**3. Обзор на използваните технологии**

3.1. nRF5 SDK

nRF SDK представлява набор от софтуерни услуги за разработка , с чиято помощ се изграждат се осъществява програмна реализация на базата на чиповете nRF51 и nRF52. Наборът от софтуерни услуги предоставя на разработчика множество модули и примерни проекти за реализация: Bluetooth low energy профили, Device Firmware Upgrade (DFU), GATT serializer.

3.2. nRF51822

Това е чипът, който бива програмиран с помощта на nRF SDK. Чипът представлява разработка на норвежката компания - Nordic Semiconductors. Компанията се специализира в разработката на безжични SoC (System on chip) системи със свръх-ниска мощност и устройства работещи в диапазона на 2.4 GHz честотна лента. Крайните разработени продукти на компанията,стигащи до потребителя, биват потребителска електроника, безжични мобилни аксесоари, безжични конзоли за игри, мишки, клавиатури и други.

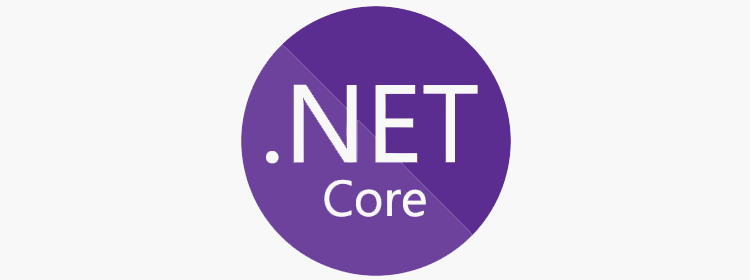
Чипът NRF51822 представлява мултипротоколна система върху чип (SOC), използваща блутут с ниска мощност (BLE - Bluetooth low energy). Основните плюсове на чипа са ниската мощност и високата производителност, както и това че предоставя разделение между програмния кода за разработка на приложението и вградените протоколни стекове. Стекът за Bluetooth low energy представлява прекомпилиран бинарен файл предоставен от Nordic Semiconductor, по такъв начин програмния код се компилира самостоятелно.

Характеристики:

* Използва 32-битов, ARM Cortex MO MCU микроконтролер.
* Налична програмируема flash памет за разработка - 256kB/128 kB.
* Програмната периферна връзка (PPI) представлява 16-канална шина за директна или автономна системна периферна комуникация без интервенция на процесора.
* Налична оперативна (RAM) памет - 32kB/16kB.
* Брой GPIO (входно - изходни пинове) - 32 броя.
* Осцилатори: 16MHz кристален осцилатор, 16MHz RC - осцилатор, 32 KHz кристален осцилатор, 32 KHz RC осцилатор.

3.3. ASP .NET Core

**ASP .NET Core представлява open - source framework, предоставен от Microsoft, за разработка на web-услуги върху платформата .NET, работеща върху платформите: macOS, Linux, Windows. За пръв път е пусната на пазара през 2016-та година. Технологията предоставя като възможности: модулизиран framework, дистрибутиран като NuGet packages, Cloud - оптимизирана среда, лек и модулизиран HTTP - pipeline.**

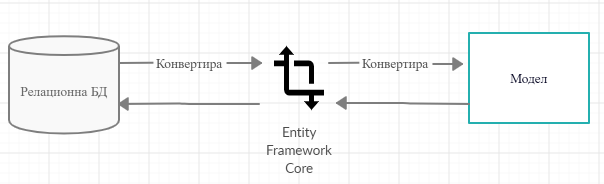
****

**3.3.1 Entity Framework Core (EFC)**

**Enitity framework core представлява крос-платформана технология с отворен код , служеща като object-relational mapper (ORM). С нейна помощ става възможно CRUD операциите към база от данни да стават директно с .NET обекти. Entity Framework Core поддържа следните бази от данни: *SQL Server, SQLite, MySQL, PostgreSQL, Oracle DB и др.* Достъпът до данните при използването на EFC-технологията се осъществява чрез използването на модел (клас, чието преназначение е да оформи, представи и енкапсулира определен вид данни). Под модел също така може да се разбира entity (запис) в базата данни.**

**Друг важен компонент при работа с EFC - технологията е класът *DbContext , част от пакета:*** Microsoft.EntityFrameworkCore. Целта е да се създаде клас***, наследяващ DbContext, който да се използва за репрезантиране на сесия с релационна база от данни.***

**Схема на работа на EFC:**

****

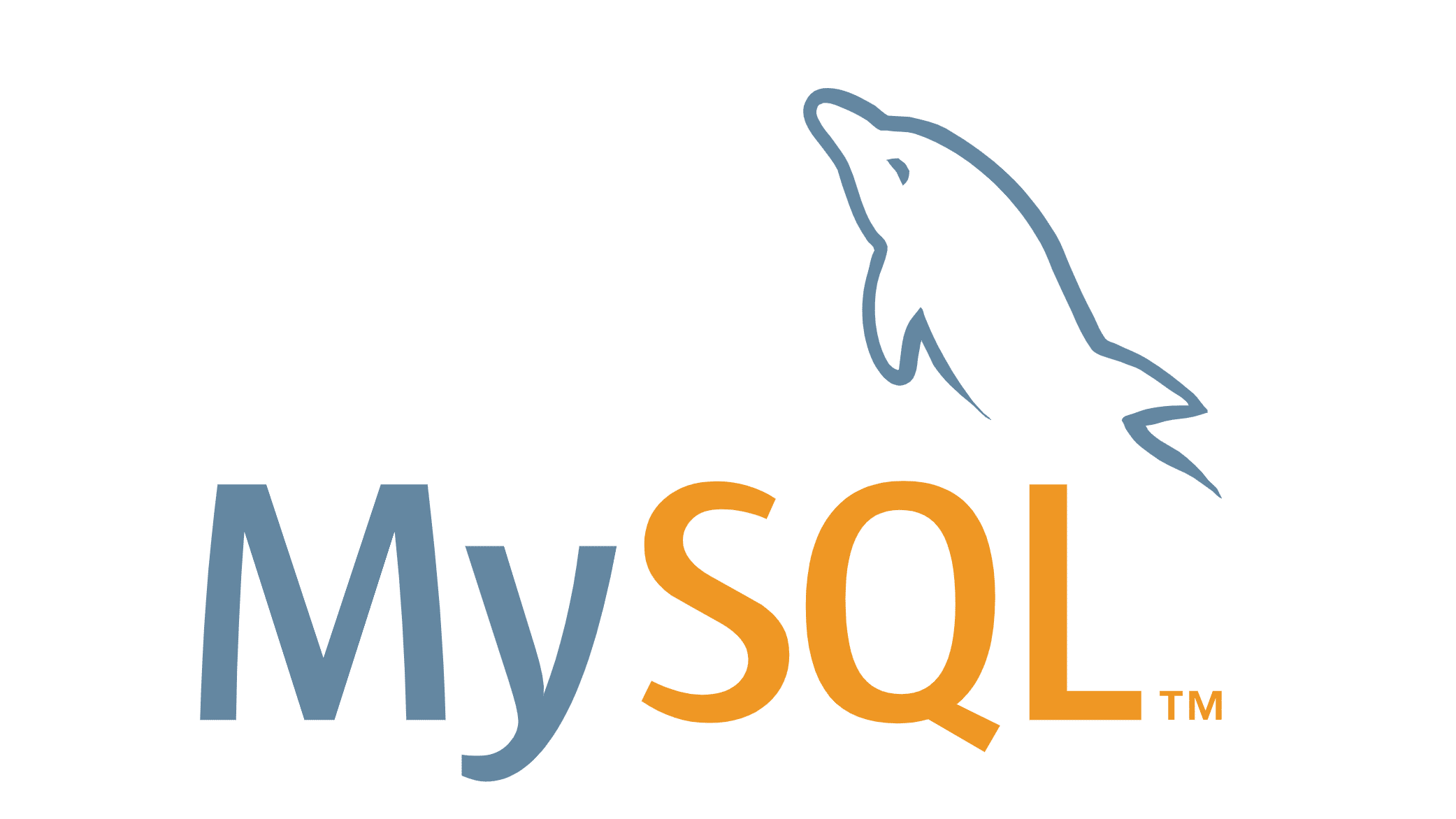
**3.3.2 Dependency injection в ASP .NET Core**

**ASP .NET Core поддържа шаблонът dependency injection, с чиято помощ се постига Inverison of Control, т.е. преодоляват се зависимостите на между класовете и зависимите от тях такива. В зависимост от обхвата на живот на даден обект се дефинират три типа обхват:**

* ***Singleton*: при този тип обхват от даден клас се създава и споделя само една единствена иснтанция по време на целия живот на приложението.**
* ***Transient*: инстанция на даден клас се създава всеки път щом бъде бъде заявена такава.**
* ***Scoped*: инстанция на даден клас се създава веднъж в зависимост от дефинирания обхат на сървиса, грижещ се за създаването на обекта. Например такъв тип обхват може да бъде обхватът при пращане на единична заявка.**

3.4 MySQL

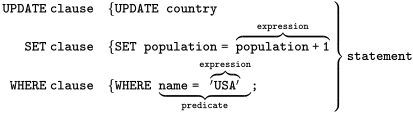
**MySQL представлява система за управление на релационни база данни, базирана на езика SQL (Structured Query Language) разпространява се като свободен софтуер от Шведската компания MySQL AB**

****

**SQL представлява език за създаване, модифициране и добавяне на данни към дадена релационна база данни. Предимствата, които предоставя са че множество записи могат да бъдат достъпени само с една единствена команда, също така се елиминира нуждата да бъде описвано как да се достъпи даден запис, т.е. без или с индекс. Синтаксисът на езика е разделен на няколко елемента включващи:**

* ***Клаузи/Clauses, представляват съставна част от изявленията и заявките.***
* ***Изявления/Statements, генерират скаларни стойности или таблици, състоящи се от колони и редове от данни.***
* ***Предикати/Predicates, специфицират състояния, които биват оценявани от SQL като (true/false/unknown), целта им е да лимитират ефекта на изявлението и заявката или да пренасочат програмния поток.***
* ***Заявки/Queries, целят връщане на данни по дадена критерия.***
* ***Изявления/Statements, чрез тях се контролират транзакции, програмен код, конекции и др.***

**Схема, описваща няколко елемента, съставящи изявление (statement):**

****

**Достъпни са много програмни интерфейси, чрез които се дава възможност програми, написани на различни програмни езици, да имат достъп до MySQL база данни. MySQL е написан на програмните езици C и C++.**

**MySQL работи на много различни платформи: GNU/Linux, Mac OS, Windows, NetBSD, FreeBSD и много други.**

**За администриране може да се използва включено приложение работещо от командна линия, както и GUI - приложение като MySQL Administrator и MySQL Query Browser.**

**3.5 Azure web-services**

**Azure web-services представлява напълно управляема web-хостинг услуга за изграждане на уеб-приложения.**

**3.5.1. Azure Database for MySQL**

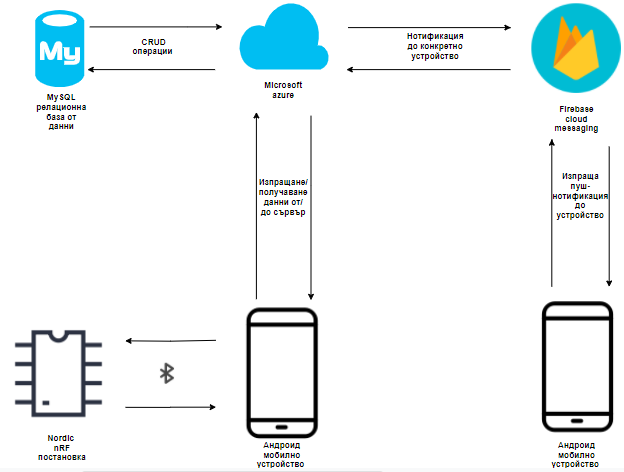
**Представлява услуга за управление на MySQL релационна база от данни. Услугата поддържа защита на данните чрез автоматизиран backup и време за възстановяване от 35 дни, както и автоматизирано управление на операционната система и базата данни. Услугата предоставя цялостно управление на базата данни. Двата режима, в които може да работи услугата са:**

* ***Single serve*r: сървър, с минимални изисквания. Създаден е с цел да може да се поеме по-голямата част от управлението на базата от данни. Поддържа версии на MySQL: 5.6, 5.7, 6.0.**
* ***Flexible server*: сървър, с чиято помощ се осигурява по-стриктен контрол върху управлението на базата от данни и конфигурационните настройки.**

**4. Проектиране на приложението**

4.1. Цялостно проектиране на приложението

Приложението се състои от четири компонента: firmware, мобилно приложение, сървърна част и база от данни. Всички те представляват цялостна система, изграждаща приложение за паркиране.



4.1. Проектиране на firmware, код за управление на установката

Управлението на установката се извършва на базата на подаваните, от клиентското андроид приложение, байтове. По такъв начин се симулира влизане/излизане на паркинг, подава се ниска или висока стойност на конкретно GPIO с цел задвижване моторчето от установката.

4.2. Проектиране на сървърна част

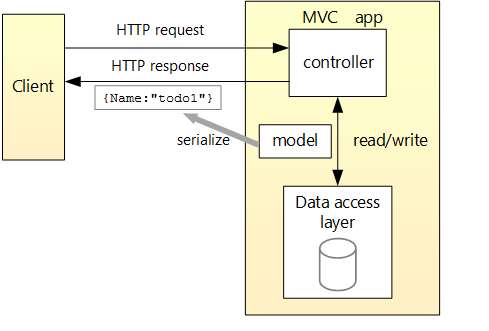
**Това е компонентът, чиято цел е да управлява (съхранява, променя, добавя) данни, свързани с престоя на даден потребител на паркинг, да вика съхранени процедури от външна база данни, както и да изпълни услугата по репортване на неправилно спряло МПС до собственика му.**

**Ключова за изготвянето на сървърната част е технологията: *REST***

**REST или RESTful, в съкращение (REpresentational State Transfer) представлява архитектура, чрез която се изгражда стандарт за достъпване на отдалечени ресурси при връзка сървър - клиент. В комбинация с HTTP - протокола се дефинира RESTful API - технологията. Тя се състои от четири метода за операция с отдалечен ресурс:**

* **GET - доставя специфичен ресурс или колекция от ресурси**
* **POST - създава нов ресурс**
* **PUT - модифицира специфичен ресурс**
* **DELETE - премахва даден ресурс**

**Предаването на данните в REST - системата става чрез JSON или XML протоколите.**

****

***Базова схема на .NET ASP Core приложение, ползващо REST архитектура***

**4.2.1 Добавяне на Model**

**Model е клас, чрез която се репрезентират данните, управлявани от приложението. Може да има бизнес - логика, състояние и данни. Като Model са дефинирани класовете:**

1. **Car: клас моделиращ, данни за МПС на потребител, съдържа пропъртитата: *Id(тип int), Plate(тип string), OnParking(тип bool). Модели от този тип клас се ползват при добавяне на МПС на потребител, както и при заявка за достъпване - всички МПС - та на даден потребител.***
2. **Customer: чрез този клас се описват данни за потребител на системата, съдържа пропъртитата: *Id(тип int), PhoneNo(тип string), съдържащо анотация за регулярен израз с цел валидиране на данните при добавяне и евентуално връщане на грешка, FCMToken(тип string), FirebaseUserId(тип string), ParkingStays (тип List<ParkingStay>). Класът се ползва при регистриране на потребител в системата, както и при разчитане данните на потребителя от JWT (Json web token) с цел аутентикация.***
3. **ParkingStay: описва данни за престоя на дадено МПС на паркинг, представлява запис от базата данни, добавян при всяко влизане на паркинг, и ъпдейтван при всяко излизане. Съдържа следните пропъртита: *Id (тип int), DateTimeEntered (тип Datetime), DateTimeExited (тип Datetime), PlateNo (тип string), Customer (тип Customer).***

**Програмен код на гореописаните класове:**

public class Car

{

public int Id { get; set; }

[Required]

[Column("plate")]

public string Plate { get; set; }

[Required]

[Column("onParking")]

public bool OnParking { get; set; }

}

public class Customer

{

public int Id { get; set; }

[RegularExpression(@"\+[3]{1}[5]{1}[9]{1}[0-9]{9}")]

[Required]

[Column("phone\_no")]

public string PhoneNo { get; set; }

[Required]

[Column("token")]

public string FCMToken { get; set; }

[Required(ErrorMessage = "Field:firebase\_user\_id must not be null!")]

[Column("firebase\_user\_id")]

public string FirebaseUserId { get; set; }

public List<ParkingStay> ParkingStays { get; set; }

}

public class ParkingStay

{

public int Id { get; set; }

[Required]

[Column("timeEnter")

public DateTime DateTimeEntered { get; set; }

[Column("timeExit")]

public DateTime DateTimeExited { get; set; }

[ForeignKey("customerId")]

public int CustomerId { get; set; }

[Required]

[Column("car\_plate")]

public string PlateNo { get; set; }

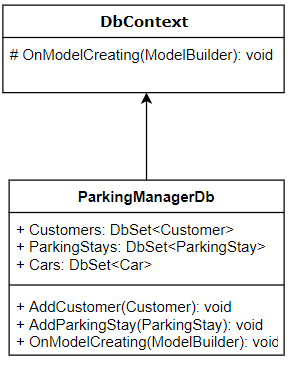
[JsonIgnore]

public Customer Customer { get; set; }

}

**4.2.2 Добавяне на database context**

**Database context представлява основен клас, чрез който става координирането на Entity Framework Core - функционалността с Model - класовете. Такъв клас се добавя като се прави наследник на класа Microsoft.EntityFrameworkCore.DbContext. В приложението това представлява класът: ParkingManagerDb:**

******

***UML - диаграма на класа ParkingManagerDb***

**Ключов за класа ParkingManagerDb е методът OnModelCreating(ModelBuilder), в него става свързването на Model -класовете с имената на таблиците от базата данни.**

**Инстанцирането на класа става чрез регистрирането на същия като service, посредством dependecy injecion (DI) - контейнер, в класа Startup.cs. По такъв начин инстанция на ParkingManagerDb се предоставя на Controller - класовете.**

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContextPool<ParkingManagerDb>(options =>

{

options.UseMySql(Configuration.GetConnectionString("DefaultConnectionString"), mySqlOptions =>

{

mySqlOptions.EnableRetryOnFailure();

});

});

}

**4.2.3 Добавяне на Controller**

**Controller е класът, отговорен за приемането на заявките идващи от клиента, както и за тяхната обработка и връщане на данни или съответно изпълняване на друго действие на сървъра.**

**За всеки клас наследяващ Controller се дефинира route, или пътят, чрез който ще бъдат достъпни методите на дадения контролер. Това става чрез използване на атрибут: [Route(@contrllerPath)] слаган при дефинирането на класа наследяващ Controller.**

1. **Път: “api/v1” достъпва ParkingStayController - класа.**
2. **Път: “api/v2” достъпва CustomerController - класа.**
3. **Път: “api/v3” достъпва PlateReportController - класа.**
4. **Път: “api/v4” достъпва CarController - класа.**

**5. Програмна реализация**

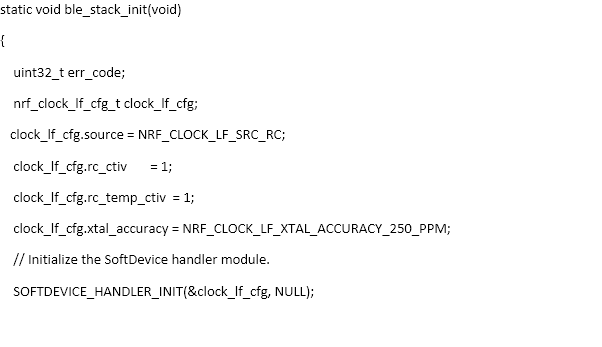
5.1. Проектиране на firmware

За проектирането на firmware е използвана интегрираната среда за разработка UVision Keil върху наборът от софтуерни услуги за разработка - nRF SDK. Проектирането минава през няколко фази, първата от тях е инициализация на основни компоненти при стартиране на приложението, последващи фази са свързани със обработка на получени през блутут байтове и управление на хардуера спрямо тях.

5.1.1. Инициализация на bluetooth low energy stack и SoftDevice

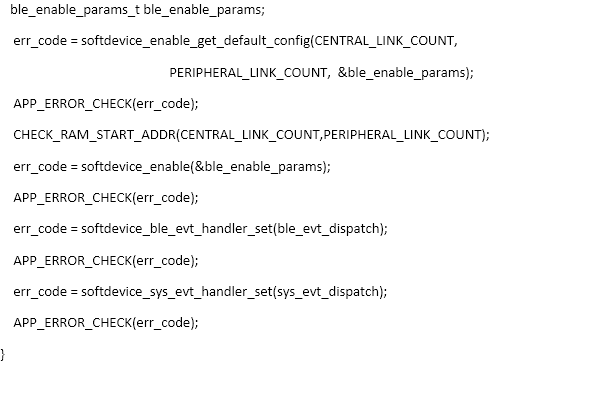
SoftDevice е безжичен протоколен стек, който е флашнат на специално място на микроконтролера. Представлява прекомпилиран бинарен файл, въпреки че е възможна евентуалната разработка на приложение без SoftDevice, при създаване на Bluetooth low energy - използващо приложение употребата на SoftDevice става задължителна.

Инициализация на SoftDevice, фрагмент от функцията **ble\_stack\_init()**:

**nrf\_clock\_lf\_cfg\_t представлява структура, на която биват инициализирани пропъртитата:**

* **source: пропърти тип unit\_8, чрез него се задава стойност източник на осцилаторен часовник.**
* **rc\_ctiv: пропърти тип uint\_8, чрез него се задава стойност на калибриращия интервал.**
* **xtal\_accuracy: пропърти тип uint\_8, чрез него се задава точността при изчисление на timing-прозорците.**

**Инициализация на BLE stack, останала част от функцията ble\_stack\_init() :**

**Функцията**: **softdevice\_enable() се грижи за инициализацията на BLE - стека, след нейното изпълнение трябва да проверим върнатата от нея стойност - uint\_8 за грешка.**

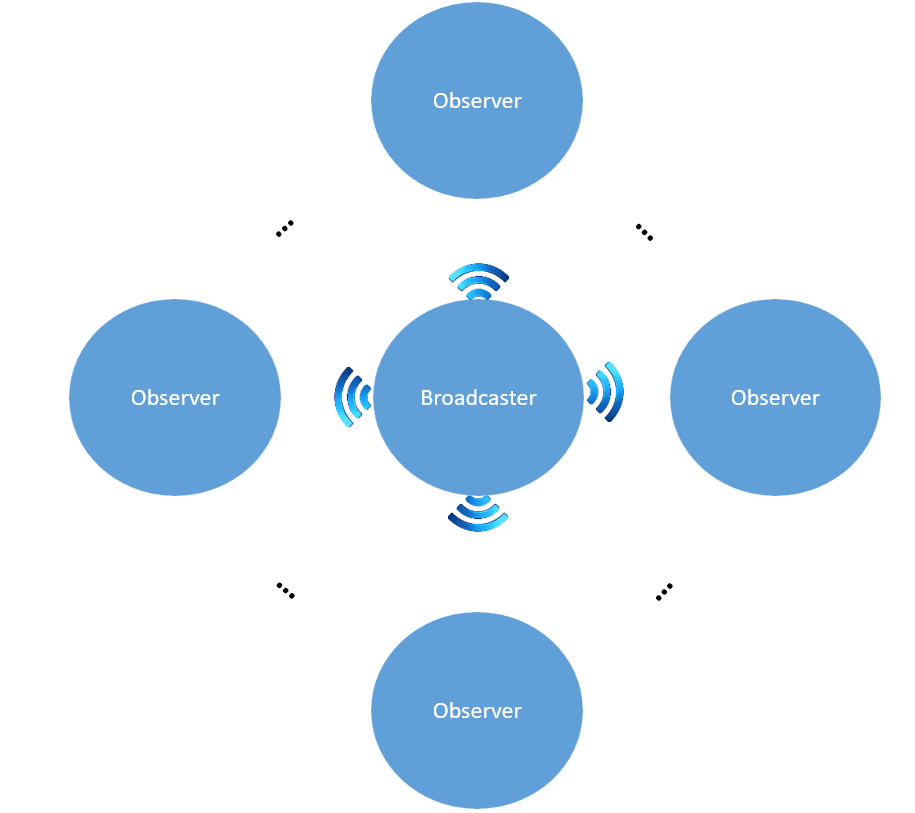
Функцията: **softdevice\_ble\_evt\_handler\_set() се грижи за регистриране на модул за обработка на BLE - събития.**

**Функцията: softdevice\_sys\_evt\_handler\_set() се грижи за регистриране на System on chip (SoC) - събития, получавани от вече инициализирания SoftDevice.**

5.1.2. Инициализация на GAP (Generic access profile):

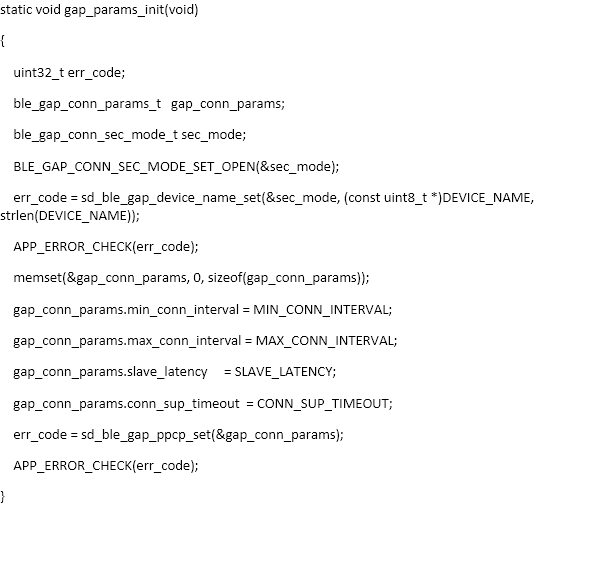
GAP е част от BLE - stack протоколът, отговорна за свързващата функционалност. Чрез него се описва как bluetooth low energy - устройствата изпълняват тези действия:

* Излъчване на данни: осъществява се, когато имаме устройства свързани в режим - Излъчвател/Наблюдател (Broadcaster/Observer). Излъчвателят изпраща advertising - пакети съдържащи данни, наблюдателят сканира за тези advertising пакети.



* Свързване: за да може да бъде осъществено BLE - свързване трябва да имаме периферно - централно двойка устройства. Централно устройство, имащо добри характеристики за памет, мощност и други. Такъв тип устройство са таблети, смартфони и др. Периферно е такъв тип устройство, имащо слаби показатели за памет, мощност и др. Такъв тип устройства са устройства за мерене - сърдечна дейност, блутут мишки и др.

Нужните за инициализация параметри на NRF51822 - устройството са: минимален/максимален интервал при свързване, таймаут на изтичане на конекцията, както и slave - латентност. За изпълнението на тази задача се грижи функцията **gap\_params\_init().**

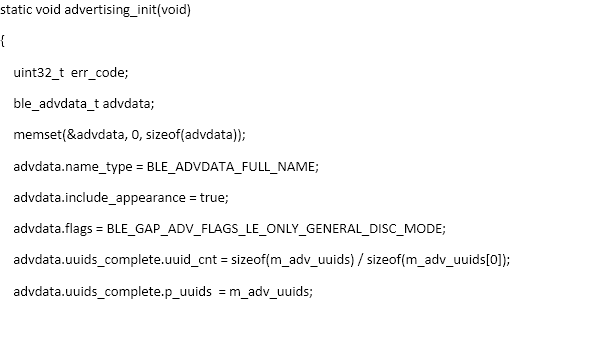


5.1.3. Инициализация на Advertising:

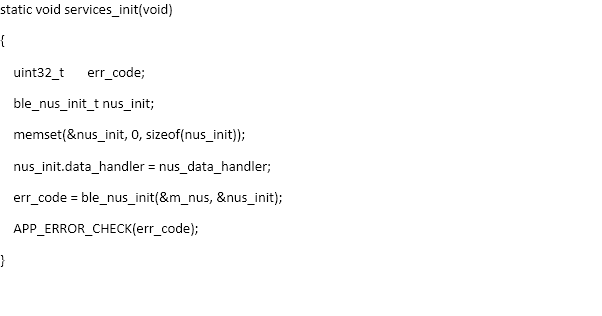
Advertising представлява процесът на излъчване на данни съдържащи основна информация не по-голяма от 31 байта. Данните от пакетът - advertising могат да бъдат прочетени без устройствата да бъдат свързани едно с друго.

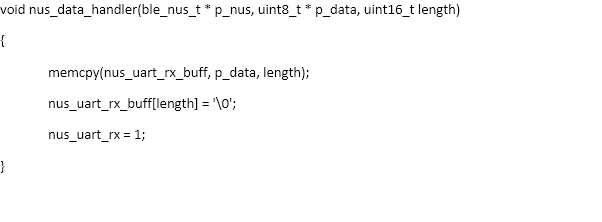
Инициализацията на advertising се състои в инициализация на следните параметри: тип име на устройството, дефинира се от изброимия тип: **ble\_advdata\_name\_type\_t ( BLE\_ADVDATA\_NO\_NAME, BLE\_ADVDATA\_SHORT\_NAME, BLE\_ADVDATA\_FULL\_NAME), брой UUID-та и други. За инициализирането на advertising се грижи функцията: advertising\_init(void).**

**Фрагмент от функцията : advertising\_init(void):**

****

5.1.4. Инициализиране на services:

**Инициализиране на услуги(services), използвани от приложението:**

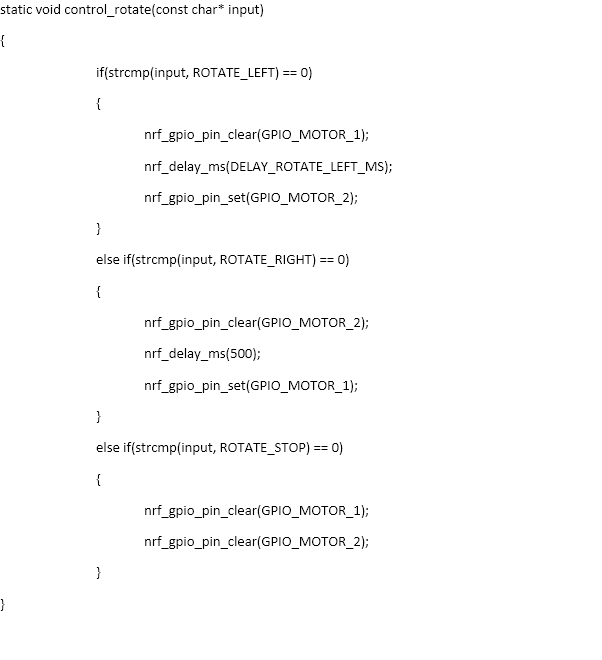
**Функцията използва структурата ble\_nus\_init\_t, с чиято помощ се създава прихващач (handler) на събитие, за което приложението трябва да бъде уведомено, в конкретния случай това е събитие, свързано с четене на данни, получени от главното устройството и записването им в nus\_uart\_rx\_buff[] - буфер: **

5.1.5. Управление на GPIO, обработка на хардуерни прекъсвания:

**Чрез управлението на пиновете p0.00 и p0.02 от чипа - NR5822 се постига завъртане в двете посоки на моторчето от макетната установка, свързано в H-мост. И двата пина са конфигурирани като изходни(output), тоест задачата им е да задават HIGH и LOW - стойности на електрическия сигнал.**

* **Команда "rotr": подава напрежение на GPIO - p0.02.**
* **Команда "rotl": подава нарежение на GPIO - p0.00.**
* **Команда "stop": зачиства двата пина.**

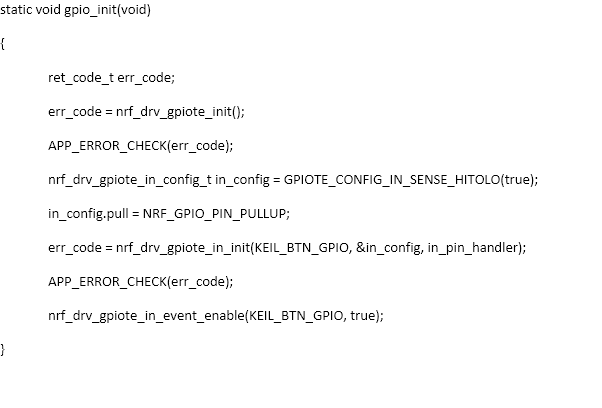
**За това се грижат функциите: void nrf\_gpio\_pin\_set(uint32\_t pin\_number), nrf\_gpio\_pin\_clear(uint32\_t pin\_number), извиквани от функцията control\_rotate(const char\* input), нейното предназначение е на база на подадения вход да завърти моторчето от макетната установка съответно на ляво или дясно, или да го спре напълно, тоест да зачисти стойностите на двата пина. Завъртането на ляво се смята за спускане на бариерата от макетната установка, затова преди подаване на напрежение към съответния пин има "приспиване" на програмата за 10 секунди:**

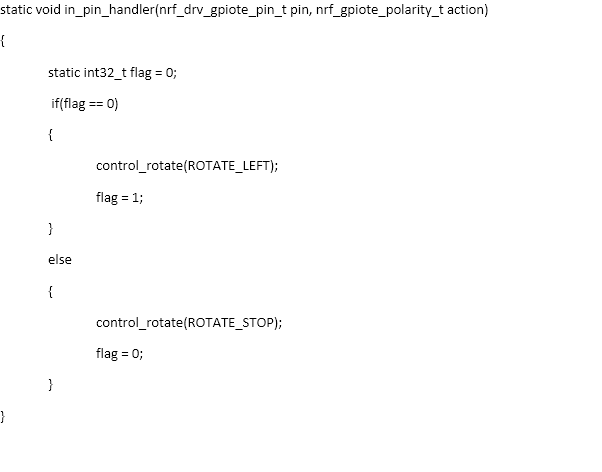
****

**Обработка на хардуерни прекъсвания:**

**Хардуерно прекъсване представлява генериране на асинхронен (неопределен във времето) сигнал към процесора, генериран от външно устройство.**

**Генерирането на такъв сигнал става при натискане на ключ: kw4-3z-3 от макетната установка. Съответно трябва да генерираме обработчик на хардуерното прекъсване, породено от него. Стъпките са следните:**

* **Конфигурира се GPIO, където се намира ключа - p0.03, така че да може да разпознава преминаване на електрическия сигнал от HIGH от LOW. Функцията gpio\_init(): **
* **Дефиниране на функция, служеща като прихващач (handler) при промяна на състоянието на сигнала от HIGH в LOW. Трябва да стане следното: ако флагът, менящ стойността си при всяко натискане на ключа, е със стойност 0, то тогава предстои спиране движението на бариерата, прекъсване работата на програмата за 10 секунди и спускането и надолу (подава се напрежение на GPIO p0.00). При повторно натискане на ключа стойността на флага вече е вдигната в 1, следователно предстои окончателно спиране на бариерата и зачистване на двата пина, управляващи мотора:**

****

**6. Ръководство на потребителя**

**7.Експериментални резултати**