Messenger.ShowMessage("Result", "Event created successfully");

}));

}

#endregion

}

}

EventsViewModel.cs

using Hospital.Models;

using NInjectConfigProject;

using Personal.Health.Services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Ninject;

using System.Runtime.CompilerServices;

using System.ComponentModel;

using System.Windows.Input;

using Personal.Health.Care.DesktopApp.Utills;

using System.Windows;

using Personal.Health.Care.DesktopApp.Pages.Views;

using FluentDateTime;

using Personal.Health.Care.DesktopApp.Model;

using Newtonsoft.Json;

using Personal.Health.Models;

namespace Personal.Health.Care.DesktopApp.ViewModels

{

public class EventsViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private static EventsViewModel instance;

private List<Event> events;

private IEventService service;

private Event selectedEvent;

private Boolean hasSelectedEvent;

#region Constructor

private EventsViewModel()

{

service = NinjectConfig.Container.Get<IEventService>();

LoadEvents();

}

public static EventsViewModel GetInstance()

{

if (instance == null)

{

instance = new EventsViewModel();

}

return instance;

}

#endregion

#region Properties

public List<Event> AllEvents { get { return events; } set { events = value; NotifyPropertyChanged(); } }

public Event SelectedEvent { get { return selectedEvent; } set { HasSelectedEvent = true; selectedEvent = value; NotifyPropertyChanged(); } }

public Boolean HasSelectedEvent { get { return hasSelectedEvent; } set { hasSelectedEvent = value; NotifyPropertyChanged(); } }

#endregion

#region INotifyPropertyChanged

private void NotifyPropertyChanged([CallerMemberName] string propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

#endregion

public async void LoadEvents()

{

string response = await service.GetAllEvents();

EventsCollection devices = JsonConvert.DeserializeObject<EventsCollection>(response);

AllEvents = devices.Events;

}

}

}

IEventService.cs

using Hospital.Models;

using Personal.Health.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Personal.Health.Services

{

public interface IEventService

{

Task<string> GetAllEvents();

Task addNewEvent(Event NewEvent);

}

}

EventService.cs

using Hospital.Models;

using System.Collections.Generic;

using Newtonsoft.Json;

using Personal.Health.Services.Impl.ServiceImpl;

using System.Threading.Tasks;

using System.Net.Http;

using Personal.Health.Models;

using System.Net;

using System.IO;

using System;

namespace Personal.Health.Services.Impl

{

public class EventService : IEventService

{

public async Task<string> GetAllEvents()

{

HttpClient http = new HttpClient();

var myRequest = new HttpRequestMessage(HttpMethod.Get, WebService.URIAddress + "events");

var response = await http.SendAsync(myRequest);

return await response.Content.ReadAsStringAsync();

}

public async Task addNewEvent(Event NewEvent)

{

var httpWebRequest = (HttpWebRequest)WebRequest.Create(WebService.URIAddress + "events/add");

httpWebRequest.ContentType = "application/json";

httpWebRequest.Method = "POST";

String json = JsonConvert.SerializeObject(NewEvent);

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))

{

streamWriter.Write(json);

streamWriter.Flush();

streamWriter.Close();

}

var httpResponse = (HttpWebResponse)httpWebRequest.GetResponse();

String result;

using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))

{

result = streamReader.ReadToEnd();

}

OperationResult OperationResult = JsonConvert.DeserializeObject<OperationResult>(result);

}

}

}

**Devices**

MyDevicesViewModel

using Hospital.Models;

using NInjectConfigProject;

using Personal.Health.Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using Ninject;

using Personal.Health.Care.DesktopApp.Model;

using System.Windows.Input;

using Personal.Health.Care.DesktopApp.Pages.Views;

using Personal.Health.Care.DesktopApp.Utills;

using Personal.Health.Models;

using Newtonsoft.Json;

namespace Personal.Health.Care.DesktopApp.ViewModels

{

public class MyDevicesViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private static MyDevicesViewModel instance;

private List<Device> devices;

private IDeviceService service;

private ICommand viewHistoryCommand;

private Device selectedDevice;

private Boolean hasSelectedDevice;

#region Constructor

private MyDevicesViewModel()

{

service = NinjectConfig.Container.Get<IDeviceService>();

viewHistoryCommand = new RelayCommand(ViewSelectedHistory);

Init();

}

private async void Init()

{

string response = await service.GetAllDevices();

DevicesCollection devices = JsonConvert.DeserializeObject<DevicesCollection>(response);

AllDevices = devices.Devices;

}

private void ViewSelectedHistory(object obj)

{

// ViewHistory viewHistory = new ViewHistory(SelectedHistory);

//viewHistory.ShowDialog();

}

public static MyDevicesViewModel GetInstance()

{

if (instance == null)

{

instance = new MyDevicesViewModel();

}

return instance;

}

#endregion

#region Properties

public List<Device> AllDevices { get { return devices; } set { devices = value; NotifyPropertyChanged(); } }

public Device SelectedDevice { get { return selectedDevice; } set { HasSelectedDevice = true; selectedDevice = value; NotifyPropertyChanged(); } }

public ICommand ViewHistoryCommand { get { return viewHistoryCommand; } set { viewHistoryCommand = value; NotifyPropertyChanged(); }}

public Boolean HasSelectedDevice { get { return hasSelectedDevice; } set { hasSelectedDevice = value; NotifyPropertyChanged(); } }

#endregion

#region INotifyPropertyChanged

private void NotifyPropertyChanged([CallerMemberName] string propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

#endregion

}

}

IDeviceService.cs

using Hospital.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Personal.Health.Services

{

public interface IDeviceService

{

Task<string> GetAllDevices();

}

}

DeviceService.cs

using Hospital.Models;

using Newtonsoft.Json;

using Newtonsoft.Json.Converters;

using Personal.Health.Services.Impl.HospitalWebService;

using Personal.Health.Services.Impl.ServiceImpl;

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Threading.Tasks;

namespace Personal.Health.Services.Impl

{

public class DeviceService : IDeviceService

{

public async Task<string> GetAllDevices()

{

HttpClient http = new HttpClient();

var myRequest = new HttpRequestMessage(HttpMethod.Get, WebService.URIAddress + "devices");

var response = await http.SendAsync(myRequest);

return await response.Content.ReadAsStringAsync();

}

}

}

**Users**

LoginViewModel.cs

using NInjectConfigProject;

using Personal.Health.Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Security;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

using Ninject;

using Personal.Health.Care.DesktopApp.Utills;

using System.Runtime.CompilerServices;

using Hospital.Models;

using System.Text.RegularExpressions;

using System.Windows;

using Personal.Health.Care.DesktopApp.Model;

using Newtonsoft.Json;

using Personal.Health.Models;

namespace Personal.Health.Care.DesktopApp.ViewModels

{

public class LoginViewModel

{

public event PropertyChangedEventHandler PropertyChanged;

private IUserService service;

private ICommand loginCommand;

private string username;

private SecureString password;

#region Constructor

public LoginViewModel()

{

NinjectConfig.ConfigureContainer();

service = NinjectConfig.Container.Get<IUserService>();

loginCommand = new RelayCommand(LoginPatient);

}

#endregion

#region Class Properties

public ICommand LoginCommand

{

get { return loginCommand; }

set { loginCommand = value; }

}

public string Username

{

get { return username; }

set { username = value; NotifyPropertyChanged(); }

}

public SecureString Password

{

get { return password; }

set

{

if (password != value)

{

password = value;

NotifyPropertyChanged();

}

}

}

#endregion

#region INotifyPropertyChanged

private void NotifyPropertyChanged([CallerMemberName] string propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

#endregion

#region Login User code

public async void LoginPatient(object obj)

{

if (Username == null || Password == null)

{

Messenger.ShowMessage(" Please enter username and password first! ");

return;

}

else

{

string response = await service.LoginUserAsync(Username, SecurityUtil.ConvertToString(password));

if (Utill.isValidUser(response))

{

var loginWindow = Application.Current.Windows.OfType<Window>().SingleOrDefault(x => x.IsActive);

if (loginWindow != null)

{

HommyUser user = JsonConvert.DeserializeObject<HommyUser>(response);

LoggedInUser.Init(user);

MainWindow mainWindow = new MainWindow();

mainWindow.Show();

loginWindow.Close();

}

else

{

Messenger.ShowMessage(" Something went wrong! ");

}

}

else

{

Messenger.ShowMessage(" Wrong Username or password. Please try again! ");

}

}

}

#endregion

}

}

IUserService.cs

using Hospital.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Personal.Health.Services

{

public interface IUserService

{

Task<string> LoginUserAsync(string username, string password);

}

}

UserService.cs

using Hospital.Models;

using System;

using Newtonsoft.Json;

using System.Net.Http;

using Personal.Health.Services.Impl.ServiceImpl;

using System.Threading.Tasks;

namespace Personal.Health.Services.Impl

{

public class UserService : IUserService

{

public async Task<string> LoginUserAsync(string username, string password)

{

HttpClient http = new HttpClient();

var myRequest = new HttpRequestMessage(HttpMethod.Get, WebService.URIAddress + "users/login?username=" + username + "&password=" + password);

var resp = await http.SendAsync(myRequest);

return await resp.Content.ReadAsStringAsync();

}

}

}

**Home and login (xaml)**

Home.xaml

<UserControl x:Class="Personal.Health.Care.DesktopApp.Pages.Home"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:Controls="clr-namespace:MahApps.Metro.Controls;assembly=MahApps.Metro"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Personal.Health.Care.DesktopApp.ViewModels"

xmlns:mui="http://firstfloorsoftware.com/ModernUI"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<UserControl.DataContext>

<local:MainPageViewModel/>

</UserControl.DataContext>

<Grid Style="{StaticResource ContentRoot}">

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

<RowDefinition Height="5\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="5\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.ColumnSpan="3" HorizontalAlignment="Left" VerticalAlignment="Top" Text="{Binding User.WelcomeUser}" FontSize="20" FontStyle="Oblique" Style="{StaticResource Heading2}" />

<TextBlock Grid.Row="1" Grid.Column="0" HorizontalAlignment="Left" VerticalAlignment="Top" Text="Quickly Add New" FontSize="18" FontStyle="Normal" Margin="0,10,0,0"/>

<TextBlock Grid.Row="2" Grid.Column="0" HorizontalAlignment="Left" VerticalAlignment="Top" Text="Info" FontSize="18" Margin="0,10,0,0" FontStyle="Normal"/>

<Controls:Tile Height="120" Width="230" Title="Devices" Grid.Row="1" ToolTip="Quickly connect new device" Command="NavigationCommands.GoToPage" CommandParameter="Pages/Options/HistoryOptions/AddDeviceOption.xaml" HorizontalAlignment="Left" VerticalAlignment="Center" Grid.Column="0" Background="DarkTurquoise" BorderBrush="Black" BorderThickness="1">

<Image Source="../../Images/Icons/dev.png" HorizontalAlignment="Center" VerticalAlignment="Center" Height="80"/>

</Controls:Tile>

<Controls:Tile Height="120" Width="230" Title="Events" ToolTip="Quickly add a new event" Command="NavigationCommands.GoToPage" CommandParameter="Pages/Options/TemplateOptions/AddEventOption.xaml" HorizontalAlignment="Left" VerticalAlignment="Center" Grid.Row="1" Grid.Column="1" Background="DarkTurquoise" BorderBrush="Black" BorderThickness="1">

<Image Source="../../Images/Icons/past\_events.png" HorizontalAlignment="Left" VerticalAlignment="Center" Height="80"/>

</Controls:Tile>

<Controls:Tile Height="120" Width="230" Title="Rules" ToolTip="Quickly add a new rule" Command="NavigationCommands.GoToPage" CommandParameter="Pages/Options/VisitationOptions/AddRuleOptionView.xaml" HorizontalAlignment="Left" VerticalAlignment="Center" Grid.Row="1" Grid.Column="2" Background="DarkTurquoise" BorderBrush="Black" BorderThickness="1">

<Image Source="../../Images/Icons/template.png" HorizontalAlignment="Left" VerticalAlignment="Center" Height="80"/>

</Controls:Tile>

<Controls:Tile Height="120" Width="230" Title="Profile" ToolTip="View Profile" Command="NavigationCommands.GoToPage" CommandParameter="/Pages/Settings/PatientProfileView.xaml" HorizontalAlignment="Left" VerticalAlignment="Top" Grid.Row="3" Grid.Column="0" Background="DarkTurquoise" BorderBrush="Black" BorderThickness="1">

<Image Source="../../Images/Icons/profile.png" HorizontalAlignment="Left" VerticalAlignment="Center" Height="80"/>

</Controls:Tile>

<Controls:Tile Height="120" Width="230" Title="Help" Command="NavigationCommands.GoToPage" ToolTip="Show Help" CommandParameter="/Pages/Settings/HelpView.xaml" HorizontalAlignment="Left" VerticalAlignment="Top" Grid.Row="3" Grid.Column="1" Background="DarkTurquoise" BorderBrush="Black" BorderThickness="1">

<Image Source="../../Images/Icons/help.png" HorizontalAlignment="Left" VerticalAlignment="Center" Height="80"/>

</Controls:Tile>

</Grid>

</UserControl>

LoginView.xaml

<mui:ModernWindow x:Class="Personal.Health.Care.DesktopApp.Pages.UserManagement.LoginView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mui="http://firstfloorsoftware.com/ModernUI"

xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"

xmlns:local="clr-namespace:Personal.Health.Care.DesktopApp.ViewModels"

xmlns:utilities="clr-namespace:Personal.Health.Care.DesktopApp.Utills"

MinHeight="350"

MinWidth="480"

mc:Ignorable="d"

Title="Hommy"

ResizeMode="NoResize"

Style="{StaticResource BlankWindow}" Height="360" Width="500">

<Window.DataContext>

<local:LoginViewModel/>

</Window.DataContext>

<Window.InputBindings>

<KeyBinding Key="Enter" Command="{ Binding LoginCommand}" />

</Window.InputBindings>

<Window.Background>

<ImageBrush ImageSource="../../Images/Pictures/login.jpg"/>

</Window.Background>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="1.5\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1.8\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="3\*"/>

<ColumnDefinition Width="1\*"/>

</Grid.ColumnDefinitions>

<!-- Application Name -->

<Label Grid.Row="0" Grid.ColumnSpan="3" Content="Hommy Administrator " HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top"

Width="482" FontStyle="Italic" FontSize="18" FontStretch="Condensed" FontWeight="Bold" Height="38"/>

<!-- Application details -->

<Label Grid.Row="0" Grid.ColumnSpan="3" Content="Hommy Administrator gives you an options for configuring your own rules and&#xD;&#xA;schedules for your devices. Please login to continue..." FontStyle="Oblique"

HorizontalAlignment="Left" Margin="10,43,-48,0" VerticalAlignment="Top" Width="482" Height="35"/>

<!-- Log on form -->

<Label Grid.Row="1" Grid.Column="1" Content="Username" HorizontalAlignment="Left" VerticalAlignment="Top" Width="150"/>

<Label Grid.Row="2" Grid.Column="1" Content="Password" HorizontalAlignment="Left" VerticalAlignment="Top" Width="150"/>

<TextBox Grid.Row="1" Grid.Column="1" HorizontalAlignment="Left" Text="{Binding Path=Username}" Height="25" TextWrapping="Wrap" VerticalAlignment="Center" Width="250"/>

<PasswordBox Grid.Row="2" Grid.Column="1" Name="password" HorizontalAlignment="Left" Height="25" VerticalAlignment="Center" Width="250">

<i:Interaction.Behaviors>

<utilities:PasswordBoxBindingBehavior Password="{Binding Password}" />

</i:Interaction.Behaviors>

</PasswordBox>

<StackPanel Grid.Row="3" Grid.Column="1" Orientation="Horizontal">

<Button Content="Log In" HorizontalAlignment="Left" VerticalAlignment="Top" Width="90" Command="{Binding LoginCommand}" />

</StackPanel>

</Grid>

</mui:ModernWindow>

**Сървър Приложението**

**Dao Interfaces**

IAction.java

package com.hommy.administrator.dao.interfaces;

import java.util.List;

import com.hommy.administrator.dao.objects.Action;

public interface IAction {

public List<Action> getActions(List<Integer> Actionsid);

public boolean addAction(Action action);

}

IDevice.java

package com.hommy.administrator.dao.interfaces;

import com.hommy.administrator.dao.objects.Device;

import com.hommy.administrator.dao.objects.Devices;

public interface IDevice {

public Devices getAllDevice();

public Device getDevice(String name);

public boolean addDevice(Device device);

public boolean removeDevice(int id);

}

IEvent.java

package com.hommy.administrator.dao.interfaces;

import com.hommy.administrator.dao.objects.Event;

import com.hommy.administrator.dao.objects.Events;

public interface IEvent {

public Events getAllEvents();

public Event getEvent(String name);

public boolean addEvent(Event event);

public boolean removeEvent(int id);

}

IRule.java

package com.hommy.administrator.dao.interfaces;

import com.hommy.administrator.dao.objects.Rule;

import com.hommy.administrator.dao.objects.Rules;

public interface IRule {

public Rules getAllRules();

public Rule getRule(Rule rule);

public boolean addRule(Rule rule);

public void removeRule(Rule rule);

}

IUser.java

package com.hommy.administrator.dao.interfaces;

import com.hommy.administrator.dao.objects.User;

public interface IUser {

public User getUser(String username, String password);

public void addUser(User user);

public void removeUser(User user);

}

**Dao Objects**

Action.java

package com.hommy.administrator.dao.objects;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Action {

private int id;

private String action;

private int executionOrder;

public Action(int id, String action) {

this.id = id;

this.action = action;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getAction() {

return action;

}

public void setAction(String action) {

this.action = action;

}

public int getExecutionOrder() {

return executionOrder;

}

public void setExecutionOrder(int executionOrder) {

this.executionOrder = executionOrder;

}

@Override

public String toString() {

return "Action [id=" + id + ", action=" + action + "]";

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

Device.java

package com.hommy.administrator.dao.objects;

import java.util.List;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Device {

private int id;

private String name;

private String macAddress;

private Protocol protocol;

private List<Action> actions;

public Device(int id) {

this.id = id;

}

public Device(String name) {

this.name = name;

protocol = Protocol.HTTPS;

}

public Device(int id, String name) {

this.id = id;

this.name = name;

protocol = Protocol.HTTPS;

}

public Device(int id, String name, String macAddress) {

this.id = id;

this.name = name;

this.macAddress = macAddress;

}

public Device(int id, String name, Protocol protocol) {

this.id = id;

this.name = name;

this.protocol = protocol;

}

public Device(int id, String name, Protocol protocol, String macAddress) {

this.id = id;

this.name = name;

this.protocol = protocol;

this.macAddress = macAddress;

}

public Device(int id, String deviceName, List<Action> actions) {

this.id = id;

this.name = deviceName;

this.actions = actions;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Protocol getProtocol() {

return protocol;

}

public void setProtocol(Protocol protocol) {

this.protocol = protocol;

}

public String getMacAddress() {

return macAddress;

}

public void setMacAddress(String macAddress) {

this.macAddress = macAddress;

}

public List<Action> getActions() {

return actions;

}

public void setActions(List<Action> actions) {

this.actions = actions;

}

@Override

public String toString() {

return "Device [id=" + id + ", name=" + name + ", macAddress=" + macAddress + ", protocol=" + protocol + "]";

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

Event.java

package com.hommy.administrator.dao.objects;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Event {

private int id;

private String name;

private String description;

private String triggeredBy;

public Event() {

}

public Event(String name) {

this.name = name;

}

public Event(int id, String name, String description) {

this.id = id;

this.name = name;

this.description = description;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getTriggeredBy() {

return triggeredBy;

}

public void setTriggeredBy(String triggeredBy) {

this.triggeredBy = triggeredBy;

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

Rule.java

package com.hommy.administrator.dao.objects;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Rule {

private int id;

private String name;

private User user;

private Event event;

private Devices devices;

private Action action;

public Rule() {

}

public Rule(String name) {

this.name = name;

}

public Rule(int id, String name) {

this.id = id;

this.name = name;

}

public Rule(int id, String name, User user, Event event, Devices device, Action action) {

this.id = id;

this.name = name;

this.user = user;

this.event = event;

this.devices = device;

this.action = action;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Event getEvent() {

return event;

}

public void setEvent(Event event) {

this.event = event;

}

public Devices getDevices() {

return devices;

}

public void setDevices(Devices devices) {

this.devices = devices;

}

public Action getAction() {

return action;

}

public void setAction(Action action) {

this.action = action;

}

@Override

public String toString() {

return "Rule [name=" + name + ", id=" + id + ", device=" + devices + ", action=" + action + "]";

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

User.java

package com.hommy.administrator.dao.objects;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class User {

private int id;

private String username;

private String password;

private String email;

public User(int id, String username, String password, String email) {

this.id = id;

this.username = username;

this.password = password;

this.email = email;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

@Override

public String toString() {

return "User [username=" + username + ", email=" + email + "]";

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

**Dao Classes**

ActionOperations.java

package com.hommy.administrator.dao.impl;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import com.hommy.administrator.dao.interfaces.IAction;

import com.hommy.administrator.dao.objects.Action;

import com.hommy.database.DatabaseManager;

public class ActionsOperation implements IAction {

@Override

public List<Action> getActions(List<Integer> Actionsid) {

if (Actionsid.size() == 0) {

return new ArrayList<Action>();

}

Connection connection = DatabaseManager.createConnection();

List<Action> actions = new ArrayList<Action>();

StringBuilder query = new StringBuilder("select \* from actions where id IN (");

PreparedStatement preparedStatement;

try {

for (int i = 0; i < Actionsid.size(); i++) {

query.append("?,");

}

int lastindex = query.lastIndexOf(",");

query.replace(lastindex, lastindex + 1, ")");

preparedStatement = connection.prepareStatement(query.toString());

int index = 1;

for (Integer id : Actionsid) {

preparedStatement.setInt(index++, id);

}

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("id");

String action = resultSet.getString("action");

System.out.println("Action: " + id + " " + action);

actions.add(new Action(id, action));

}

} catch (SQLException e) {

e.printStackTrace();

}

return actions;

}

@Override

public boolean addAction(Action action) {

Connection connection = DatabaseManager.createConnection();

int result = 0;

String insertSQL = "INSERT INTO actions (action) VALUES (?)";

try {

PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

preparedStatement.setString(1, action.getAction());

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 1) {

return true;

}

return false;

}

}

DeviceOperation.java

package com.hommy.administrator.dao.impl;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import com.hommy.administrator.dao.interfaces.IDevice;

import com.hommy.administrator.dao.objects.Action;

import com.hommy.administrator.dao.objects.Device;

import com.hommy.administrator.dao.objects.Devices;

import com.hommy.database.DatabaseManager;

public class DevicesOperation implements IDevice {

static {

DatabaseManager.initialize();

}

@Override

public Devices getAllDevice() {

Connection connection = DatabaseManager.createConnection();

Devices devices = new Devices();

String query = "select \* from devices";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("Id");

String deviceName = resultSet.getString("deviceName");

String communicationProtocol = resultSet.getString("communicationprotocol");

System.out.println("Device: " + id + " " + deviceName + " " + communicationProtocol);

Device device = getDevice(deviceName);

devices.addDevice(device);

}

} catch (SQLException e) {

e.printStackTrace();

}

return devices;

}

@Override

public Device getDevice(String name) {

Connection connection = DatabaseManager.createConnection();

String query = "select \* from devices where devicename=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setString(1, name);

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

int id = resultSet.getInt("Id");

String deviceName = resultSet.getString("deviceName");

String communicationProtocol = resultSet.getString("communicationprotocol");

List<Integer> actionsId = getDeviceActions(connection, id);

List<Action> actions = new ActionsOperation().getActions(actionsId);

System.out.println("Event: " + id + " " + deviceName + " " + communicationProtocol);

return new Device(id, deviceName, actions);

} else {

System.out.println("Device with name: " + name + " does not exist");

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

@Override

public boolean addDevice(Device device) {

Connection connection = DatabaseManager.createConnection();

int result = 0;

String insertSQL = "INSERT INTO devices (devicename, communicationprotocol) VALUES (?, ?)";

try {

PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

preparedStatement.setString(1, device.getName());

preparedStatement.setString(2, device.getProtocol().toString());

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 1) {

return true;

}

return false;

}

@Override

public boolean removeDevice(int id) {

Connection connection = DatabaseManager.createConnection();

int result = 0;

String query = "delete from devices where id=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setInt(1, id);

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 0) {

return false;

}

return true;

}

private List<Integer> getDeviceActions(Connection connection, int deviceId) throws SQLException {

String query = "select actionId from DEVICEACTIONS where deviceID=?";

List<Integer> actionsID = new ArrayList<Integer>();

PreparedStatement preparedStatement;

preparedStatement = connection.prepareStatement(query);

preparedStatement.setInt(1, deviceId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("actionId");

actionsID.add(id);

}

return actionsID;

}

}

EventsOperation.java

package com.hommy.administrator.dao.impl;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import com.hommy.administrator.dao.interfaces.IEvent;

import com.hommy.administrator.dao.objects.Event;

import com.hommy.administrator.dao.objects.Events;

import com.hommy.database.DatabaseManager;

public class EventsOperation implements IEvent {

static {

DatabaseManager.initialize();

}

@Override

public Events getAllEvents() {

Connection connection = DatabaseManager.createConnection();

Events events = new Events();

String query = "select \* from events";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("Id");

String eventName = resultSet.getString("eventname");

String description = resultSet.getString("description");

System.out.println("Event: " + id + " " + eventName + " " + description);

events.addEvent(new Event(id, eventName, description));

}

} catch (SQLException e) {

e.printStackTrace();

}

return events;

}

public Event getEvent(Connection connection, int id) {

String query = "select \* from events where id=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setInt(1, id);

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

String eventName = resultSet.getString("eventname");

String description = resultSet.getString("description");

System.out.println("Event: " + id + " " + eventName + " " + description);

return new Event(id, eventName, description);

} else {

System.out.println("Event with this Id: " + id + " does not exist");

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

@Override

public Event getEvent(String name) {

Connection connection = DatabaseManager.createConnection();

String query = "select \* from events where eventname=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setString(1, name);

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

int id = resultSet.getInt("Id");

String eventName = resultSet.getString("eventname");

String description = resultSet.getString("description");

System.out.println("Event: " + id + " " + eventName + " " + description);

return new Event(id, eventName, description);

} else {

System.out.println("Event with name: " + name + " does not exist");

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

@Override

public boolean addEvent(Event event) {

Connection connection = DatabaseManager.createConnection();

int result = 0;

String insertSQL = "INSERT INTO Events (eventname, description, triggeredBy) VALUES (?, ?, ?)";

try {

PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

preparedStatement.setString(1, event.getName());

preparedStatement.setString(2, event.getDescription());

preparedStatement.setString(3, event.getTriggeredBy());

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 1) {

return true;

}

return false;

}

@Override

public boolean removeEvent(int id) {

Connection connection = DatabaseManager.createConnection();

int result = 0;

String query = "delete from events where id=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setInt(1, id);

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 0) {

return false;

}

return true;

}

}

RulesOperation.java

package com.hommy.administrator.dao.impl;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import com.hommy.administrator.dao.interfaces.IRule;

import com.hommy.administrator.dao.objects.Action;

import com.hommy.administrator.dao.objects.Device;

import com.hommy.administrator.dao.objects.Devices;

import com.hommy.administrator.dao.objects.Event;

import com.hommy.administrator.dao.objects.Rule;

import com.hommy.administrator.dao.objects.RuleDao;

import com.hommy.administrator.dao.objects.Rules;

import com.hommy.database.DatabaseManager;

public class RulesOperation implements IRule {

static {

DatabaseManager.initialize();

}

@Override

public Rules getAllRules() {

Connection connection = DatabaseManager.createConnection();

List<Rule> rules = new ArrayList<Rule>();

Rules rulesManage = new Rules();

String query = "select \* from rules";

Statement statement;

try {

statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(query);

System.out.println("All rules: ");

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("rulename");

int eventId = resultSet.getInt("eventId");

int deviceId = resultSet.getInt("deviceId");

int actionId = resultSet.getInt("actionId");

int executionOrder = resultSet.getInt("executionOrder");

System.out.println("Rule: " + id + " " + name);

}

} catch (SQLException e) {

e.printStackTrace();

}

rulesManage.setRules(rules);

return rulesManage;

}

public void getRuleForUserAndDevice(int userId, int DeviceId) {

Connection connection = DatabaseManager.createConnection();

Rules rulesManage = new Rules();

String query = "select \* from rules where userId=? and deviceId=?";

Statement statement;

try {

statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(query);

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("rulename");

int eventId = resultSet.getInt("eventId");

int deviceId = resultSet.getInt("deviceId");

int actionId = resultSet.getInt("actionId");

int executionOrder = resultSet.getInt("executionOrder");

System.out.println("Rule: " + id + " " + name + " " + deviceId + " " + actionId);

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public Rule getRuleByName(String name) {

Connection connection = DatabaseManager.createConnection();

Devices devices = new Devices();

Rule rule = new Rule();

String query = "select \* from rules where rulename=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setString(1, name);

ResultSet resultSet = preparedStatement.executeQuery();

boolean isFirstIteration = true;

List<Integer> actionsId = new ArrayList<Integer>();

while (resultSet.next()) {

if (isFirstIteration) {

int id = resultSet.getInt("id");

String rulename = resultSet.getString("rulename");

int eventId = resultSet.getInt("eventId");

rule.setId(id);

rule.setName(rulename);

Event event = new EventsOperation().getEvent(connection, eventId);

rule.setEvent(event);

}

int deviceId = resultSet.getInt("deviceId");

int actionId = resultSet.getInt("actionId");

devices.addDevice(new Device(deviceId));

actionsId.add(actionId);

}

rule.setDevices(devices);

} catch (SQLException e) {

e.printStackTrace();

}

return rule;

}

public Rules getRulesNameForUser(int userId) {

Connection connection = DatabaseManager.createConnection();

Rules rulesManage = new Rules();

String query = "select UNIQUE ruleName from rules where userId=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setInt(1, userId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

String name = resultSet.getString("rulename");

System.out.println("Rule: " + name);

rulesManage.addRule(new Rule(name));

}

} catch (SQLException e) {

e.printStackTrace();

}

return rulesManage;

}

@Override

public Rule getRule(Rule rule) {

// TODO Auto-generated method stub

return null;

}

@Override

public boolean addRule(Rule rule) {

Connection connection = DatabaseManager.createConnection();

String ruleName = rule.getName();

int userId = rule.getUser().getId();

int eventId = rule.getEvent().getId();

for (Device device : rule.getDevices().getDevices()) {

int deviceId = device.getId();

for (Action action : device.getActions()) {

int actionId = action.getId();

int executionOrder = action.getExecutionOrder();

boolean isAddedSuccessfully = addRuleDeviceActions(connection, userId, eventId, deviceId, actionId,

ruleName, executionOrder);

if (!isAddedSuccessfully) {

return false;

}

}

}

return true;

}

public boolean addRuleDao(RuleDao rule) {

Connection connection = DatabaseManager.createConnection();

String insertSQL = "INSERT INTO RULES(USERID, EVENTID, DEVICEID, ACTIONID, RULENAME, EXECUTIONORDER) VALUES (?, ?, ?, ?, ?, ?)";

int result = 0;

try {

PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

preparedStatement.setInt(1, rule.getUserId());

preparedStatement.setInt(2, rule.getEventId());

preparedStatement.setInt(3, rule.getDeviceId());

preparedStatement.setInt(4, rule.getActionId());

preparedStatement.setString(5, rule.getRuleName());

preparedStatement.setInt(6, 1);

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 1) {

return true;

}

return false;

}

public boolean addRuleDeviceActions(Connection connection, int userId, int eventId, int deviceId, int actionId,

String ruleName, int executionOrder) {

String insertSQL = "INSERT INTO RULES(USERID, EVENTID, DEVICEID, ACTIONID, RULENAME, EXECUTIONORDER) VALUES (?, ?, ?, ?, ?, ?)";

int result = 0;

try {

PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

preparedStatement.setInt(1, userId);

preparedStatement.setInt(2, eventId);

preparedStatement.setInt(3, deviceId);

preparedStatement.setInt(4, actionId);

preparedStatement.setString(5, ruleName);

preparedStatement.setInt(6, executionOrder);

result = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (result == 1) {

return true;

}

return false;

}

}

UserOperations.java

package com.hommy.administrator.dao.impl;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import com.hommy.administrator.dao.interfaces.IUser;

import com.hommy.administrator.dao.objects.User;

import com.hommy.database.DatabaseManager;

public class UsersOperation implements IUser {

static {

DatabaseManager.initialize();

}

@Override

public User getUser(String username, String password) {

Connection connection = DatabaseManager.createConnection();

String query = "select \* from users where username=? and password=?";

PreparedStatement preparedStatement;

try {

preparedStatement = connection.prepareStatement(query);

preparedStatement.setString(1, username);

preparedStatement.setString(2, password);

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

int id = resultSet.getInt("id");

String email = resultSet.getString("email");

System.out.println("User: " + resultSet.getString("username") + " " + resultSet.getString("password")

+ " " + email);

return new User(id, username, password, email);

} else {

System.out.println("User does not exist");

return null;

}

} catch (SQLException e) {

e.printStackTrace();

return null;

}

}

}

**Services**

HommyService.java

package com.hommy.service;

import java.io.IOException;

import javax.ws.rs.Consumes;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import com.fasterxml.jackson.core.JsonParseException;

import com.fasterxml.jackson.databind.JsonMappingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.hommy.administrator.dao.impl.DevicesOperation;

import com.hommy.administrator.dao.objects.Device;

import com.hommy.administrator.service.OperationResult;

import com.hommy.administrator.service.Result;

import com.hommy.service.subscribers.SmartPhone;

@Path("/service")

public class HommyService {

static {

new RegisterDevice().register(new SmartPhone(), TopicsPool.getTopic("Notification"));

}

@Path("/register")

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response registerDevice(final String device) throws JsonParseException, JsonMappingException, IOException {

ObjectMapper mapper = new ObjectMapper();

boolean isAddedSuccessfully = false;

if (device != null) {

Device newDevice = mapper.readValue(device, Device.class);

isAddedSuccessfully = new DevicesOperation().addDevice(newDevice);

// register

}

if (isAddedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

@Path("/unregister")

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response unregisterDevice(final String device) throws JsonParseException, JsonMappingException, IOException {

ObjectMapper mapper = new ObjectMapper();

boolean isAddedSuccessfully = false;

if (device != null) {

Device newDevice = mapper.readValue(device, Device.class);

isAddedSuccessfully = new DevicesOperation().addDevice(newDevice);

}

if (isAddedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

@Path("/publish")

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response publish(final String publication) throws JsonParseException, JsonMappingException, IOException {

ObjectMapper mapper = new ObjectMapper();

boolean isPublishedSuccessfully = false;

if (publication != null) {

Publication newPublication = mapper.readValue(publication, Publication.class);

PublicationQueue.addPublication(newPublication);

isPublishedSuccessfully = true;

if (!PublicationDispatcher.isBusy()) {

PublicationDispatcher.executeImmediately();

}

}

if (isPublishedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

@Path("/request")

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response receiveRequests(final String device) throws JsonParseException, JsonMappingException, IOException {

ObjectMapper mapper = new ObjectMapper();

boolean isAddedSuccessfully = false;

if (device != null) {

Device newDevice = mapper.readValue(device, Device.class);

isAddedSuccessfully = new DevicesOperation().addDevice(newDevice);

}

if (isAddedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

}

DeviceService.java

package com.hommy.administrator.service;

import java.io.IOException;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import com.fasterxml.jackson.core.JsonParseException;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.JsonMappingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.hommy.administrator.dao.impl.DevicesOperation;

import com.hommy.administrator.dao.objects.Device;

import com.hommy.administrator.dao.objects.Devices;

@Path("/devices")

public class DeviceService {

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getAllDevices() throws JsonProcessingException {

Devices devices = new DevicesOperation().getAllDevice();

String result = devices.toJSON();

return Response.status(200).entity(result).build();

}

@Path("/device")

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getDevice(@QueryParam("name") String name) throws JsonProcessingException {

Device device = null;

if (name != null) {

device = new DevicesOperation().getDevice(name);

}

if (device != null) {

return Response.status(200).entity(device.toJSON()).build();

} else {

return Response.status(200)

.entity(new OperationResult(Result.FAILED, "Could not find device with this name.").toJSON())

.build();

}

}

@Path("/remove")

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response removeDevice(@QueryParam("id") int id) throws JsonProcessingException {

boolean isRemovedSuccessfully = false;

if (id > 0) {

isRemovedSuccessfully = new DevicesOperation().removeDevice(id);

}

if (isRemovedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200)

.entity(new OperationResult(Result.FAILED, "Could not find device with this id.").toJSON()).build();

}

}

@Path("/add")

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response addDevice(final String device) throws JsonParseException, JsonMappingException, IOException {

ObjectMapper mapper = new ObjectMapper();

boolean isAddedSuccessfully = false;

if (device != null) {

Device newDevice = mapper.readValue(device, Device.class);

isAddedSuccessfully = new DevicesOperation().addDevice(newDevice);

}

if (isAddedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

}

EventService.java

package com.hommy.administrator.service;

import java.io.IOException;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import com.fasterxml.jackson.core.JsonParseException;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.JsonMappingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.hommy.administrator.dao.impl.EventsOperation;

import com.hommy.administrator.dao.objects.Event;

import com.hommy.administrator.dao.objects.Events;

@Path("/events")

public class EventsService {

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getAllEvents() throws JsonProcessingException {

Events events = new EventsOperation().getAllEvents();

String result = events.toJSON();

return Response.status(200).entity(result).build();

}

@Path("/event")

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getEvent(@QueryParam("name") String name) throws JsonProcessingException {

Event event = null;

if (name != null) {

event = new EventsOperation().getEvent(name);

}

if (event != null) {

return Response.status(200).entity(event.toJSON()).build();

} else {

return Response.status(200)

.entity(new OperationResult(Result.FAILED, "Could not find an event with this name").toJSON())

.build();

}

}

@Path("/remove")

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response removeEvent(@QueryParam("id") int id) throws JsonProcessingException {

boolean isRemovedSuccessfully = false;

if (id > 0) {

isRemovedSuccessfully = new EventsOperation().removeEvent(id);

}

if (isRemovedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

@Path("/add")

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response addEvent(final String event) throws JsonParseException, JsonMappingException, IOException {

ObjectMapper mapper = new ObjectMapper();

boolean isAddedSuccessfully = false;

if (event != null) {

Event newEvent = mapper.readValue(event, Event.class);

isAddedSuccessfully = new EventsOperation().addEvent(newEvent);

}

if (isAddedSuccessfully) {

return Response.status(200).entity(new OperationResult(Result.OK).toJSON()).build();

} else {

return Response.status(200).entity(new OperationResult(Result.FAILED).toJSON()).build();

}

}

}

**Hommy Utils**

HTTPUtil.java

package com.hommy.utils;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import org.apache.http.HttpResponse;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.HttpClientBuilder;

import com.fasterxml.jackson.databind.ObjectMapper;

public class HTTPUtil {

public static boolean sendMessage(Message message) {

HttpClient client = HttpClientBuilder.create().build();

HttpPost post = new HttpPost(HommyUtil.GOOGLE\_CLOUD\_MESSAGING\_URI);

post.addHeader("Authorization", HommyUtil.AUTHORIZATION\_GOOGLE\_SERVER\_KEY);

post.addHeader("Content-Type", HommyUtil.CONTENT\_TYPE\_JSON);

StringEntity postingString;

StringBuffer result = new StringBuffer();

try {

postingString = new StringEntity(message.toJSON());

System.out.println(message.toJSON());

post.setEntity(postingString);

HttpResponse httpResponse = client.execute(post);

if (httpResponse.getStatusLine().getStatusCode() == HommyUtil.HTTP\_STATUS\_CODE\_OK) {

BufferedReader rd = new BufferedReader(new InputStreamReader(httpResponse.getEntity().getContent()));

String line = "";

while ((line = rd.readLine()) != null) {

result.append(line);

}

ObjectMapper mapper = new ObjectMapper();

MessageResponse response = mapper.readValue(result.toString(), MessageResponse.class);

return isSuccessfullyReceived(response);

} else {

System.out.println("Response Code : " + httpResponse.getStatusLine().getStatusCode());

}

} catch (IOException e) {

e.printStackTrace();

}

return false;

}

private static boolean isSuccessfullyReceived(MessageResponse response) {

int numberOfNotifiedDevices = response.getSuccess();

int numberOfNOTNotifiedDevices = response.getFailure();

if (numberOfNotifiedDevices > 0 && numberOfNOTNotifiedDevices == 0) {

return true;

}

return false;

}

}

MessageResponse.java

package com.hommy.utils;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class MessageResponse {

private String multicast\_id;

private int success;

private int failure;

private String canonical\_ids;

private Object[] results;

public String getMulticast\_id() {

return multicast\_id;

}

public void setMulticast\_id(String multicast\_id) {

this.multicast\_id = multicast\_id;

}

public int getSuccess() {

return success;

}

public void setSuccess(int success) {

this.success = success;

}

public int getFailure() {

return failure;

}

public void setFailure(int failure) {

this.failure = failure;

}

public String getCanonical\_ids() {

return canonical\_ids;

}

public void setCanonical\_ids(String canonical\_ids) {

this.canonical\_ids = canonical\_ids;

}

public Object[] getResults() {

return results;

}

public void setResults(Object[] results) {

this.results = results;

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

MessageData.java

package com.hommy.utils;

import java.util.HashMap;

import java.util.Map;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class MessageData {

private Map<String, String> data = new HashMap<String, String>();

public MessageData() {

}

public boolean add(String key, String value) {

if (data.containsKey(key)) {

return false;

} else {

data.put(key, value);

return true;

}

}

public boolean remove(String key, String value) {

if (data.containsKey(key)) {

data.remove(key, value);

return true;

} else {

return false;

}

}

public boolean remove(String key) {

if (data.containsKey(key)) {

data.remove(key);

return true;

} else {

return false;

}

}

public Map<String, String> getData() {

return data;

}

public void setData(Map<String, String> data) {

this.data = data;

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

Message.java

package com.hommy.utils;

import java.util.HashMap;

import java.util.Map;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class Message {

private String to;

private Map<String, String> data = new HashMap<String, String>();

public Message() {

}

public Message(String to) {

this.to = to;

}

public String getTo() {

return to;

}

public void setTo(String to) {

this.to = to;

}

public boolean add(String key, String value) {

if (data.containsKey(key)) {

return false;

} else {

data.put(key, value);

return true;

}

}

public boolean remove(String key, String value) {

if (data.containsKey(key)) {

data.remove(key, value);

return true;

} else {

return false;

}

}

public boolean remove(String key) {

if (data.containsKey(key)) {

data.remove(key);

return true;

} else {

return false;

}

}

public Map<String, String> getData() {

return data;

}

public void setData(Map<String, String> data) {

this.data = data;

}

public String toJSON() throws JsonProcessingException {

ObjectMapper mapper = new ObjectMapper();

return mapper.writeValueAsString(this);

}

}

**Publish-Subscribe**

ITopic.java

package com.hommy.service.interfaces;

public interface ITopic {

/\*\*

\* This methods is for register new device.

\*

\* @param device

\*/

public boolean register(IDeviceSubscriber device);

/\*\*

\* This methods is for unregister device.

\*

\* @param device

\*/

public boolean unregister(IDeviceSubscriber device);

/\*\*

\* This method is used to notify the subscribers of change

\*/

public void notifySubscribers();

/\*\*

\* This method is used to get updates from topic

\*

\* @param device

\* @return

\*/

public Object getUpdate(IDeviceSubscriber device);

/\*\*

\* This method is used to publish message to the topic

\*

\* @param message

\*/

public void publishMessage(Object message);

}

Topic.java

package com.hommy.service.topics;

import java.util.ArrayList;

import java.util.List;

import com.hommy.service.interfaces.IDeviceSubscriber;

import com.hommy.service.interfaces.ITopic;

public abstract class Topic implements ITopic {

private List<IDeviceSubscriber> subscribers = new ArrayList<IDeviceSubscriber>();

private final Object MUTEX = new Object();

private final String topicName;

private String message;

private boolean isChanged;

public Topic(String topicName) {

this.topicName = topicName;

}

public String getTopicName() {

return topicName;

}

public List<IDeviceSubscriber> getSubscribers() {

return subscribers;

}

public String getMessage() {

return message;

}

public boolean isChanged() {

return isChanged;

}

@Override

public boolean register(IDeviceSubscriber device) {

if (device != null) {

synchronized (MUTEX) {

if (!subscribers.contains(device))

return subscribers.add(device);

}

}

return false;

}

@Override

public boolean unregister(IDeviceSubscriber device) {

if (device != null) {

synchronized (MUTEX) {

return subscribers.remove(device);

}

}

return false;

}

@Override

public void notifySubscribers() {

List<IDeviceSubscriber> observersLocal = null;

// synchronization is used to make sure any observer registered after

// message is received is not notified

synchronized (MUTEX) {

if (!isChanged)

return;

observersLocal = new ArrayList<>(this.subscribers);

this.isChanged = false;

}

for (IDeviceSubscriber device : observersLocal) {

device.update();

}

}

@Override

public Object getUpdate(IDeviceSubscriber device) {

return this.message;

}

@Override

public void publishMessage(Object message) {

System.out.println("Message Published to Topic:" + message);

this.message = (String) message;

this.isChanged = true;

notifySubscribers();

}

}

IDeviceSubscriber.java

package com.hommy.service.interfaces;

public interface IDeviceSubscriber {

/\*\*

\* This method updates the subscriber, used by topics

\*/

public void update();

/\*\*

\* This method is used to subscribe to a specific topic to observe

\*

\* @param topic

\*/

public void setTopic(ITopic topic);

}

DeviceSubscriber.java

package com.hommy.service.subscribers;

import com.hommy.service.interfaces.IDeviceSubscriber;

import com.hommy.service.interfaces.ITopic;

public abstract class DevcieSubscriber implements IDeviceSubscriber {

private String deviceName;

private ITopic topic;

public DevcieSubscriber(String deviceName) {

this.deviceName = deviceName;

}

public String getDeviceName() {

return deviceName;

}

public void setDeviceName(String deviceName) {

this.deviceName = deviceName;

}

public ITopic getTopic() {

return topic;

}

@Override

public void setTopic(ITopic topic) {

this.topic = topic;

}

}