**Съдържание**

1. Въведение **4**
2. Анализ на темата. Цели и задачи на приложението **5**
   1. Цели **6**
   2. Задачи **7**
   3. Анализ **8**
3. Обзор на използваните технологии **9**
   1. ASP.NET Core **13**
   2. MVC **12**
   3. Entity Framework Core **10**
   4. Microsoft SQL Server **11**
   5. Flutter **12**
   6. JavaScript **13**
   7. jQuery **14**
   8. HTML **15**
   9. CSS **16**
4. Проектиране и архитектура на приложението **17**
   1. Логическа архитектура на уеб приложението **18**
   2. Логическа архитектура на мобилното приложение **19**
   3. Архитектура на базата данни **19**
5. Програмна реализация **20**
   1. Авторизация и автентикация **21**
   2. Комуникация с базата данни **22**
   3. Запис в системата **23**
   4. Мобилно приложение  **24**
6. Ръководство на потребителя **25**
   1. Начална страница **26**
   2. Регистрация **27**
   3. Вход **28**
   4. Магазин за сензори **29**
   5. Добавяне на потребителски сензор **30**
   6. Преглед на потребителските сензори **31**
   7. Редакция на потребителски сензор **32**
   8. Изтриване на потребителски сензор **33**
   9. Преглед на потребителските сензори на картите **34**
   10. Известия **35**
7. Експериментални резултати
   1. Достъп до чужди записи **36**
   2. Прихващане на изключения **37**
8. Използвана литература **38**

**Въведение**

В днешно време тенденциите в разработването на програмните системи са насочени все повече към към уеб технологиите. Една от основните причини за това е, че уеб приложенията са достъпни от всяка точка на света. При тях липсва необходимостта от инсталиране на локалната машина на потребителя, а е достатъчен уеб браузър. Това ги прави много удобни и лесно приложими. С помощта на уеб базираните информационни системи могат лесно да се организират и поддържат различни управленски, комуникационни и др. дейности.

В следващите страници ще бъде представено изграждането на онлайн система за управление и преглед на сензори. В това забързано ежедневие, което водим в наши дни хората...

Разработката на онлайн системата за управление и преглед на сензори е представена в следните глави:

* Във втора глава са разгледани целите и ползите от приложението. Описва се какво има направено до този момент като приложения и програми.
* В трета глава са описани използваните технологии – езици за програмиране, бази данни и др.
* В четвърта глава са разгледани архитектурите на – уеб приложението, мобилното приложение и базата данни.
* В пета глава се разглеждат някои по-важни аспекти от кода, за какво служи той и начините по които се реализира дадената функционалност.
* В шеста глава е показано подробно ръководство на потребителя.
* В седма глава са разгледани експериментални резултати от срещани проблеми и прихващане на грешки.

**Анализ на темата. Цели и задачи на приложението**

**Цели**

В това забързано ежедневие, в което живеем днес е от значение всяка секунда време, което отделяме. Затова системата е предназначена да улесни информирането и известяването на потребителя за закупените от него сензори като предоставя данни за тях в реално време на едно централизирано място.

**Основни задачи**

1. Приложението трябва да е лесно и интуитивно за употреба.
2. Да осигурява възможност за бърза регистрация и автентикация.
3. Да предоставя лесен начин за добавяне/премахване на сензори от магазина.
4. Да предоставя лесен начин за „закупуване“/премахване сензори от всеки потребител.
5. Да предоставя лесен начин за преглед на закупените сензори от потребителя на карта.
6. Да предоставя лесно видими данни за всеки един закупен сензор от потребителя в реално време.
7. Да има възможност за допълнително лесно разширяване и обновяване на същвствуващите функции, както и добавяне на нови функционалности.

**Анализ**

**Datum Platform Interactive**

 Системата Platform Interactive на Datum е софтуер, чрез който клиентите могат да получават информация за устройствата на клиентите.

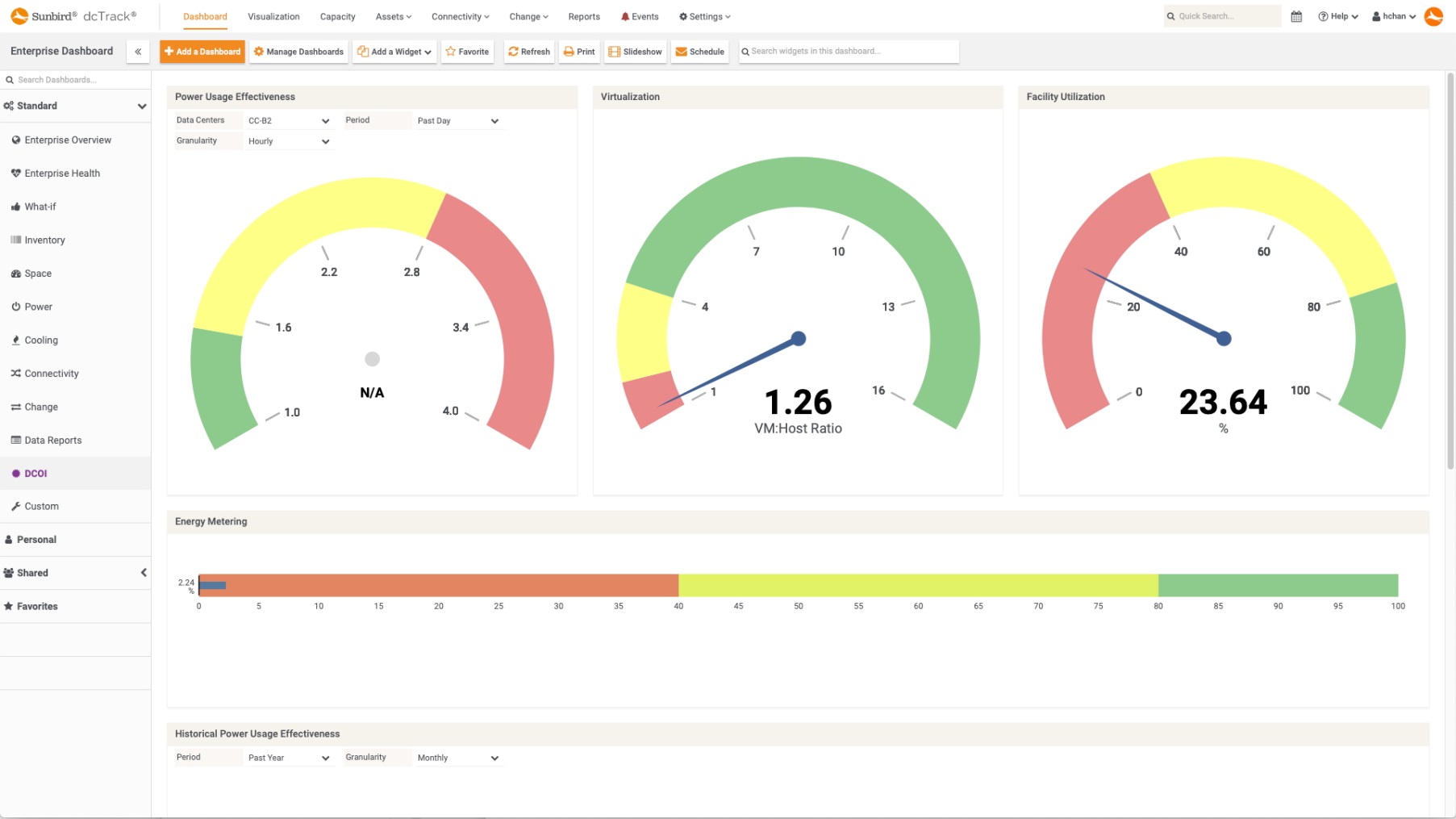
Част от предлаганите функционалности:

* Системата им е изцяло онлайн, което означава, че не са необходими никакви хардуерни действия от хора, които да инсталират или монтират системата.
* Системата им позволява скалиране, чрез повишаване/намаляване на плана, за да се предостави пълноценно потребителско изживяване.
* Системата им разполага с различни видове известяване – СМС, имейл, телефонно обаждане.

**Sunbird Energy**

Системата Energy на Sunbird предоставя софтуер за преглед на данни от сензори в реално време с цел измерване на ефективност, пестене и редуциране на използваната енергия.

Част от предлаганите функционалности:

* Възможност за измерване и мониторинг на данните от сензорите
* Информация за сензорите в реално време
* Добре разработен и лесен за употреба визуален интерфейс

След анализ на двете представени системи можем да направим следните изводи:

* Подобни приложения вече съществуват, което не е учудващо предвид масовостта на идеологията за контролиране на техниката от едно централизирано място. Всяко от тях е уникално и предоставя различни допълнителни функционалности
* Sunbird Energy има разработено десктоп приложение, докато Datum Platform Interactive е онлайн решение
* Datum Platform Interactive предлага различни видове известявания, които липсват при Sunbird Energy

**Обзор на използваните технологии**

**ASP.NET Core**

ASP.NET Core е безплатна софтуерна рамка за уеб разработка с отворен код. Също така тя се явява и следващата стъпка в еволюцията на ASP.NET. Тя е разработена съвместно от Microsoft и общността, която е събрала през годините на своето развитие. ASP.NET Core е модулярна софтуерна рамка, която върви на крос-платформената .NET Core. Въпреки, че е нова софтуерна рамка изградена върху нов уеб стек, тя има висока степен на съвместимост и прилики с ASP.NET.

Спрямо своите предшественици ASP.NET Core поддържа нова функция – т.нар. “side by side versioning”. При нея различни приложения, които използват една и съща машина могат да таргетират различни версии на ASP.NET Core, в зависимост от версиите (и нуждите) си. Това не е възможно с по-стари издания на ASP.NET.

Съвместим е с Windows, MacOS и Linux, което го прави гъвкав и удобен за използване. Подходящ е за употреба, тъй като има интегрирани множество от библиотеки и компоненти, които се изпълняват по време на работа и компилация.

В случая не само платформите са от голяма значимост, но и това, че разработваната система има нужда от висока производителност и надграждане. .NET Core и ASP.NET Core са най-добрата опция в случая, тъй като предоставят нужната производителност за стартиране и функциониране на сървърната част. Освен това, управляваното време за изпълнение улеснява разработката, garbage-collection-а и гарантира безопасно изпълнение.

.NET Core е набор от изпълними, библиотечни и компилационни компоненти, които могат да се иползват в различни конфигурации за работа на устройства и в облака. Междуплатформено и с отворен код .NET Core предоставя олекотен модел за разработване и гъвкавостта да се работи с различни инструменти за разработване и платформени ОС. .NET Core е достъпно в GitHub под лиценза MIT като включва няколко технологии - .NET Core, ASP.NET Core, Entity Framework Core и др.

**MVC**

Архитектурата на приложението е изградена върху архитектурния шаблон MVC (Model-View-Controller), който помага да се постигне разпределение на идеите. При използването на този шаблон заявките на потребителите се насочват към контролера, който отговаря за работата с модела, за да се извършат очакваните от потребителя действия или да се изведат резултати от заявката.



***Модел:***

Моделът представлява част от приложението, което реализира домейн логиката, също известна като бизнес логика. Домейн логиката обработва данните, които се предават между базата данни и потребителския интерфейс. Например, в една система за инвентаризация, моделът отговаря за това дали елемент от склада е наличен. Моделът може да бъде част от заявлението, което актуализира базата данни когато даден елемент е продаден или доставен в склада. Често моделът съхранява и извлича официална информация в базата данни.

***Изглед:***

Изгледите са тези, които определят как ще бъде визуализиран потребителският интерфейс (UI) на приложението. Характерното при тях е, че може да се включи C# код във файловете. Те са със специално разширение .cshtml и използват Razor Engine за компилация.

***Контролер:***

Контролерите са класове, които се създават в MVC приложението. Намират се в папка Controllers. Всеки един клас, който е от този тип, трябва да има име завършващо с наставка „Controller“. Контролерите обработват постъпващите заявки, въведени от потребителя и изпълняват подходящата логика за изпълнение на приложението.

**Entity Framework Core**

Entity Framework Core е лека, разширяема и многоплатформена версия на популярната технология за достъп и поддръжка на данни Entity Framework. Тя може да ни служи за създаване на връзка между обекти, позволявайки на разработчиците да работят с база от данни с .NET обекти и елиминирайки нуждата от голяма част от кода за достъп до данните, която би се наложило да напишат.

Има много начини, чрез които може да се проектира такава библиотека, EF Core е създадена като object-relational mapper (ORM). Този тип дизайн работи като създава връзката между две страни базата данни с API-то, което използва и обектно ориентирана софтуерна страна. Това е един от най-бързите начини, чрез които разработчиците на софтуер достъпват базата от данни бързо и лесно. Именно затова, EF Core се използва за разработката на текущото приложение – за достъп и контрол на база от данни, която съхранява данните на приложението.

**Microsoft SQL Server**

В същността си MS SQL Server е система за управление на релационни бази от данни. Като съвър, основната му функция е съхраняване и предоставяне на данни така, както са заявени от други приложения, които могат да са пуснати както на същата машина, така и през мрежа.

Основната причина за това MS SQL Server да е предпочитана платформа за разработка е това, че когато потребителите искат данни, те ги искат колкото може по-бързо. Таблиците са оптимизирани в използването на памет, а за натоварване при анализ на данните, потребителите могат да се възползват от обновяеми клъстерирани и индексирани колони за тези таблици за около 100 пъти по-бързи заявки.

Друг голям плюс на MS SQL Server е, че включва няколко инструмента в помощ за планиране на миграции на данните в съществуващите таблици. Сигурността също е добре планирана – на ниво ред, а данните са винаги криптирани. Това означава, че данните, пазни в облака не могат да бъдат разчетени от никого.

И за да затворим цикъла, MS SQL Server, в пълната си версия, може да бъде съхранен в облачното пространство без да трябва да се управлява какъвто и да е хардуер. Това става възможно благодарение на виртуалните машини на Azure, които са пуснати на много и различни географски региони по света. Освен това те имат разнообразие от размери (в памет) на машините. Така потребителите могат да създават виртуална машина със SQL Server, която да бъде с правилната версия и операционна система. Обновяването и поддръжката се случват автоматично, както и създаването на архив на състоянието на базата, така че да не се губи информация.

**Flutter**

Flutter е софтуерна рамка с отворен код създаден от Google през 2017, чрез който могат да се създават приложения за Android, iOS, Windows, Linux, MacOS и уеб като споделят една и съща версия на кода.

Архитектурата на Flutter се разделя на 4 категории:

1. ***Dart платформата*** – Flutter приложенията използват език за програмиране наречен Dart. Изпълняват се в Dart виртуална машина, която поддържа just-in-time компилация, което означава, че програмата може да се компилира в реално време докато работи, което позволява така наречената функция – “hot reload”, чрез която модификации на кода могат да бъдат заредени върху вече работещо приложение.
2. ***Flutter Engine*** – Flutter използва engine, който е написан основно на C++, като позволява достъп до ниско ниво на Google Skia графичната библиотека. Също така използва специфични интерфейси на Android и iOS SDK-та. Flutter Engine-а имплементира основните библиотеки на рамката, които включват анимации, графики, работа с файлове, достъп до мрежата и др.
3. ***Foundation library*** – библиотека наисана на Dart, която предоставя достъп до базови класове и функции, които се използват при създаването на Flutter приложения както и API-та, които комуникират с Engine-на.
4. ***Design***-specific widgets – Flutter приложенията използват така наречените уиджети при разработка. Основните типа са два – Material Design уиджети, които имплементират и следват насоките на Google за дизайн и разработка и Cupertino уиджети, които имплементират и следват насоките на Apple за дизайн и разработка.

**JavaScript**

JavaScript е интерпретируем език за програмиране, разпространяван с повечето уеб браузъри. Поддържа обектно-ориентиран и функционален стил на програмиране. Създаден е в Netscape през 1995 г. Най-често се прилага към HTML-а на Интернет страница с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптове JSON, както и за много други приложения. JavaScript не трябва да се бърка с Java, съвпадението на имената е резултат от маркетингово решение на Netscape. JavaScript е стандартизиран под името EcmaScript.

**jQuery**

jQuery е разпространена библиотека на JavaScript, публикувана в началото на 2006 от Джон Резиг. В основата си jQuery опростява достъпа до всеки елемент на дадена уеб-страница, като по този начин позволява лесно изграждане на динамична функционалност в страниците.

jQuery е безплатен и open source софтуер, лицензиран под MIT лиценз. jQuery се използва в 72% от 10000-те най-посещавани сайтове, което я прави най-популярната JavaScript алтернатива днес

**HTML**

HTML (съкращение от термина на английски: HyperText Markup Language, произнасяно най-често като „ейч-ти-ем-ел“, в превод „език за маркиране на хипертекст“) е основният маркиращ език за описание и дизайн на уеб страници. HTML е стандарт в Интернет. Последната версия е 5 от 2014 г.

Описанието на документа става чрез специални елементи, наречени HTML елементи или маркери, които се състоят от етикети или тагове (HTML tags) и ъглови скоби (като например елемента <html>). HTML елементите са основната градивна единица на уеб страниците. Чрез тях се оформят отделните части от текста на една уеб страница, като заглавия, цитати, раздели, хипертекстови препратки и т.н. Най-често HTML елементите са групирани по двойки <h1> и </h1>.

В повечето случаи HTML кодът е написан в текстови файлове и се хоства на сървъри, свързани към Интернет. Тези файлове съдържат текстово съдържание с маркери – инструкции за браузъра за това как да се показва текстът. Например <маркер> Някакъв текст. </край на маркера>. Предназначението на уеб браузърите е да могат да прочетат HTML документите и да ги превърнат в уеб страници. Браузърите не показват HTML таговете, а ги използват, за да интерпретират съдържанието на страницата.

Основното предимство на HTML е, че документите, оформени по този начин, могат да се разглеждат на различни устройства, а не само на екрана. Документът може да бъде правилно оформен и върху монитора на персонален компютър, и върху миниатюрния дисплей на пейджър или мобилен телефон.

HTML може да прикрепя скриптове писани на езици като JavaScript, което променя поведението на уеб страницата. Може да се използва Cascading Style Sheets (CSS), който определя изгледа и оформлението на текста и други материали. World Wide Web Consortium (W3C) поддържа и двете CSS и HTML и насърчава използването на CSS в HTML страниците от 1997. Това допринася за разделяне съдържанието и структурата на уеб страниците от тяхното визуално представяне

**CSS (Cascading Style Sheets)**

CSS (Cascading Style Sheets) е език за описание на стилове (език за стилови листове, style sheet language) – използва се основно за описване на представянето на документ, написан на език за маркиране. Най-често се използва заедно с HTML, но може да се приложи върху произволен XML документ. Официално спецификацията на CSS се поддържа от W3C.

CSS е създаден с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995 г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страницата. В резултат на това HTML кода се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един-единствен файл, и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл.

**Проектиране и архитектура на приложението**

**Логическа архитектура на уеб приложението**

Уеб приложението на системата за управление и преглед на сензори се състои от 4 основни части разпределени в различни проекти – ***App***, ***Models***, ***Services***, ***Common***

**App:** ASP.NET Core проект използващ MVC шаблон за архитектура. Съдържа следните компоненти:

1. ***Identity Area*** – Служи за автентикация на потребителите като използва Razor Pages архитектура.
   1. ***Register Page*** – Служи за регистрация на нови потребители в системата. Поддържа следните Action методи:
      1. ***OnGetAsync***
      2. ***OnPostAsync***
   2. ***Login Page*** – Служи за вход в системата на регистрираните потребители. Поддържа следните Action методи:
      1. ***OnGetAync***
      2. ***OnPostAsync***
   3. ***Logout Page***- Служи за изход/отписване на потребителя от системата. Поддържа следните Action методи:
      1. ***OnPostAsync***
   4. ***ConfirmEmail Page*** – Служи за потвърждение на акаунта при отваряне на линка изпратен на имейл адреса на потребителя при регистрация. Поддържа следните методи:
      1. ***OnGetAsync***
2. ***Controllers*** – Служат за управлението и връщането на данните.
   1. ***HomeController*** – Обслужва заявките на началната страница на приложението. Поддържа следните Action методи:
      1. ***Index***  - Връща HomeIndexViewModel модел с попълнени данни за визуализиране на изгледа.
      2. ***GetUserSensorCoordinates***- Връща JSON с данните за закупените потребителски сензори, които ще бъдат визуализирани с помощта на Google карти.
   2. ***SensorController*** – Обслужва заявките за CRUD операции на сензорите в магазина. Поддържа следните Action методи:
      1. ***GET/all*** – Връща всички налични сензори в базата данни.
      2. ***GET/{id}*** – Връща един сензор филтриран по id.
      3. ***GET/userdCound/{id}*** – Връща броя на потребители закупили сензора филтриран по id.
      4. ***POST*** – Служи за създаване на нов сензор
      5. ***PUT*** – Служи за обновяване на вече съществуващ сензор
      6. ***DELETE/{id}*** – Служи за изтриване на сензор филтриран по id
   3. ***SensorPropertyController*** – Обслужва заявките за CRUD операции на настройките на сензорите в магазина. Поддържа следните Action методи:
      1. ***GET/all*** – Връща всички налични настройки на сензорите в базата данни.
      2. ***GET/{id}*** – Връща една настройка за сензор филтрирана по id.
      3. ***GET/sensormeasuretypes*** – Връща всички налични видове сензори от базата данни.
      4. ***POST*** – Служи за създаване на нова настройка за сензор
      5. ***PUT*** – Служи за обновяване на вече съществуваща настройка за сензор
      6. ***DELETE/{id}*** – Служи за изтриване на настройка засензор филтрирана по id
   4. ***SensorDataController*** – Връща стойностите на всеки сензор от базата данни. Поддържа следните Action методи:
      1. ***GET/all***
   5. ***SensorStoreController***- Обслужва заявките за управление на сензорите в магазина. Поддържа следните Action методи:
      1. ***Index*** – Връща SensorStoreViewModel модел използван за визуализиране на сензорите в изгледа.
      2. ***ReloadSensorsTable*** – Служи за обновяване на страницата със сензори при прилагане на филтър.
   6. ***UserSensorController*** – Обслужва заявките за управление на закупените сензори от потребители. Поддържа следните Action методи:
      1. ***Index*** – Връща UserSensorIndexViewModel модел използван за визуализиране на потребителските сензори в изгледа.
      2. ***ReloadUserSensorsTable*** – Връща PartialView съдържащо HTML част показваща таблицата със закупените сензори. Поддържа филтриране и pagination.
      3. ***Create*** – Служи за създаване на потребителски сензор.
      4. ***Edit*** – Служи за редактиране на вече съществуващ потребителски сензор.
      5. ***GetGaugeData*** – Връща стойността на даден потребителски сензор филтриран по id от базата данни.
      6. ***Delete*** – Изтрива потребителски сензор филтриран по id.
      7. ***GetUserSensorCoordinates*** – Връща кооринадите на потребителските сензори, които ще бъдат визуализирани с помощта на Google карти.
   7. ***ErrorController*** - Служи за показване на страници за грешка при изключения или несъществуващи страници. Поддържа следните Action методи:
      1. ***Index*** – Използва се при грешки със статус код различен от 404.
      2. ***PageNotFound*** – Използва се при грешки със статус код 404.
3. ***Hubs*** – Служи за изпращане на съобщения до потребителския интерфейс използвайки SignalR сокет. Съдържа следните класове:
   1. ***SensorStoreHub*** – Използва се за изпращане на съобщения до всички автентикирани потребители при добавяне/изтриване на сензор от магазина.
   2. ***NotificationManager*** – Служи за контейнер, който съдържа наличните Hub-ове и изпраща съобщения през тях.
4. ***Views*** – Изгледи, които се използват за визуализация на потребителския интетрфейс. Разпределени са по папки като за всеки контролер има отделна папка, която съдържа .cshtml файлове с имената на всеки Action
5. ***Startup.cs*** – Служи за настройване на ASP.NET Core сървъра. Тук са регистрирани ендпойнтите, класовете за Dependency Injection, SignalR, Logging и др.
6. ***Program***.***cs*** – Стартовата точка на уеб приложението. Съдържа и стартиране на Logging provider-а.

**Models** - .NET Core class library съдържаща всички модели използвани от приложението. Разпределение по папки:

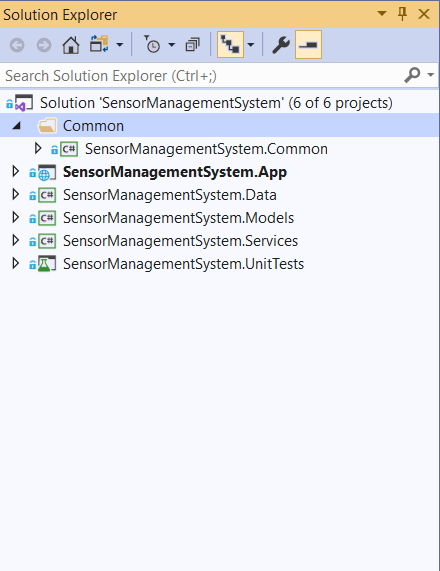
1. ***AutoMapper*** – Съдържа конфигурационен профил и разширения за Automapper библиотеката.
2. ***DTOs*** – Съдържа Data Transfer Objects използвани от Sensor, SensorProperty и SensorData API. Съдържа следните класове:
   1. ***SensorDataDTO*** – Използван от SensorDataController за трансфер на данни.
   2. ***SensorDTO*** – Използван от SensorController за трансфер на данни.
   3. ***SensorPropertyDTO*** – Използван от SensorPropertyController за трансфер на данни.
   4. ***UserSensorGaugeData*** – Използван от UserSensorController за трансфер на данни
3. ***Entities*** – Съдържа класовете използвани за писане/четене на данни от базата данни. Съдържа следните класове:
   1. ***BaseEntity*** – Базов клас съдържащ основни пропъртита.
   2. ***RoleEntity*** – Използва се за управление на потребителските роли.
   3. ***SensorEntity*** – Използва се за записване/четене на сензор обекти.
   4. ***SensorPropertyEntity*** – Използва се за записване/четене на настройки за сензор обекти.
   5. ***UserEntity*** – Използва се за достъп до потребителите.
   6. ***UserSensorEntity*** – Използва се за записване/четене на потребителски сензори.
4. ***ViewModels*** – Служат за показване на каква точно информация да бъде показана в изгледите. Съдържа следните класове:
   1. ***CreateUpdateUserSensorViewModel*** *–* Използва се при създаване/редакция на потребителски сензор
   2. ***HomeIndexViewModel* –** Използва се при показване на данните от началната страница.
   3. ***PaginatedList*** – Използва се като помощен клас за pagination.
   4. ***SensorPropertyViewModel*** – Използва се за показване на настройка на сензор
   5. ***SensorStoreViewModel*** – Използва се за показване на сензори в магазина
   6. ***SensorViewModel*** - Използва се при презареждане на сензорите в магазина.
   7. ***UserSensorIndexViewModel*** – Използва се за показване на закупените потребителски сензори.
   8. ***UserSensorMapViewModel*** – Използва се за показване на закупените потребителски сензори в Google карти.
   9. ***UserSensorTableViewModel*** – Използва се при таблицата с потребителски закупените сензори.
   10. ***UserSensorViewModel*** – Използва се при презареждане потребителски закупените сензори.

**Services** - .NET Core class library съдърщата бизнес логика относно операциите за четене/запис с базата данни. Съдържа следните класове и интерфейси:

1. ***IEmailService – EmailService*** – Използва се за генериране и изпращане на съобщения по даден имейл адрес.
2. ***SensorDataFetchHostedService*** - .NET Core Hosted Service, който работи на заден фон, докато уеб приложението е стартирано. Служи за актуализиране на стойностите на сензорите като прави API заявка към SensorDataController на всеки 10 секунди. След като получи данните за сензорите актуализира тези стойности на съответните потребителски сензори в базата данни.
3. ***ISensorDataService - SensorDataService*** – Връща всички сензори като генерира произволна стойност за всеки един от тях в рамките на неговия обхват и тип.
4. ***ISensorPropertyService – SensorPropertyService*** – Използва се за осъществяване на CRUD операциите за настройка на сензорите предоставени от SensorPropertyController.
5. ***ISensorService – SensorService*** - Използва се за осъществяване на CRUD операциите NA сензорите предоставени от SensorController.
6. ***IUserSensorService – UserSensorService*** -Използва се за осъществяване на CRUD операциите на потребителските сензори предоставени от UserSensorController.

**Common** - .NET Core class library съдърщата логика, която може да бъде използвана от няколко проекта. Съдържа следното разпределение

* + - 1. ***Extensions*** – Сърдържа класове разширяващи функционално други класове.
         1. ***ClaimsPrincipalExtensions*** – Съдържа метод GetId(), който връща Id стойността на автентикирания потребител.
         2. ***IApplicationBuilderExtensions*** – Съдържа UseErrorLogging() и UseErrorHandling() методи, които свързват Middleware-ите към основните настройки на приложението.
         3. ***JsonHelper*** – Съдържа помощни методи, които сериализират/десериализират Stream обекти в JSON и обратно.
      2. ***Middlewares*** – Служат за допълнителна логика преди/след достигане на заявката до контролерите.
         1. ***ErrorHandlingMiddleware*** – Ако възникне изключение пренасочва към ErrorController със съответния Action спрямо статус кода на отговора от заявката за „страницата не е намерена“ или за „грешка в сървъра“.
         2. ***ErrorLoggingMiddleware*** – Служи за логване на възникнали изключения при HTTP заявките.
      3. ***WebClients*** – Съдържа клиент, който служи за извършване на различни видове HTTP заявки.
         1. ***HttpWebClient*** – Клас който съдържа различни методи извършващи HTTP заявки.
      4. ***Constants*** – Клас с константи достъпни за целия Solution.



**Логическа архитектура на мобилното приложението**

Мобилното приложение на системата за управление и преглед на сензори се състои от 4 основни части разпределени в различни проекти – ***Models***, ***Services***, ***Widgets***, ***Main***

**Models** *–* Съдържа модели, които биват използвани за различни операции из приложението.

1. ***Sensor.dart*** – Модел, който дефинира полетата на един SensorDTO обект. Съдържа методи като:
   1. ***fromJson()*** – Връща инстанция на обекта от JSON.
   2. ***toJson()*** – Връща JSON от попълнен обект.
   3. ***All()*** – Създава Resource обект, който ще върне лист от обекти.
   4. ***InitResourceByIdWithoutResponse(id)*** – създава Resource обект, който не очаква обратно инстанция на обекта.
   5. ***initResourceByIdWithResponse(id)*** – Създава Resource обект, който очаква обратно инстанция на обекта.
   6. ***initResourceByIdWithIntResponse(id)*** – Създава Resource обект, който очаква обратно число като резултат.
   7. ***initWithJsonBody(payload)*** – Създава Resource обект, който праща обект в HTTP заявката като Body параметър.
2. ***SensorProperty.dart*** Модел, който дефинира полетата на един SensorPropertyDTO обект. Съдържа методи като:
   1. ***fromJson()*** – Връща инстанция на обекта от JSON.
   2. ***toJson()*** – Връща JSON от попълнен обект.
   3. ***All()*** – Създава Resource обект, който ще върне лист от обекти.
   4. ***InitResourceByIdWithoutResponse(id)*** – създава Resource обект, който не очаква обратно инстанция на обекта.
   5. ***initResourceByIdWithResponse(id)*** – Създава Resource обект, който очаква обратно инстанция на обекта.
   6. ***initWithJsonBody(payload)*** – Създава Resource обект, който праща обект в HTTP заявката като Body параметър.

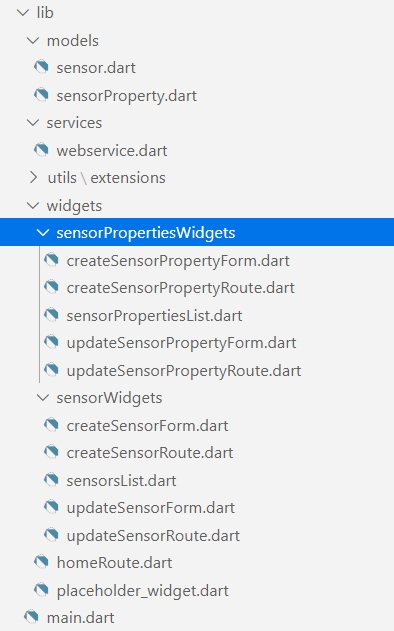
**Services** – Съдържа класове, които служат за извършване на HTTP заявки.

1. ***Resource<T>*** - Generic клас, който съдържа данни като URL, функция за parse-ване на обект/JSON-JSON/обект и body.
2. ***WebService*** – Връща различни видове Future като load, fetch, delete, send, update.

**Widgets** – Уиджети служещи за визуализация и бизнес логика. Разпределение на файловете:

1. ***SensorPropertiesWidgets*** – Служат за логиката свързана с настройките на сензорите
   1. ***CreateSensorPropertyForm*** – Уиджет, който служи за създаване на настройка за сензор и валидиране на формата.
   2. ***CreateSensorPropertyRoute*** – Показва екрана за създаване на настройка за сензор.
   3. ***SensorPropertiesList*** – Показва лист от наличните настройки за сензори като ги взима, чрез API заявка към SensorPropertyController.
   4. ***UpdateSensorPropertyForm*** – Уиджет, който служи за редакция на настройка за сензор и валидиране на формата.
   5. ***UpdateSensorPropertyRoute*** – Показва екрана за редактиране на настройка за сензор.
2. ***SensorWidgets*** – Служат за логиката свързана с сензорите
   1. ***CreateSensorForm*** – Уиджет, който служи за създаване на настройка за сензор и валидиране на формата.
   2. ***CreateSensorRoute*** – Показва екрана за създаване на сензор.
   3. ***SensosList*** – Показва лист от наличните сензори като ги взима, чрез API заявка към SensorController.
   4. ***UpdateSensoryForm*** – Уиджет, който служи за редакция на сензор и валидиране на формата.
   5. ***UpdateSensorRoute*** – Показва екрана за редактиране на сензор.
3. ***HomeRoute*** – Служи за показване на екрана за navigation bar с бутони, които превключват между различните екрани.

**Main** – Стартовата точка на мобилното приложение.



**Логическа архитектура на базата данни на приложението**

Мобилното приложение на системата за управление и преглед на сензори се състои от 4 основни части разпределени в различни проекти – ***Models***, ***Services***, ***Widgets***, ***Main***

**Използвана литература**

ASP.NET Core - <https://bg.wikipedia.org/wiki/ASP.NET_MVC>

Flutter - https://en.wikipedia.org/wiki/Flutter\_(software)