

**Курсова работа**

**по**

**Програмни Среди**

**Изпълнена от:**

**Студент:**

**Валентин Николаев Велков**

**Факултет:**

**ФКСТ**

**Специалност**:

**ИТИ**

**Група:**

**53**

**Факултетен** **№:**

**501216029**

**Тема на Проекта**

Личен медицински картон. Да се проектира и реализира програма, която да следи здравното състояние на потребителя. Да се предостави възможност за отбелязване на историята на посещенията при лекари и специалисти. Проверка за необходимост и напомняне за профилактични прегледи. Справки за честотата на изкараните заболявания и предстоящи прегледи.

**Детайлно задание**

За реализирането на програма, която да следи здравното състояние на

пациенти ще се реализира както Server, така и Client, постигайки Client–server model. Това осугирява сигурност / интегритет на данните съхранявани в

централен сървър. Client ще комуникира със Server чрез REST API като идентифицирането ще стане посредством authentication имплементация (Basic / OAuth). Данните ще се съхраняват на сървъра / backend в база от данни (MySQL database).

Разширявайки заданието на курсовия проект - приложението ще представлява

регистър за множество от болници, тоест за един пациент ще се показва

историята от посещение във всички болници, а не от една.

Client ще представлява десктоп приложение, чрез което потребителя ще може да се впише в системата - доктор или пациент. Като в зависимост от вписаната

роля, потребителя ще има различни възможности за работа с програмата.

Докторът ще може да вписва посещения/диагнози **към определена болница за определен пациент**. Той има цялостен контрол върху данните свързани с посещения при   
него, но не и контрол върху данни за други доктори.

Пациентът ще може единствено да преглежда своите данни - посещения /

диагнози / бъдещи задължителни посещения / резервации. Той ще вижда

данните структурирано - за всяка болница, доктор към болницата и данни за

посещението.

Администраторът ще може да създава болници към приложението, а болницата ще може да вписва доктори към нея. По този начин е възможно

цялостния контрол да става през Client приложението. Възможно е и един доктор да е вписан към няколко болници.

Backend / Server ще представлява web application, който ще изложи REST API за използване с цел реализиране на Client описан по-горе както и база от данни за съхраняване на данните.

Чрез REST API ще може да се:

1. Установява автентичност (login)
2. Създаде нов потребител от доктор
3. Регистриране на нова болница
4. Вписване на нов доктор към болница
5. Извличане на конкретна информация за пациента - болести / диагнози / посещения и т.н.
6. Извличане на конкретна информация принадлежаща на доктора - неговите пациенти спрямо всяка болница, в която е вписан
7. Извличане на конкретна информация за болницата - доктори / пациенти
8. Вписване на информация от доктора за пациента:
   * посещение

* диагноза
  + резервация

Всяка от гореизброените възможности ще бъде достъпна само в зависимост от

предоставените credentials - тоест докторът ще може да прави промени свързани с пациентите –

резервации, посещения, но няма да може да вижда информацията на пациента

към други доктори или болници,а пациента няма да може да прави подобни

промени или дори да вижда подробна информация за доктора. Тоест

scope-a на REST API възможностите е в зависимост от логнатия потребител.

1. **Функционалности**

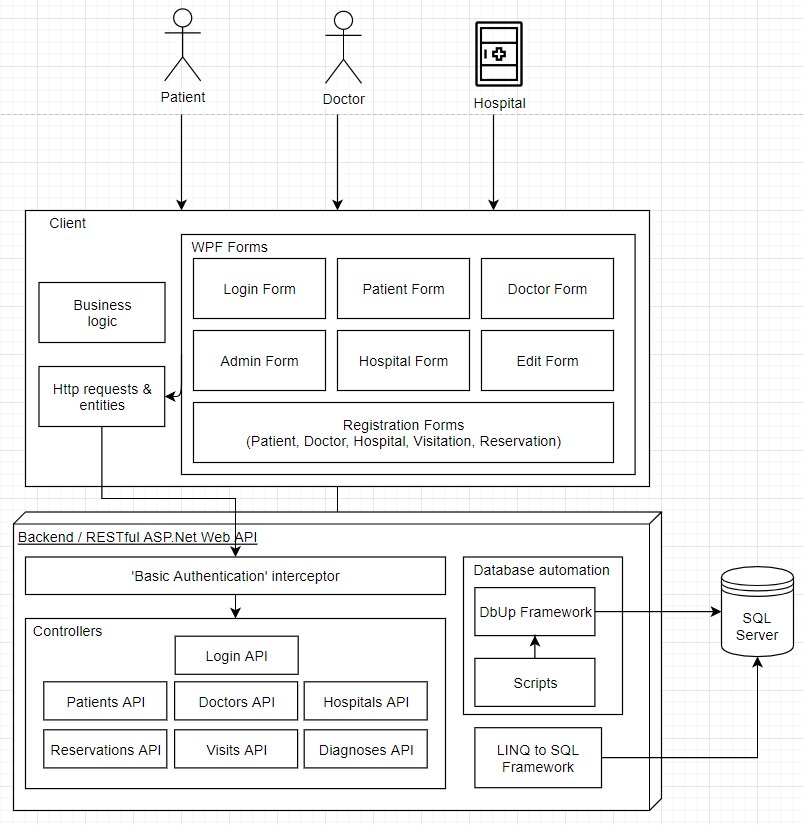
Остановените функционалности по време на разработването на приложението са следните:

* Сигурност (Authentication) при ‘login’ на всеки един от потребителите независимо от тяхната роля
  1. Пациент
* Възможност за извличането на собствена лична информация (имена, егн, адрес)
* Способност за обновяване/промяна на личните данни
* Извличане на информация за собствени резервации
* Извличане на информация за собствени диагнози
* Извличане на информация за предишни посещения
* Преглед на докторите в системата и техните специализации с цел посещение  
  1. Доктор
* Възможност за извличане на собствена лична информация (имена, егн, специализация и т.н.)
* Възможност за обновяване/промяна на личните данните
* Разглеждане на история на посещения от пациенти при него
* Разглеждане и създаване на резервация за конкретен пациент за конкретна дата и час
* Разглеждане и създаване на диагноза за конкретен пациент
* Възможност за обновяване/промяна на данните към посещения/резервации/диагнози
* Регистриране на нов пациент в системата, в случай на посещение от нерегистриран човек  
  1. Болница
* Преглед на лична информация на болницата (име, локация и т.н.)
* Възможност за обновяване/промяна на данните към болницата
* Възможност за преглед на всички доктори работещи в болницата
* Възможност за преглед на всички посещения на доктори работещи в болницата
* Възможност за преглед на всички резервации към доктори работещи в болницата
* Възможност за регистрация на нов доктор като работещ в болницата  
  1. Администратор
* Ексклузивни права за преглед, промяна на пълната информация в системата, включително и регистриране на нова болница, доктор, пациент. Има достъп до всяка от функционалностите описани за другите потребители през REST API.

1. **Архитектура**

Гореизброените функционалности са постигнати чрез Client – Server архитектура.

Клиента (UI) представлява WPF application, който си комуникира със сървъра – backend RESTful API.



1. **Използвани технологии**

* ASP.NET WEB API 2 Controller

С цел лесна имплементация на RESTful услугата чрез имплементация на Controller и множество Models (DTO), както и генериране на API документация.

* SQL Server

Съхраняване на бизнес информацията. Лесна интеграция с ‘LINQ to SQL’ за удобство при работа с базата данни както и DbUp за автоматизирано създаване.

* LINQ to SQL

Позволява програматичен достъп до информацията от базата както и добавяне/модифициране на нова такава. Не е нужно да се грижим за транзакционност и грешки при комуникацията с базата.

* DbUp

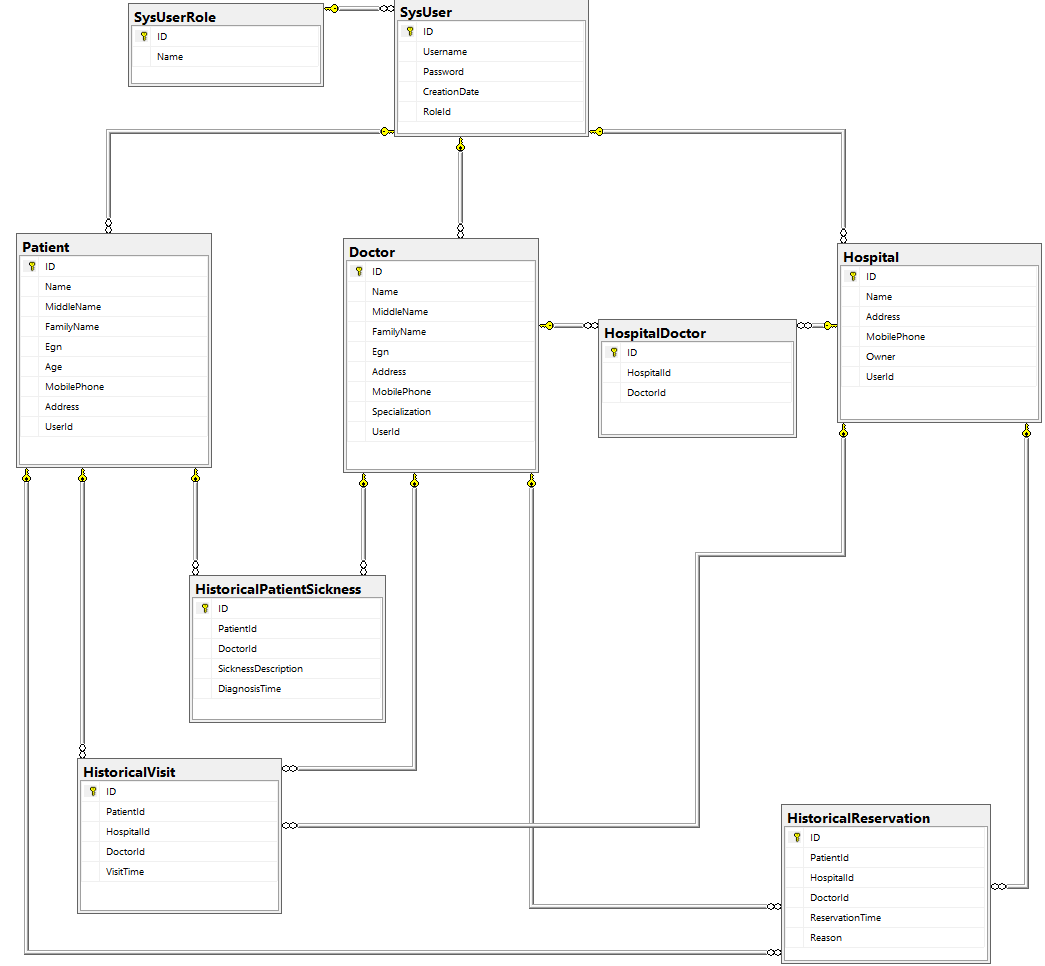
Автоматизация на създаването на база от данни, полезен при миграция. Дефинираме SQL скриптове, които се изпълняват при стартиране на приложението. След изпълнението на даден файл, той се отбелязва за изпълнен в базата (библиотеката си менажира своя таблица), което гарантира че няма да се изпълни отново. При промени в базата е лесно дефинирането на нови файлове и изпълнението им при стартиране. Спестява ‘ръчна’ (manual work) работа.

* AutoMapper

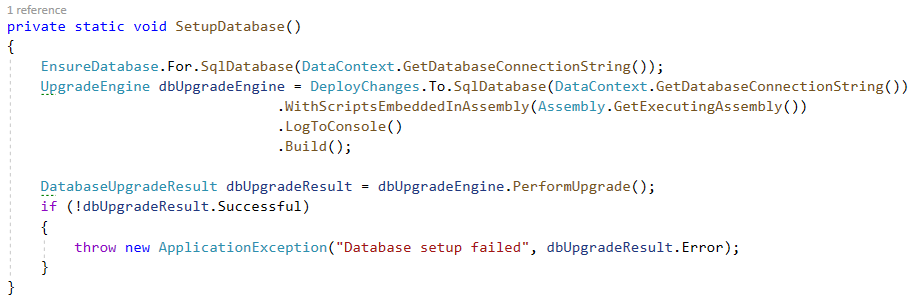
Библиотека, която спомага при mapping на сложните обекти върнати от ‘LINQ to SQL’ при комуникация с базата към прости DTO обекти, които съхраняват само информацията, която е необходимо да се предостави на потребителя при REST отговора. Спомага и за автоматичен mapping на получените DTO обекти, които се получават чрез parse на body от POST/PUT заявки към ‘LINQ to SQL‘ заявки, за бързото добавяне/променяне в базата данни.

1. **База данни**

Създадената база данни чрез дефинираните скриптове има следната ER схема:



**Генерирането на по-горе показаната схема става посредством DbUp Framework:**

****

**Скриптове, част от проекта (Embedded resource), който DbUp използва:**

1. User management

CREATE TABLE SysUserRole(

ID INT IDENTITY(1, 1) PRIMARY KEY,

Name VARCHAR(50) NOT NULL

);

INSERT INTO SysUserRole(Name) VALUES ('Admin')

INSERT INTO SysUserRole(Name) VALUES ('Patient')

INSERT INTO SysUserRole(Name) VALUES ('Doctor')

INSERT INTO SysUserRole(Name) VALUES ('Hospital')

CREATE TABLE SysUser(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

Username VARCHAR(255) NOT NULL,

Password VARCHAR(255) NOT NULL,

CreationDate DATE NOT NULL,

RoleId INT NOT NULL,

FOREIGN KEY (RoleId) REFERENCES SysUserRole(ID)

);

1. Doctor, Hospital, Patient

CREATE TABLE Doctor(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

Name VARCHAR(255) NOT NULL,

MiddleName VARCHAR(255) NOT NULL,

FamilyName VARCHAR(255) NOT NULL,

Egn CHAR(10) NOT NULL,

Address VARCHAR(255) NOT NULL,

MobilePhone VARCHAR(100) NOT NULL,

Specialization VARCHAR(255) NOT NULL,

UserId UNIQUEIDENTIFIER NOT NULL,

FOREIGN KEY (UserId) REFERENCES SysUser(ID)

);

CREATE TABLE Hospital(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

Name VARCHAR(255) NOT NULL,

Address VARCHAR(255) NOT NULL,

MobilePhone VARCHAR(100) NOT NULL,

Owner VARCHAR(255) NOT NULL,

UserId UNIQUEIDENTIFIER NOT NULL,

FOREIGN KEY (UserId) REFERENCES SysUser(ID)

);

CREATE TABLE Patient(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

Name VARCHAR(255) NOT NULL,

MiddleName VARCHAR(255) NOT NULL,

FamilyName VARCHAR(255) NOT NULL,

Egn CHAR(10) NOT NULL,

Age INT NOT NULL,

MobilePhone VARCHAR(100) NOT NULL,

Address VARCHAR(255) NOT NULL,

UserId UNIQUEIDENTIFIER NOT NULL,

FOREIGN KEY (UserId) REFERENCES SysUser(ID)

);

1. Link between hospital and doctor

CREATE TABLE HospitalDoctor(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

HospitalId UNIQUEIDENTIFIER NOT NULL,

DoctorId UNIQUEIDENTIFIER NOT NULL,

FOREIGN KEY (HospitalId) REFERENCES Hospital(ID) ON DELETE CASCADE,

FOREIGN KEY (DoctorId) REFERENCES Doctor(ID) ON DELETE CASCADE

);

1. Historical data (diagnoses, reservations, visits)

CREATE TABLE HistoricalPatientSickness(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

PatientId UNIQUEIDENTIFIER NOT NULL,

DoctorId UNIQUEIDENTIFIER NOT NULL,

SicknessDescription TEXT NOT NULL,

DiagnosisTime DATETIME NOT NULL,

FOREIGN KEY (PatientId) REFERENCES Patient(ID) ON DELETE CASCADE,

FOREIGN KEY (DoctorId) REFERENCES Doctor(ID) ON DELETE CASCADE

);

CREATE TABLE HistoricalReservation(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

PatientId UNIQUEIDENTIFIER NOT NULL,

HospitalId UNIQUEIDENTIFIER NOT NULL,

DoctorId UNIQUEIDENTIFIER NOT NULL,

ReservationTime DATETIME NOT NULL,

Reason TEXT NOT NULL,

FOREIGN KEY (PatientId) REFERENCES Patient(ID) ON DELETE CASCADE,

FOREIGN KEY (HospitalId) REFERENCES Hospital(ID) ON DELETE CASCADE,

FOREIGN KEY (DoctorId) REFERENCES Doctor(ID) ON DELETE CASCADE

);

CREATE TABLE HistoricalVisit(

ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),

PatientId UNIQUEIDENTIFIER NOT NULL,

HospitalId UNIQUEIDENTIFIER NOT NULL,

DoctorId UNIQUEIDENTIFIER NOT NULL,

VisitTime DATETIME NOT NULL,

FOREIGN KEY (PatientId) REFERENCES Patient(ID) ON DELETE CASCADE,

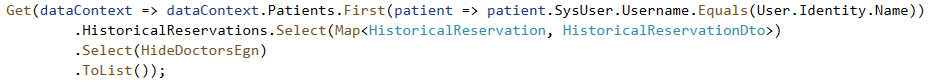
FOREIGN KEY (HospitalId) REFERENCES Hospital(ID) ON DELETE CASCADE,

FOREIGN KEY (DoctorId) REFERENCES Doctor(ID) ON DELETE CASCADE

);

Имайки по-горе създадената база данни, комуникацията с нея с цел read/write става лесно посредством LINQ to SQL.

Пример:

Намирането на всички резервации за конкретен пациент сравнявайки Username стойността от базата с User.Identity.Name, и извличането на съответните резервации за този пациент:

(Конкретния пример е взет от заявка, която предоставя всички резервации на пациента, за да може той да ги следи, затова има наличието на HideDoctorsEgn метод в Select, за да скрием „чувствителна“ информация от пациета – него не го интересува егн на доктора. По този начин REST API-то няма да го върне)

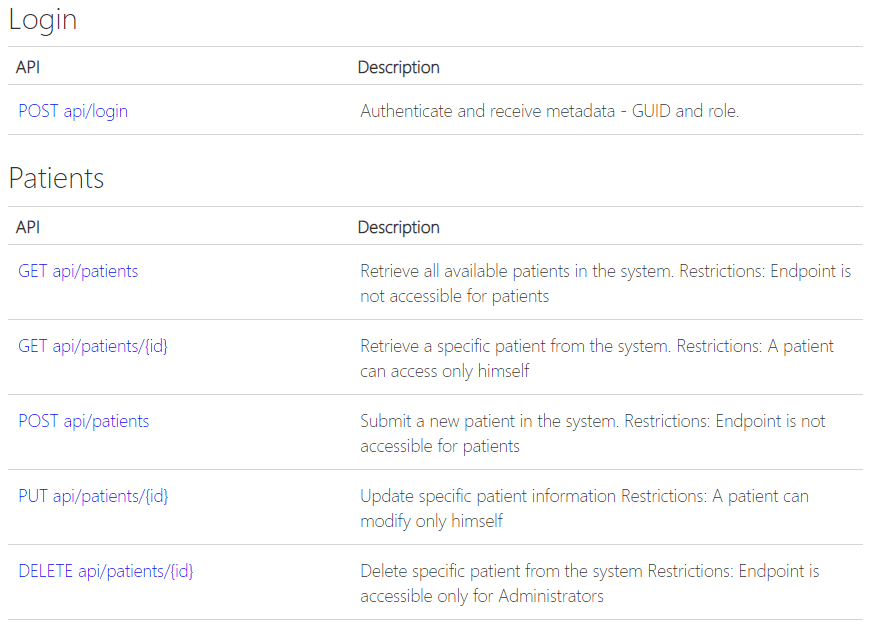
1. **Имплементация**

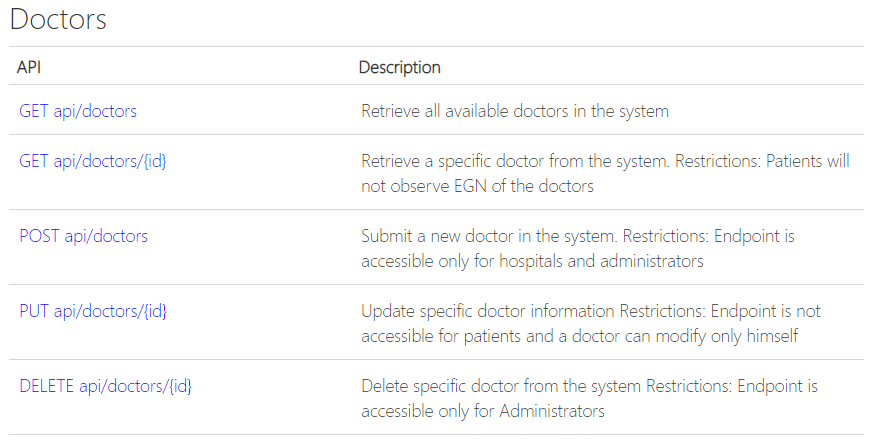
REST API-то има 6 основни ресурса: Patients, Hospitals, Doctors, Reservations, Diagnosis, Visits.

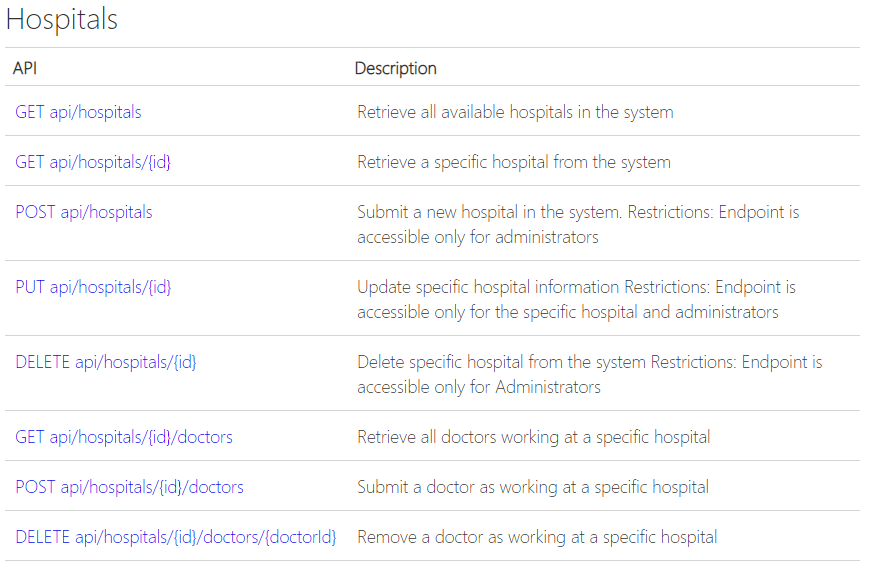
Важното и общото за тях е че всички работят с Authorization header, чрез който се установява самоличността и достъпа на потребителя до конкретен Endpoint.  
  
В случай успешното изпълнение на заявка се връща статус код за успех (200/202/203 и т.н.) и body с информацията (ако е GET request).

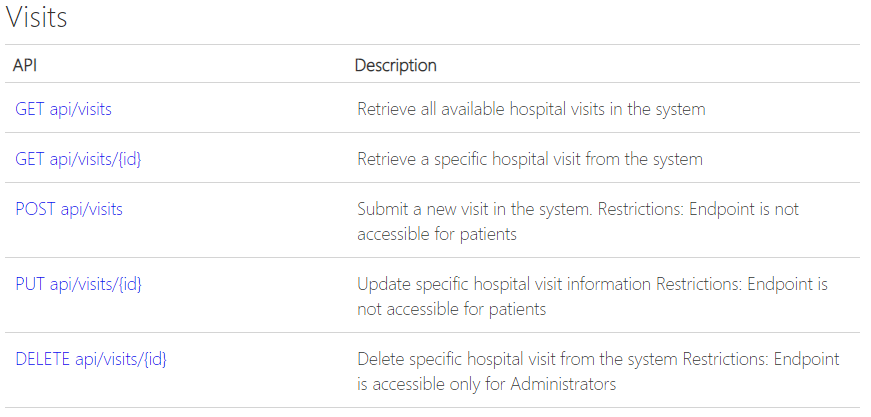
В случай на неуспешното изпълнение на заявка се връща статус код за неуспех и body с информация за грешката.

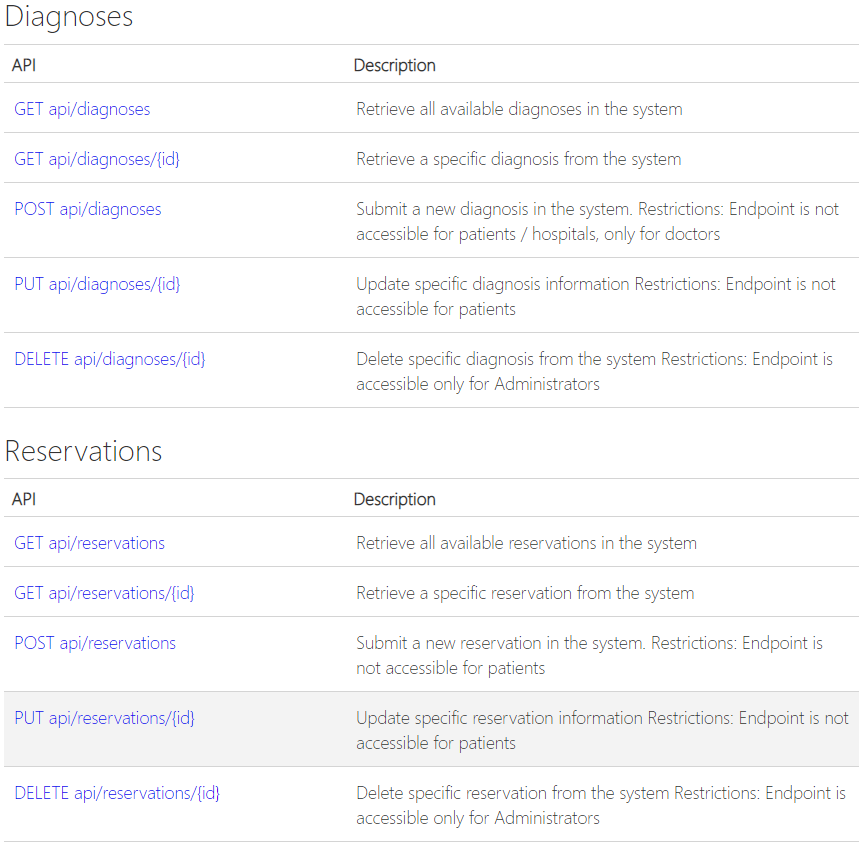
По-долу са описани конкретните API endpoints, а също така и детайлно описание на всеки endpoint какво приема/връща.



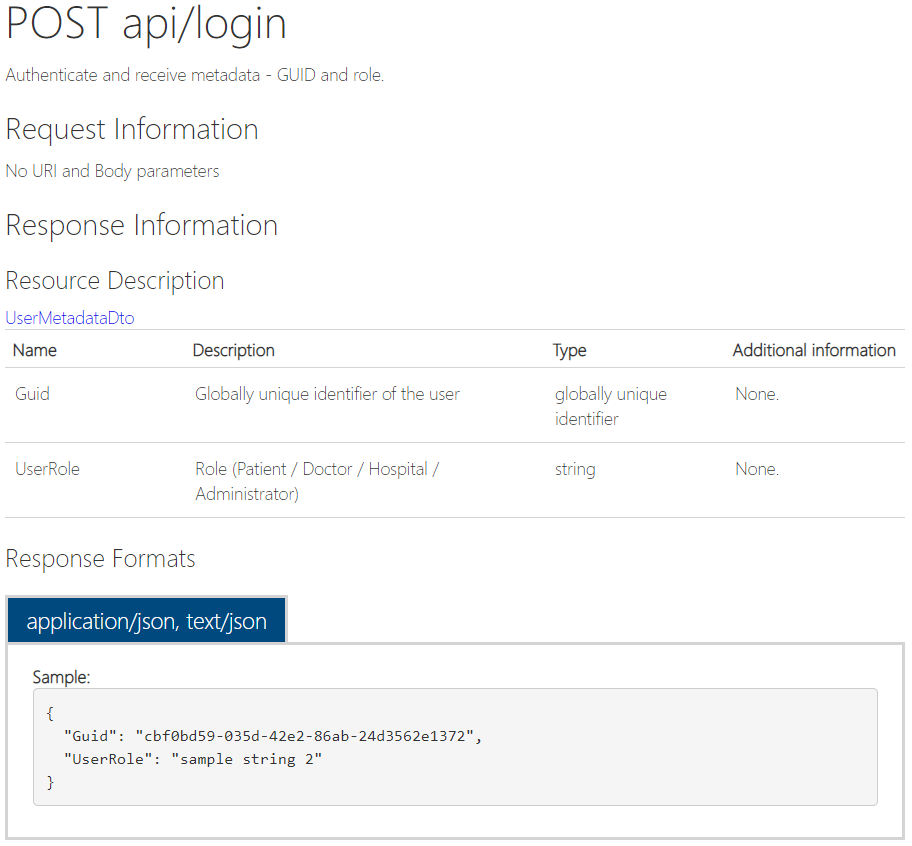






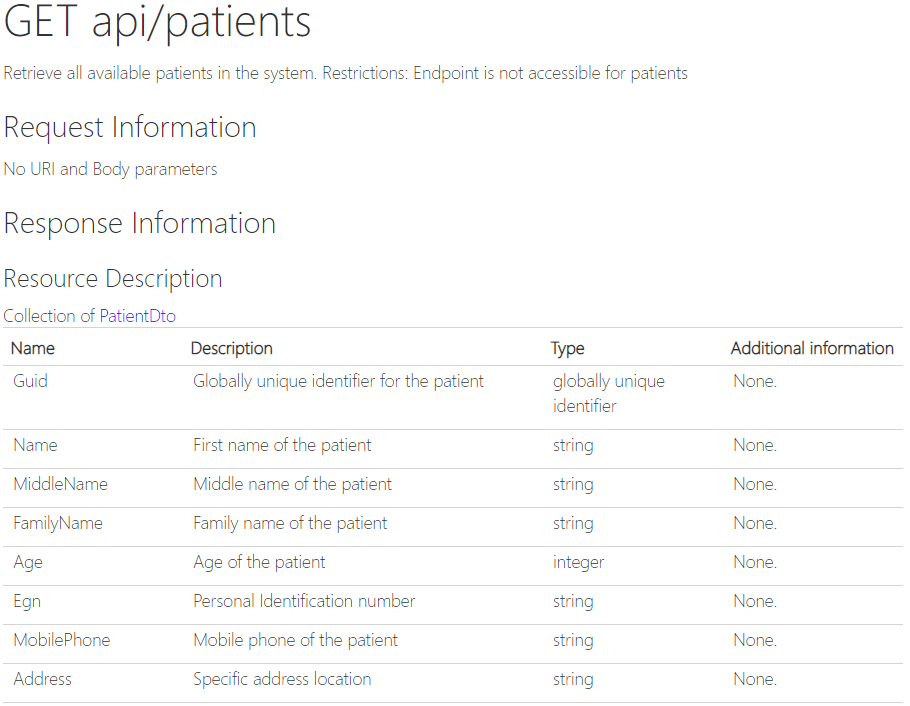


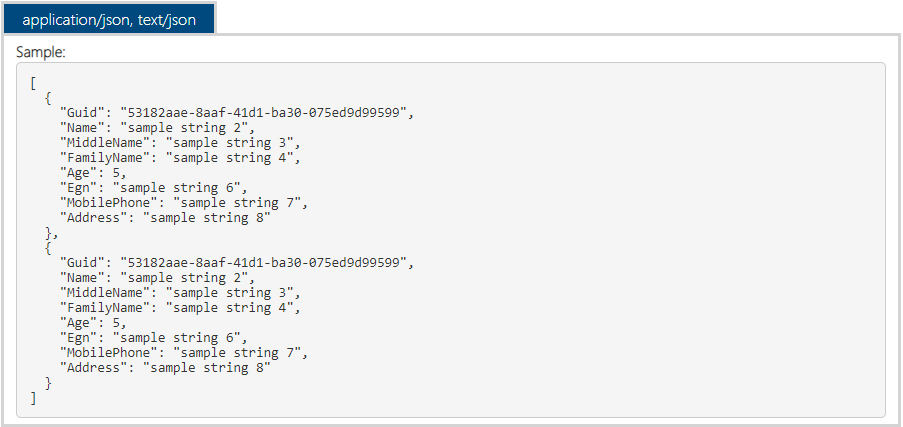
* 1. **Login API**

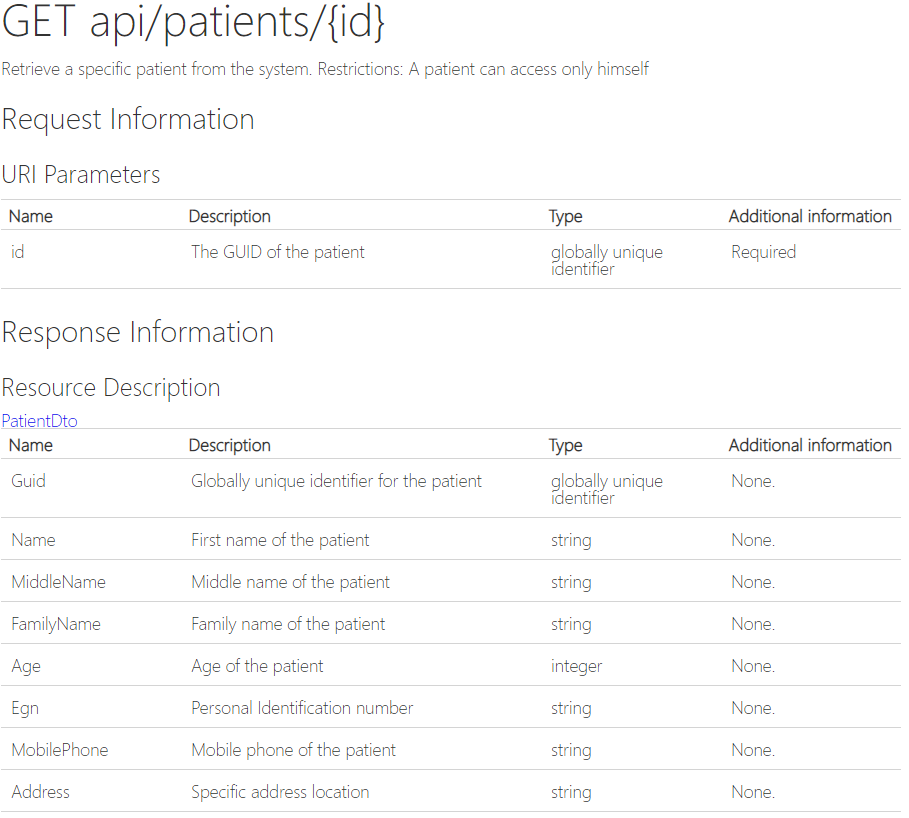


Authorization се случва посредством Header с Basic authentication credentials – а именно base64 encoded username:password. Използвайки HTTPS осигуряваме encrypted headers и сигурност.

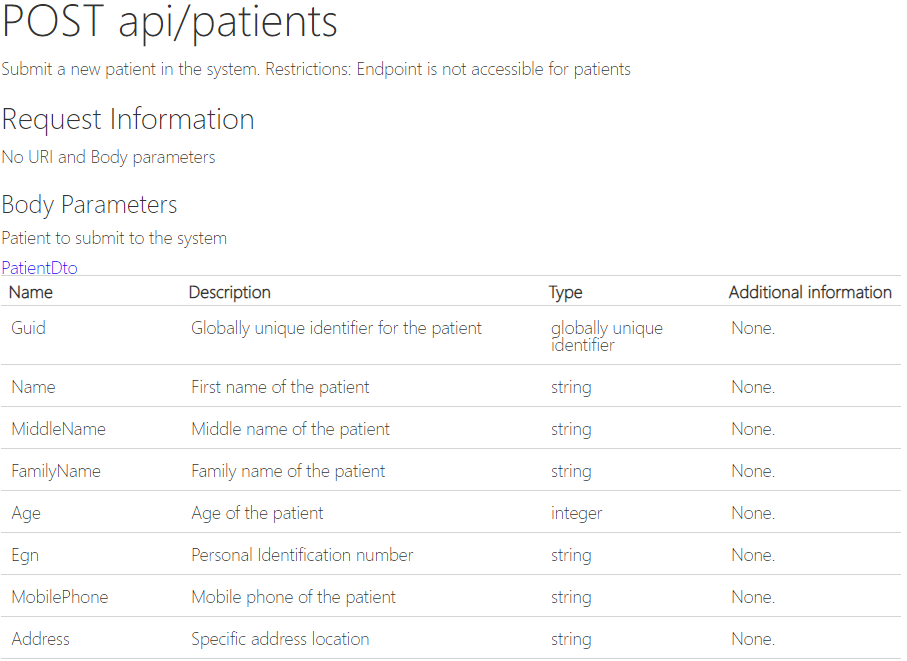
* 1. **Patients API**

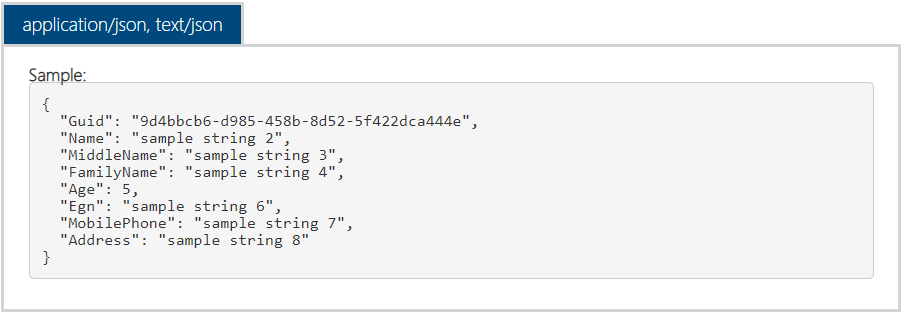
****

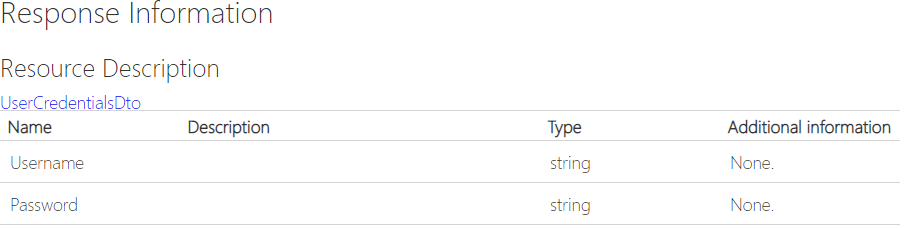


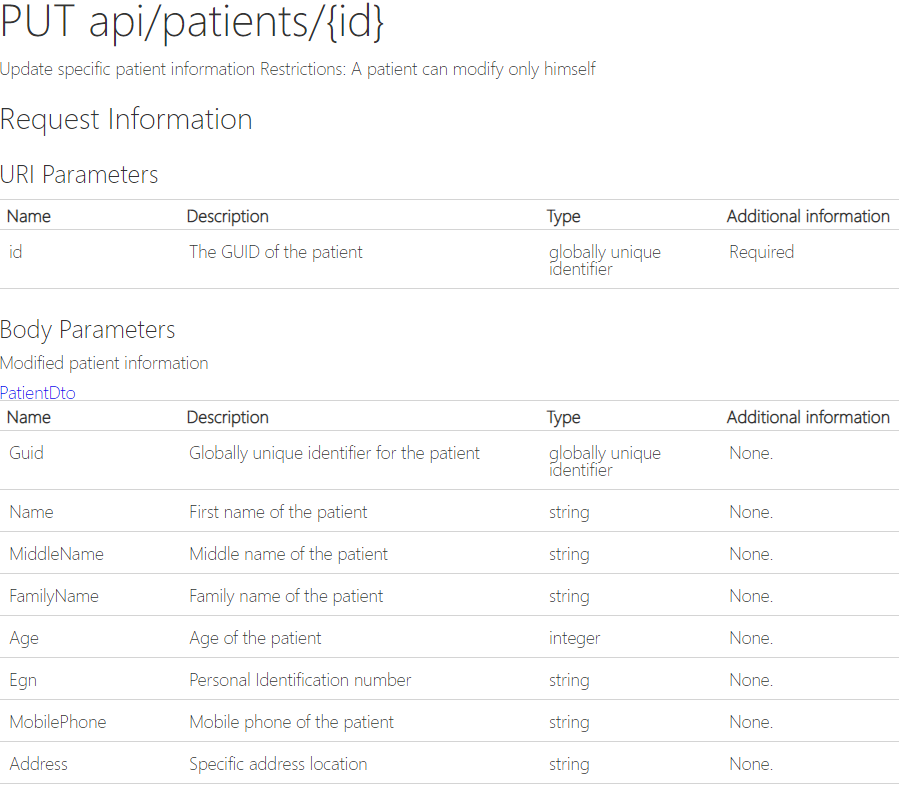


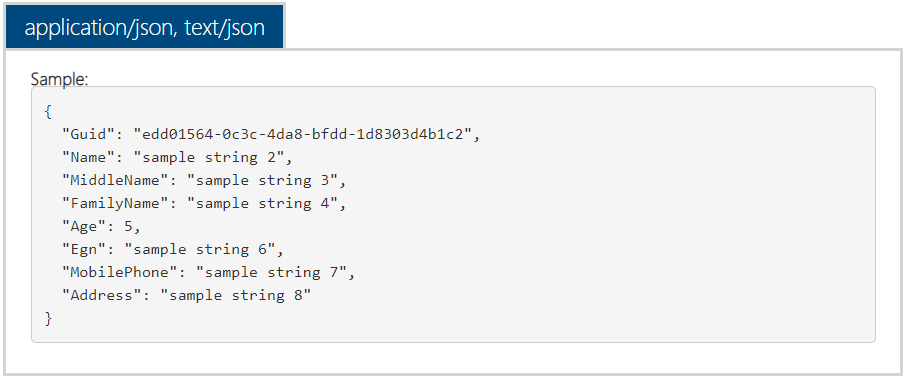


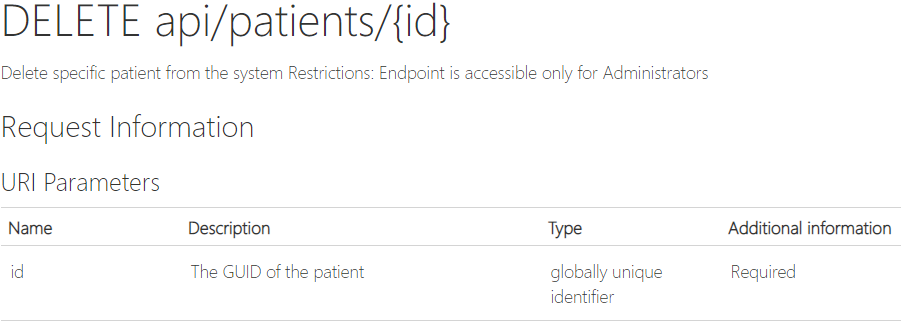




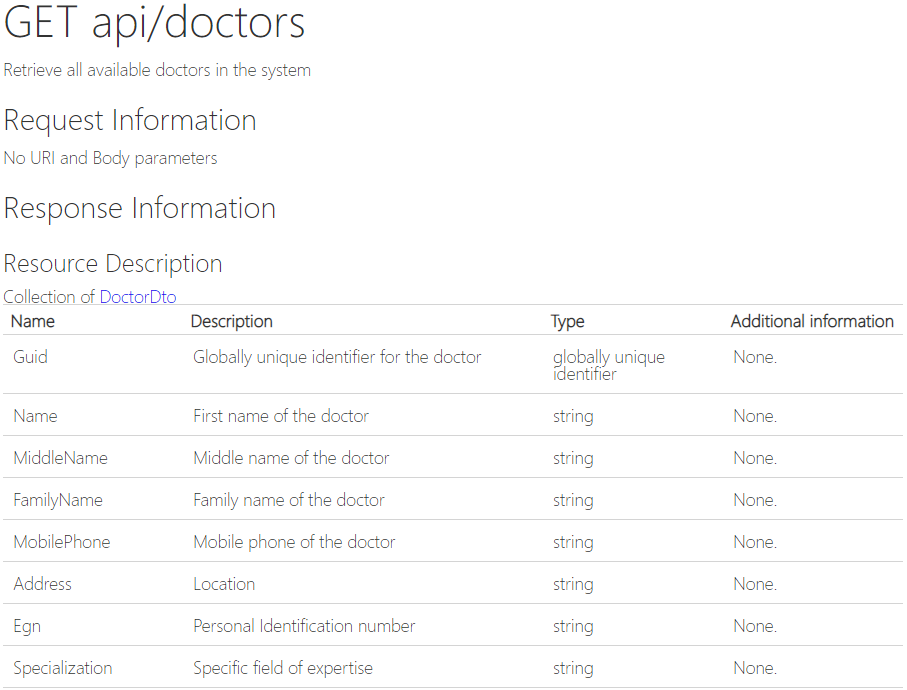




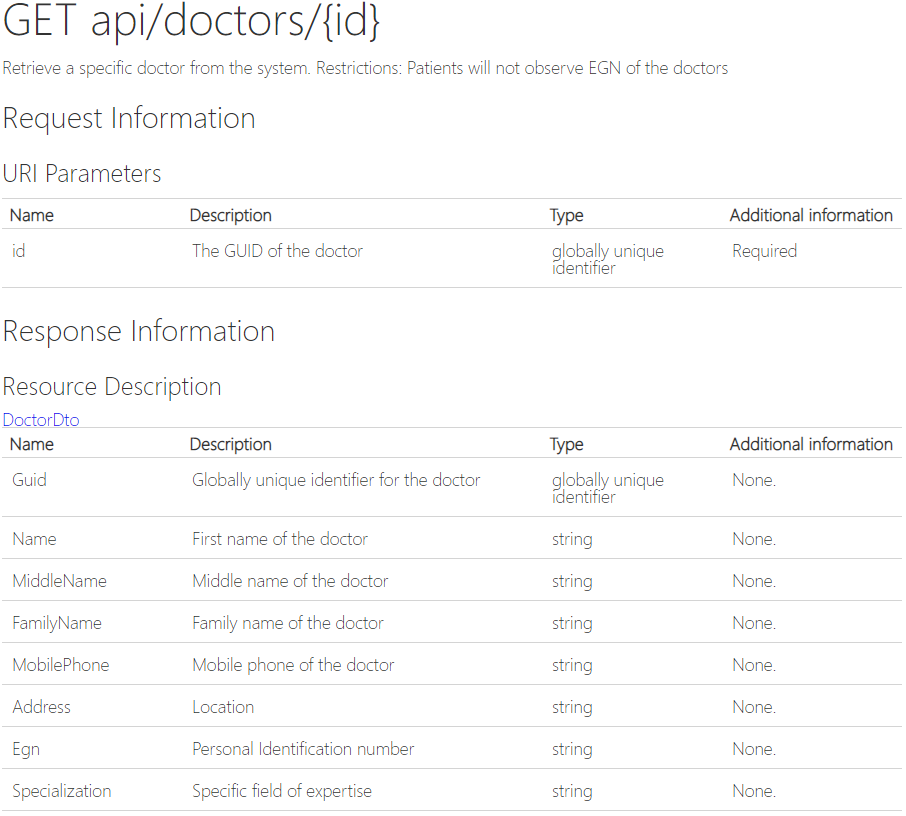


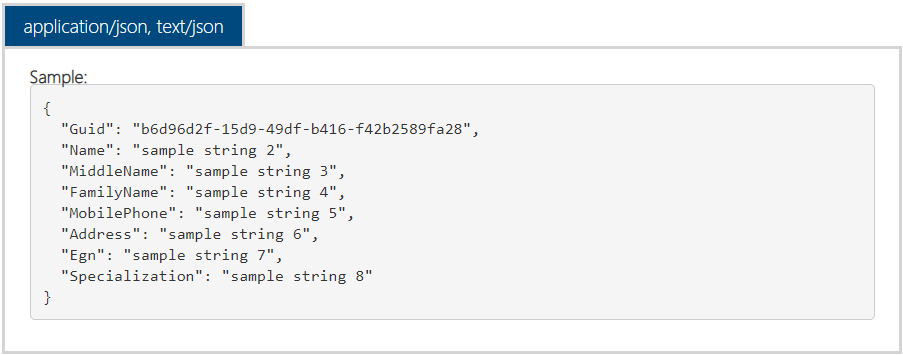


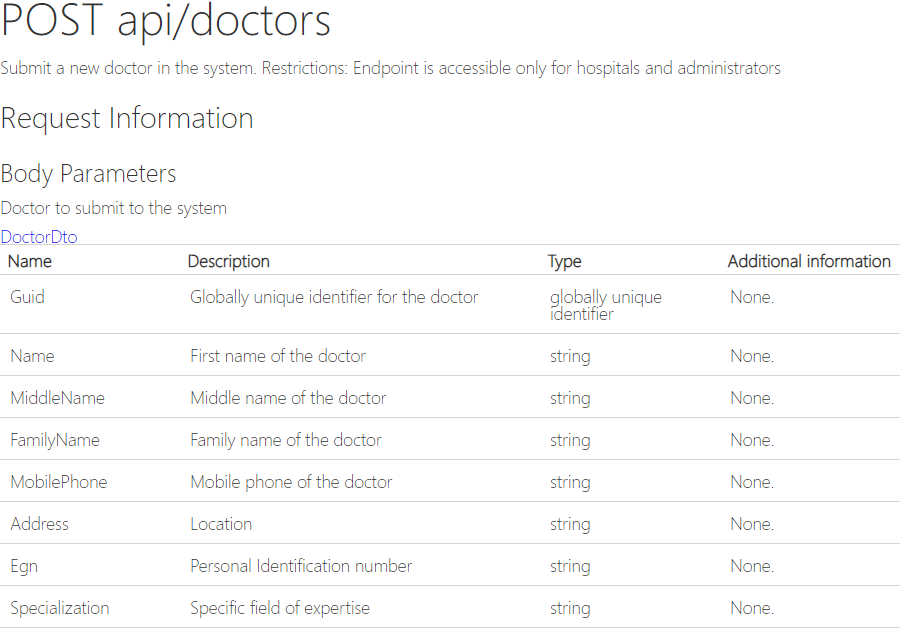
* 1. **Doctors API**

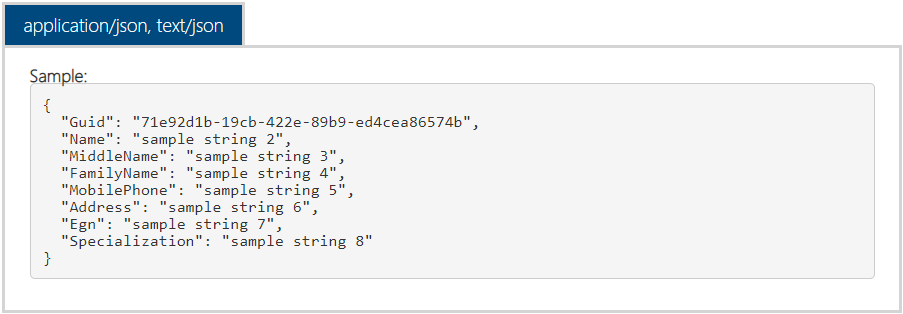


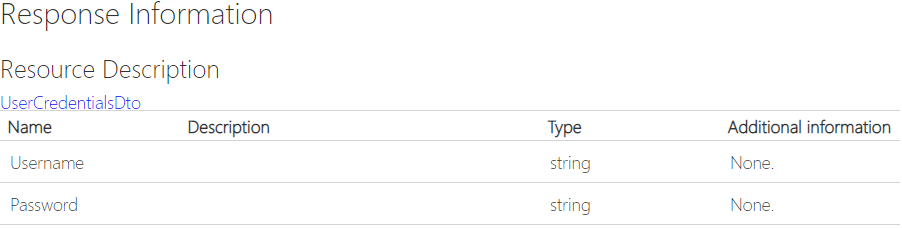


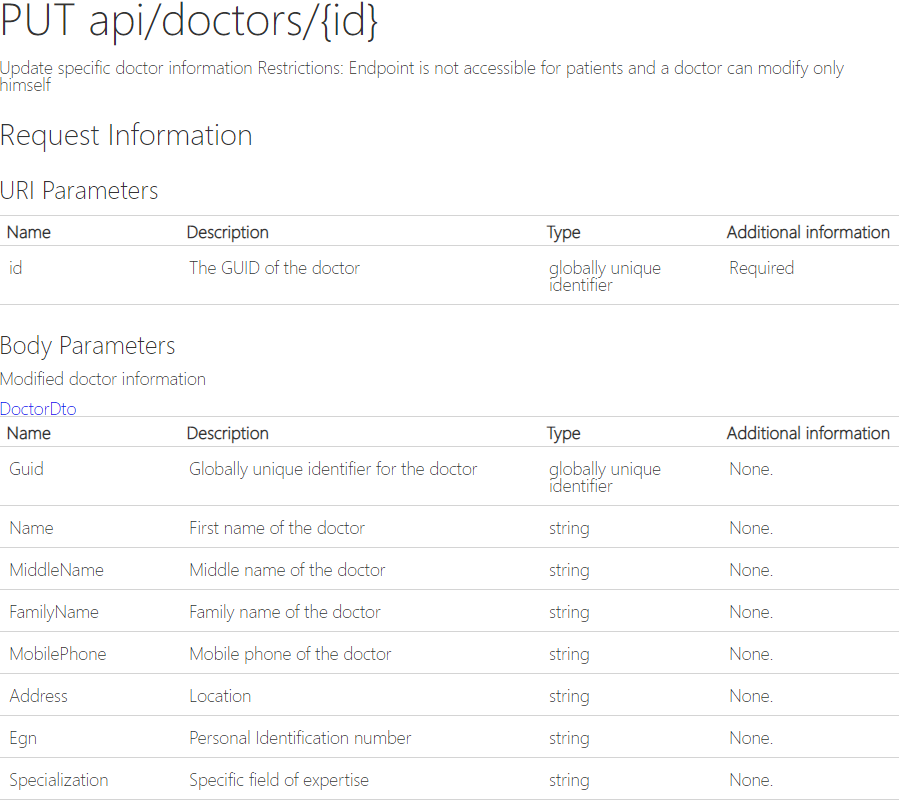




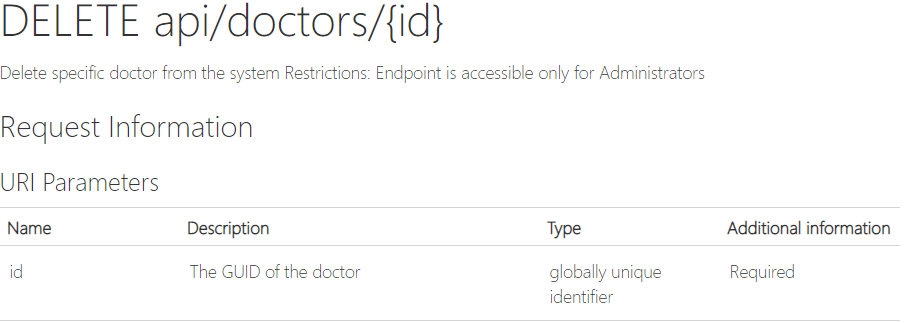




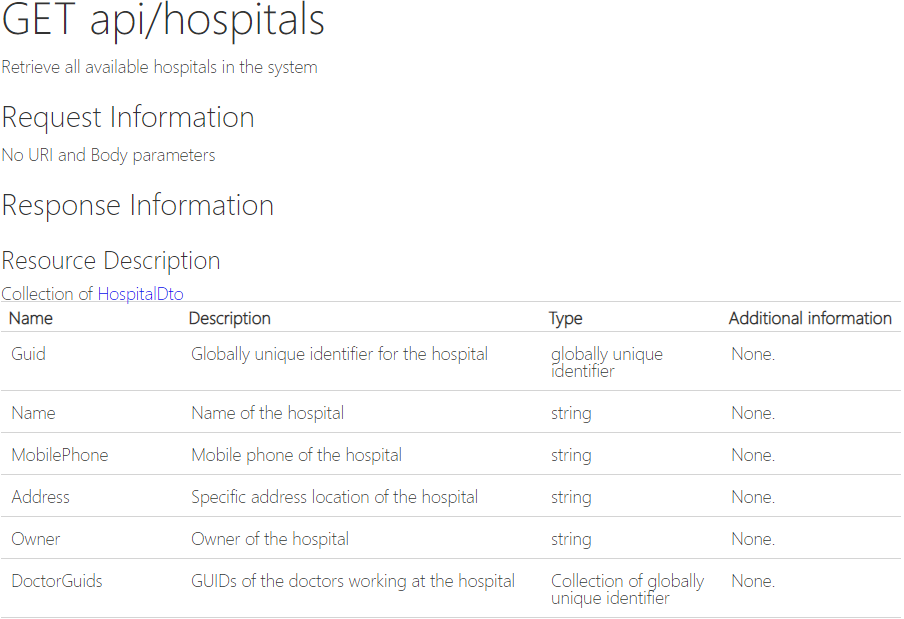




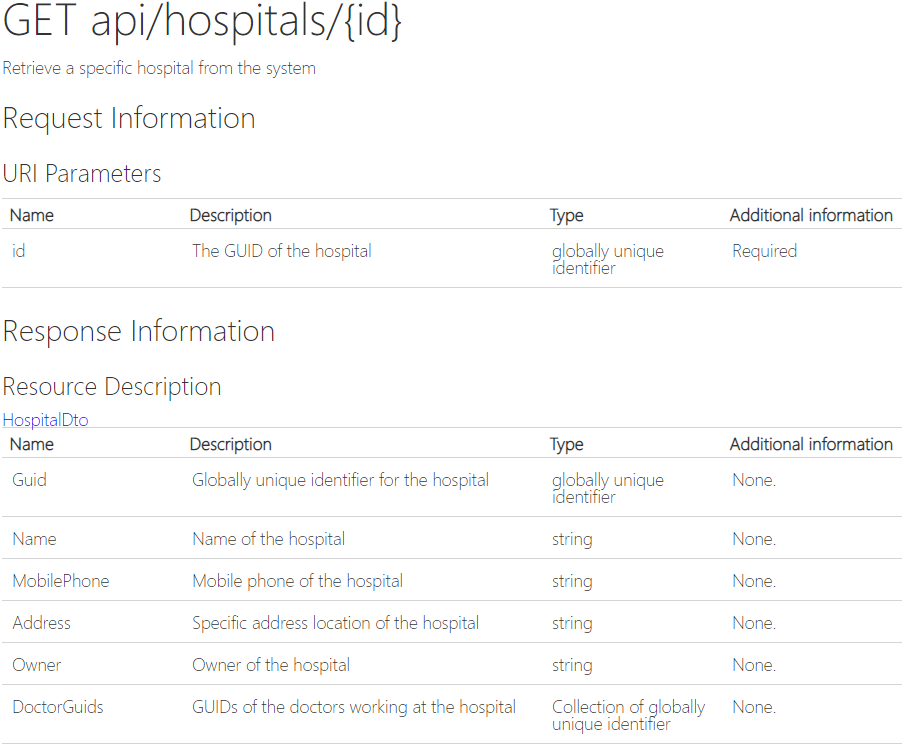




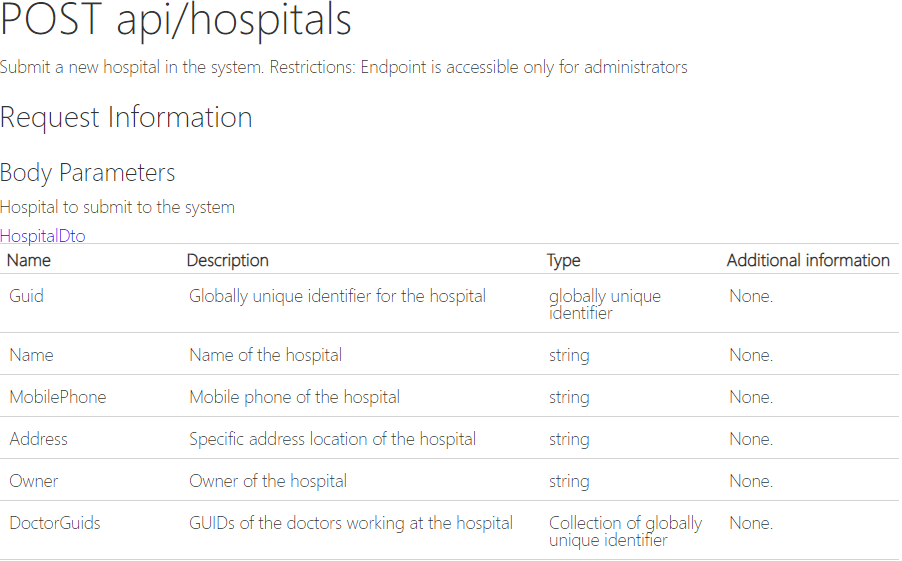
* 1. **Doctors API**



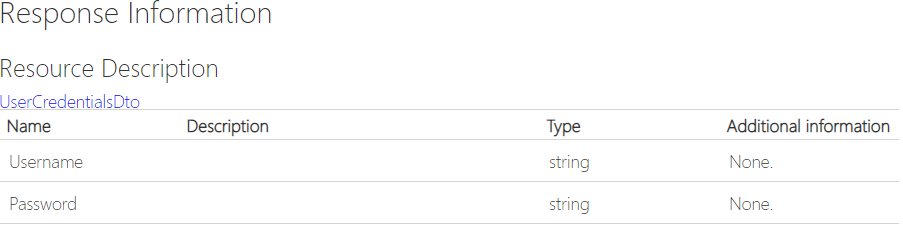


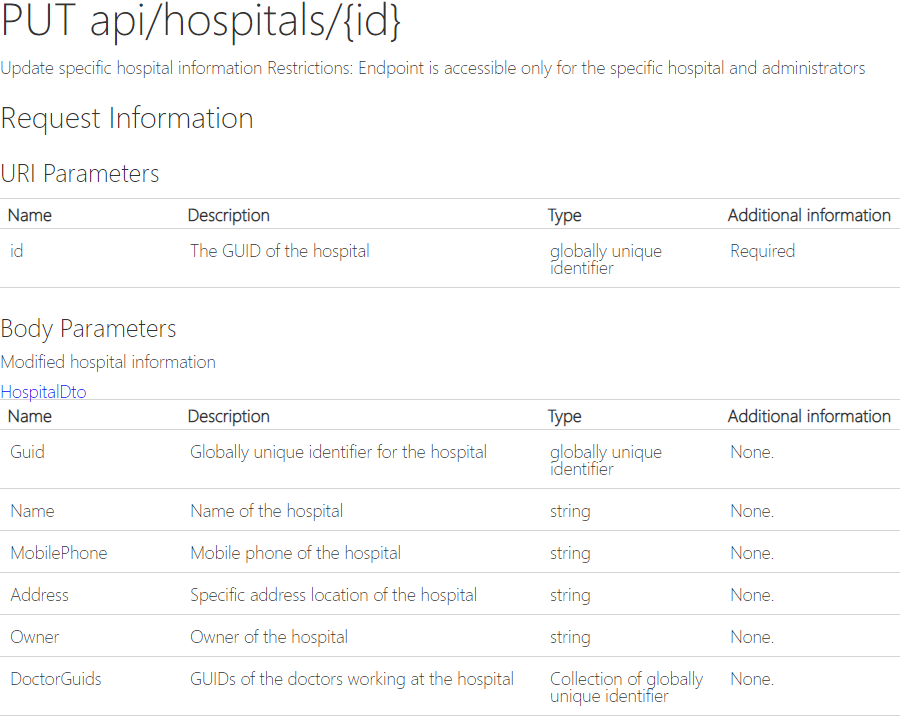




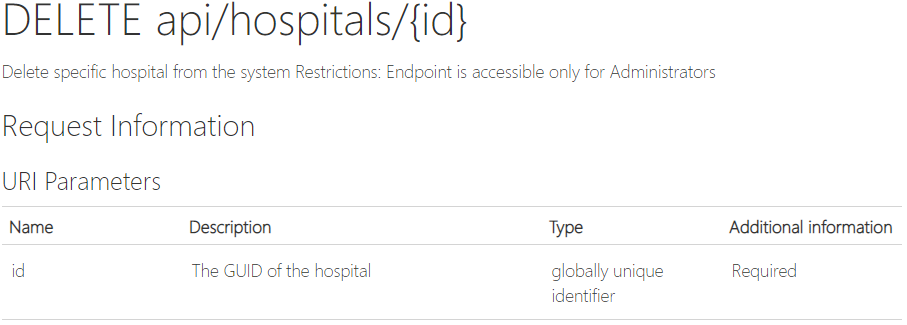


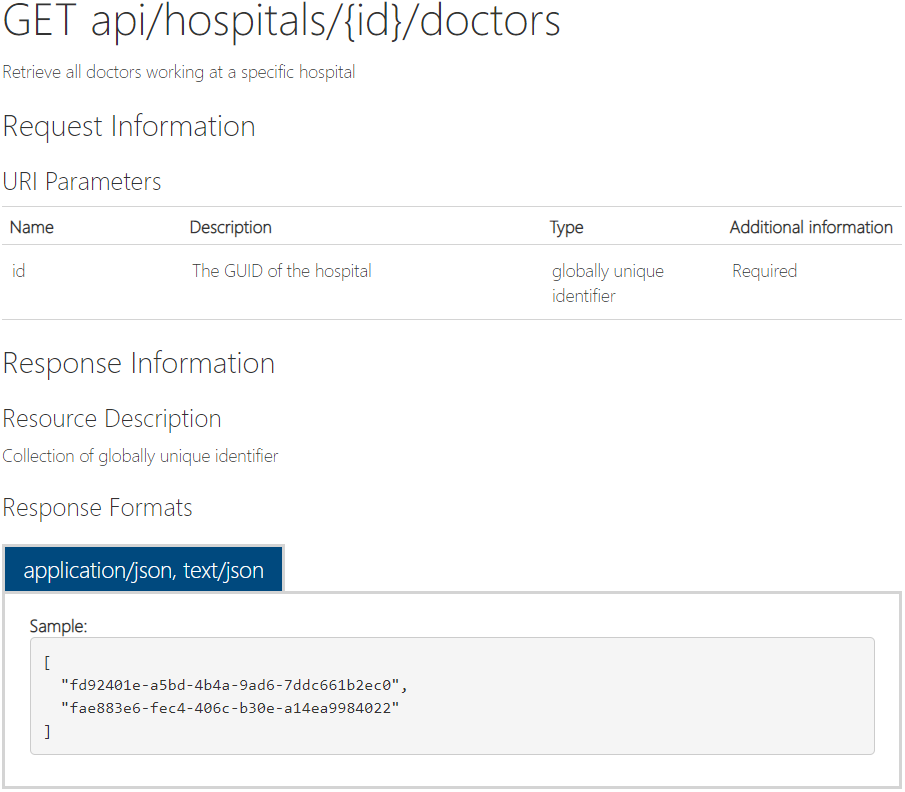


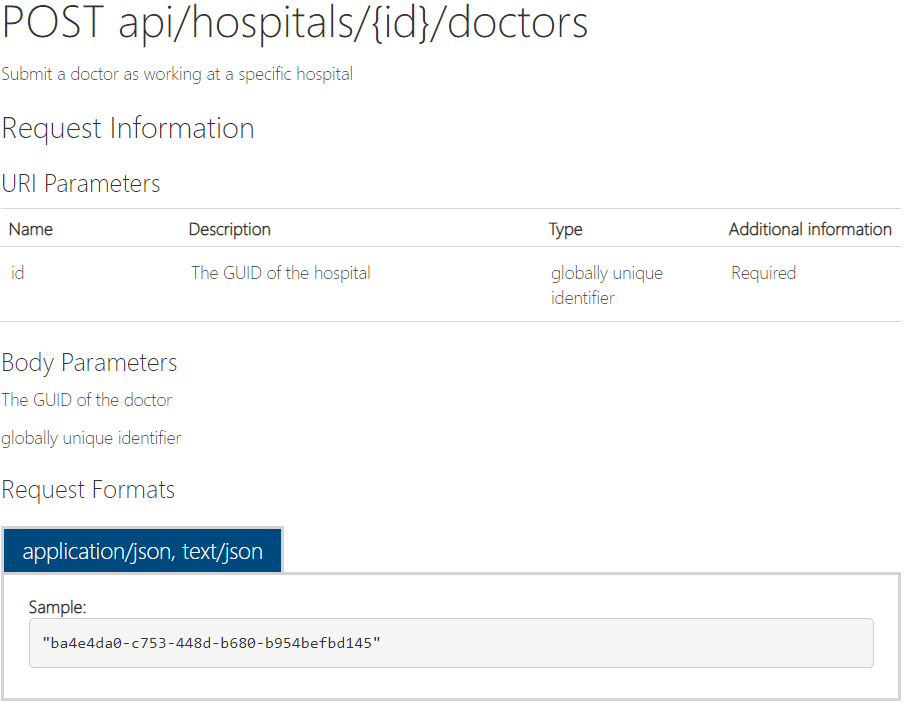


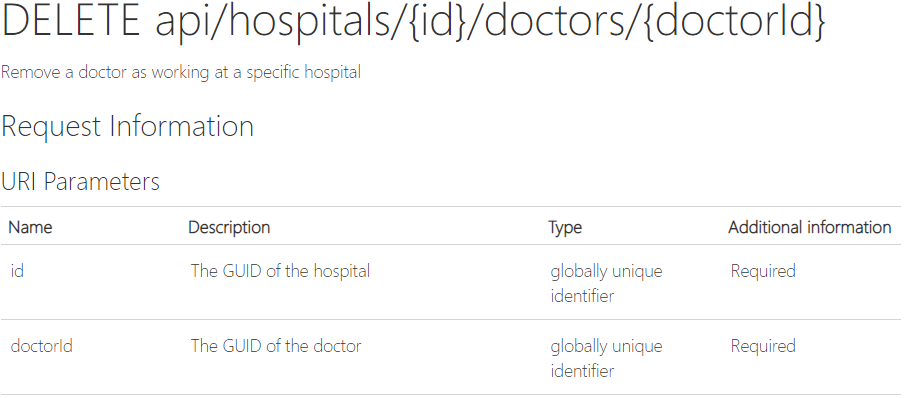




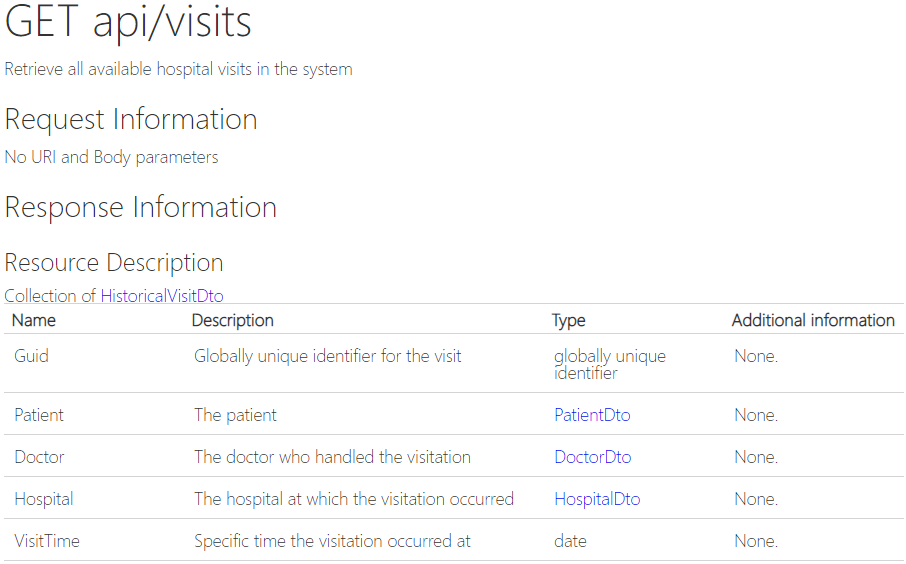






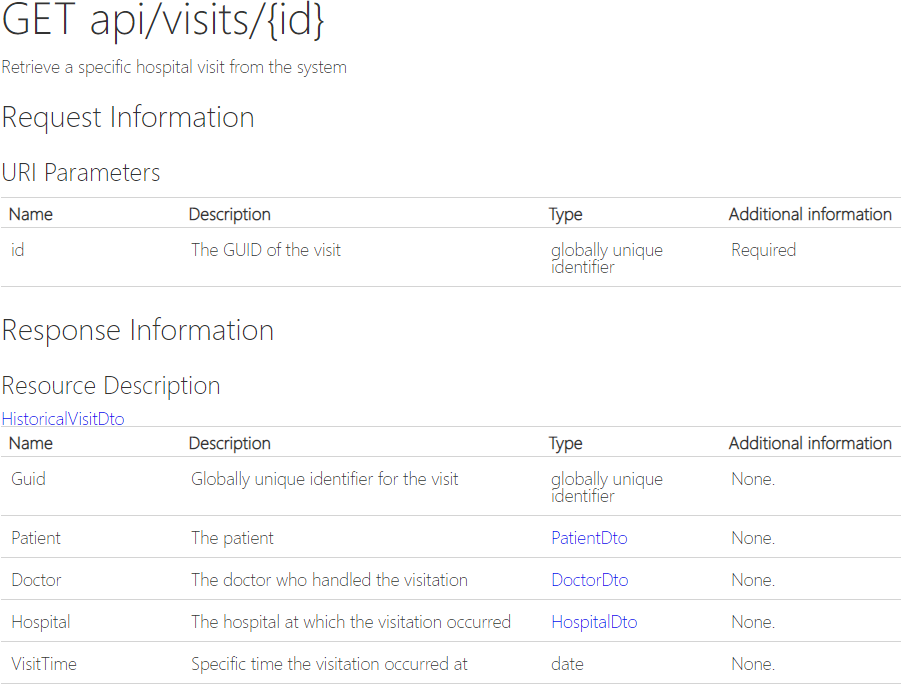


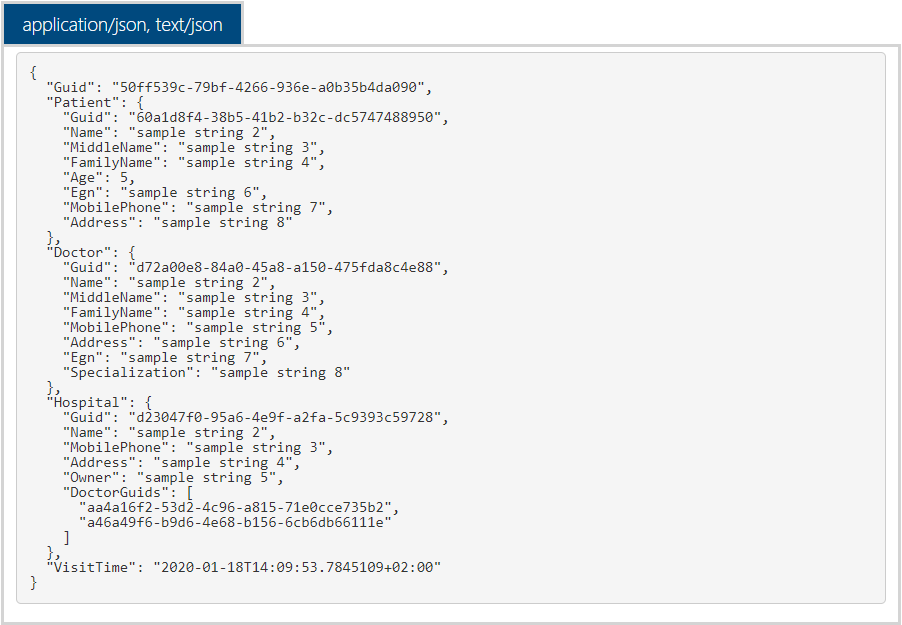
* 1. **Visits API**

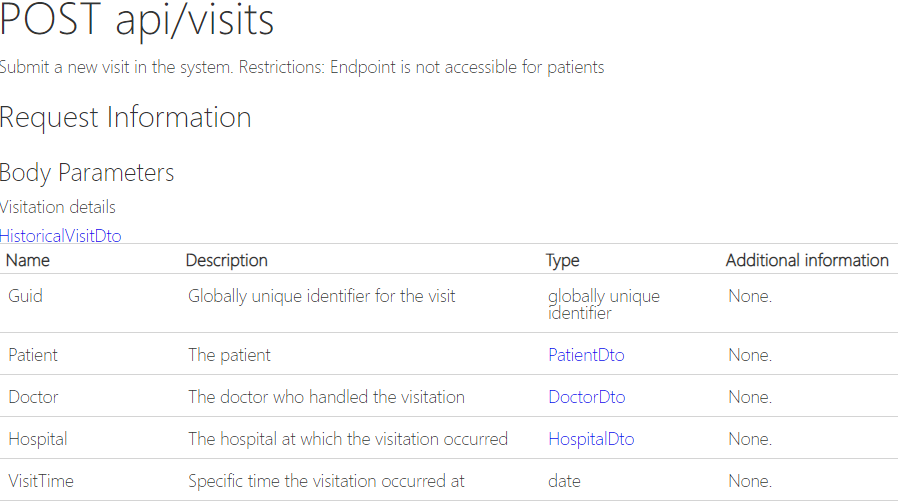


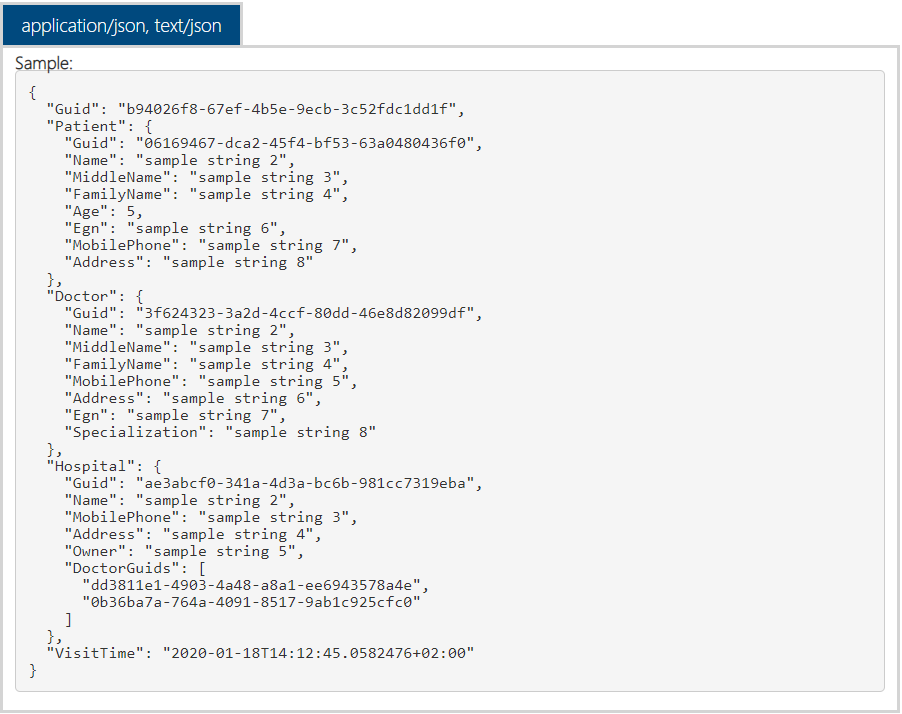


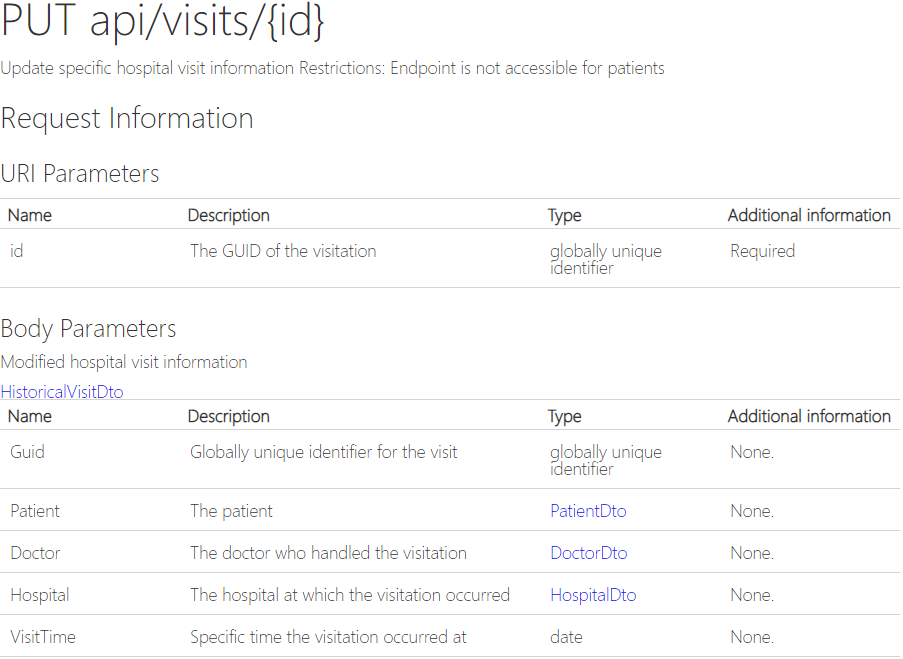


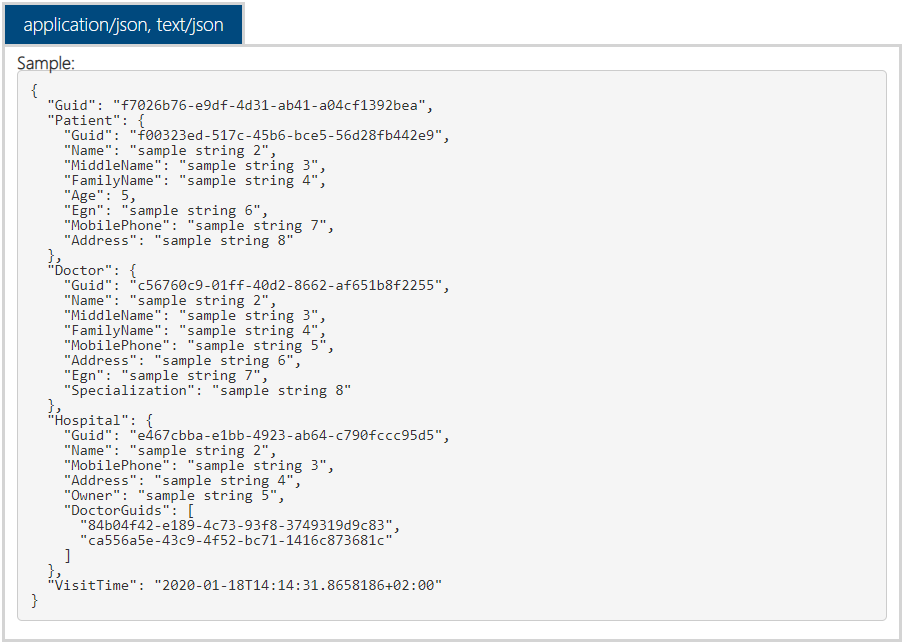


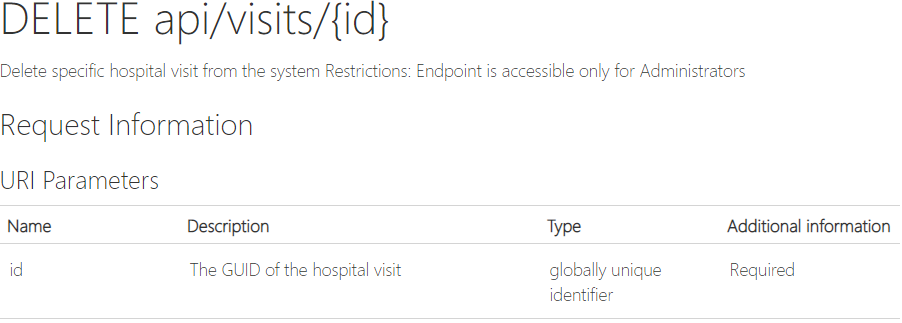




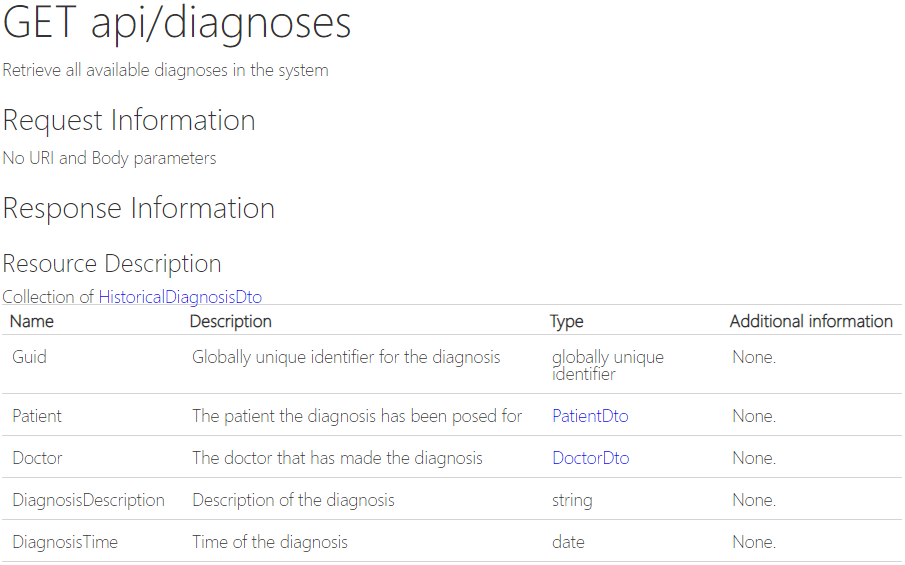




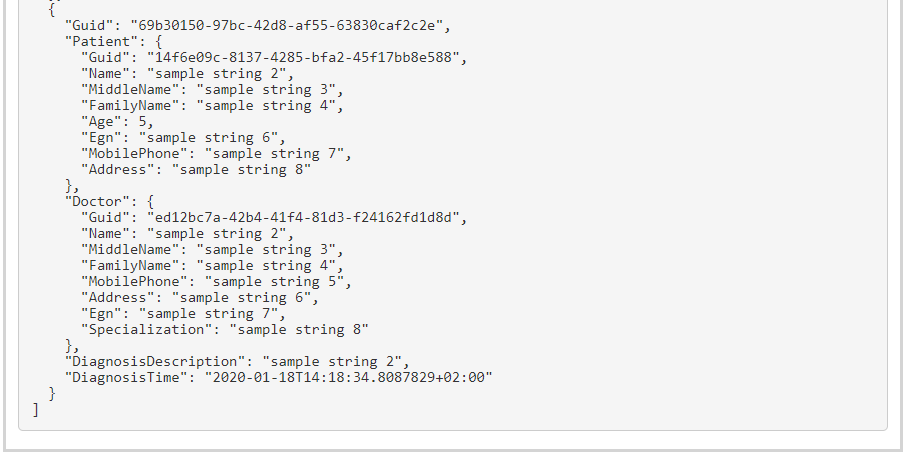


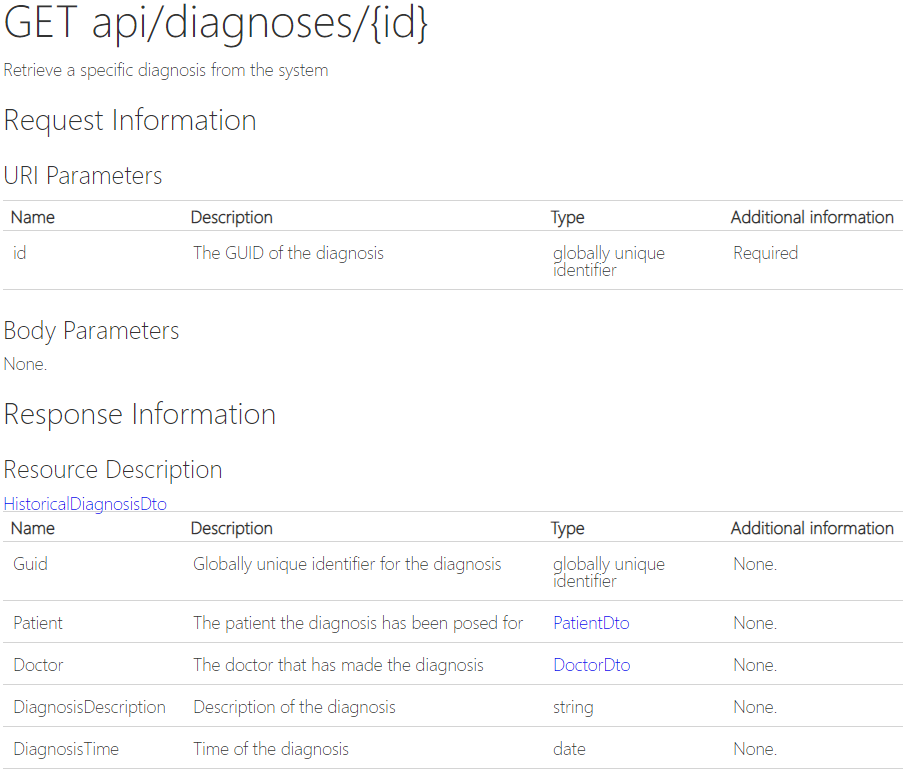


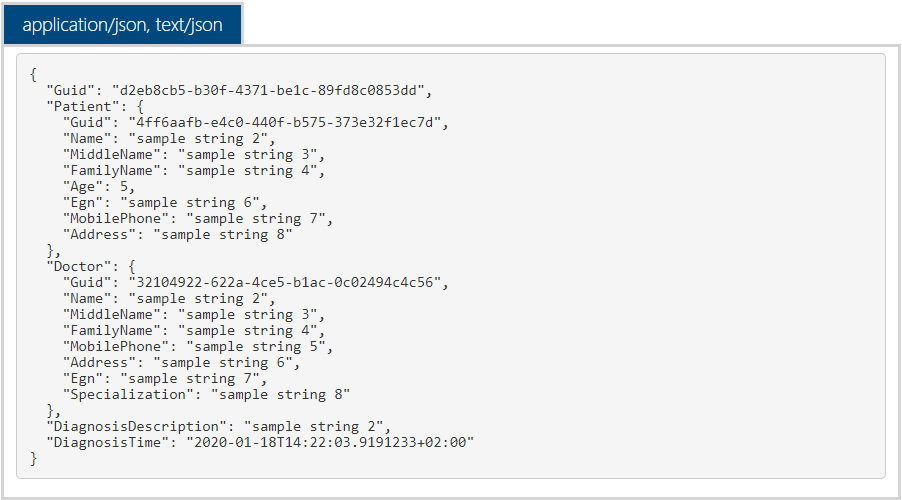
* 1. **Diagnoses API**

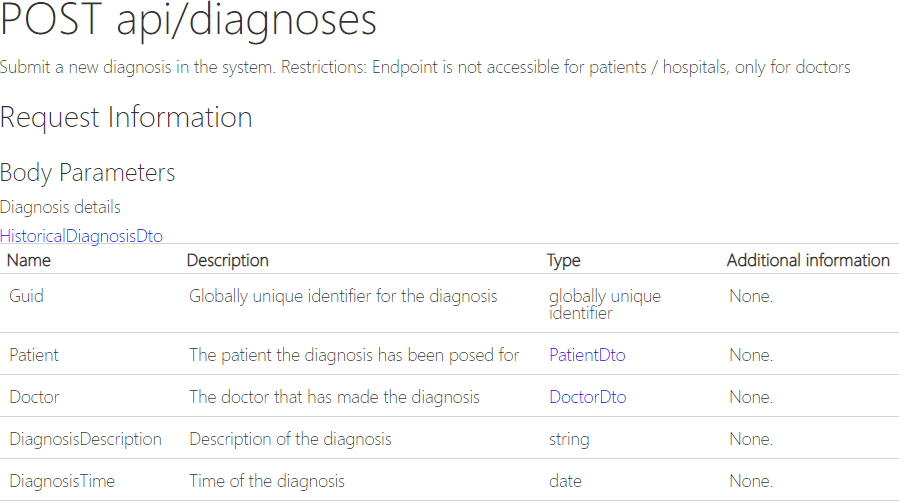
****

****

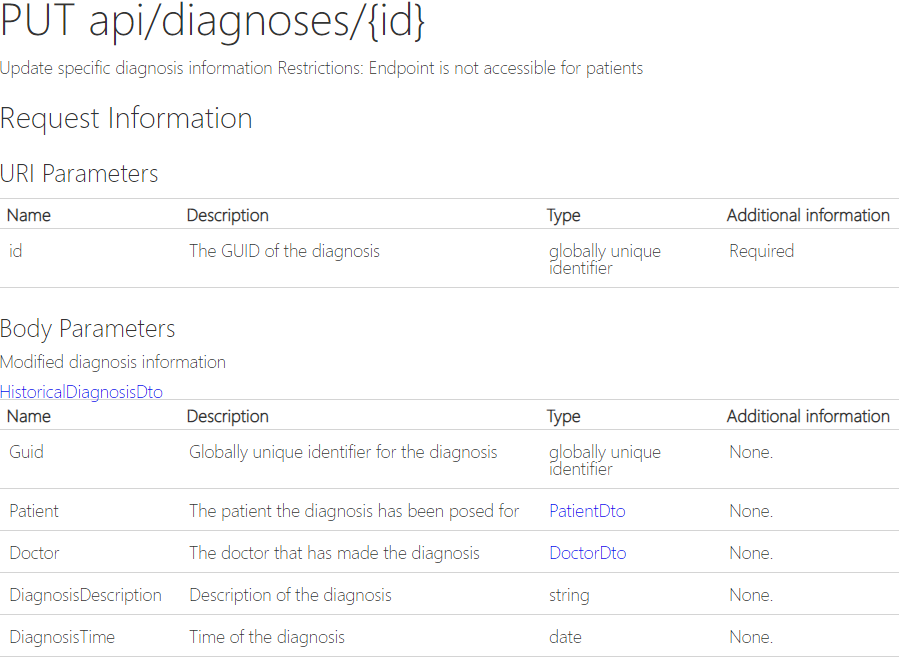
****

****

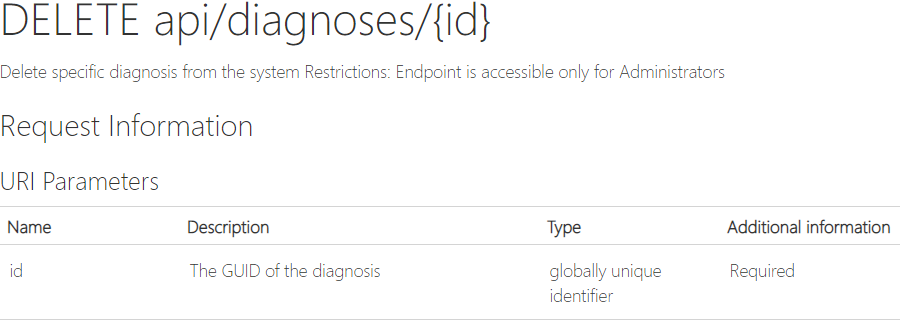
****

****

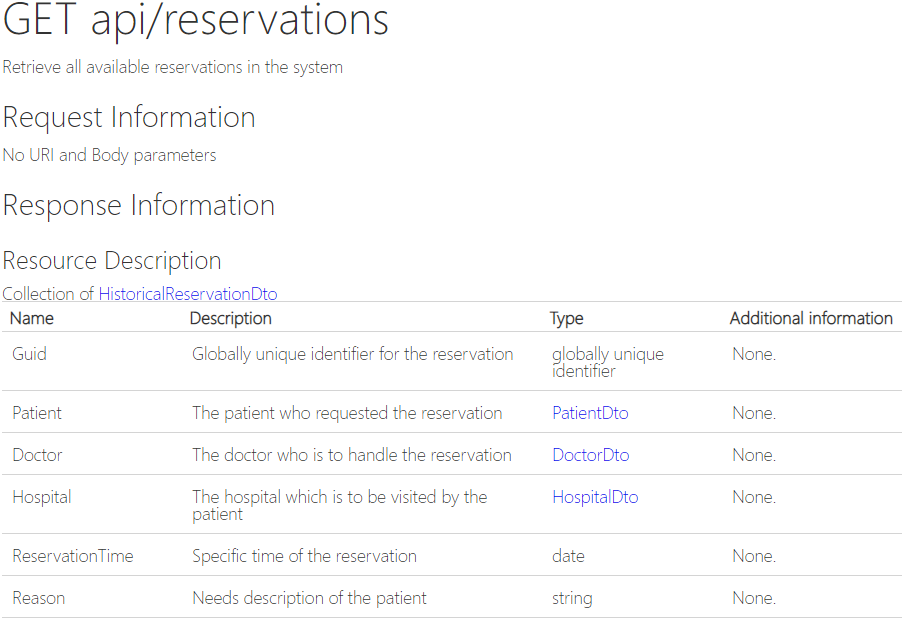
****

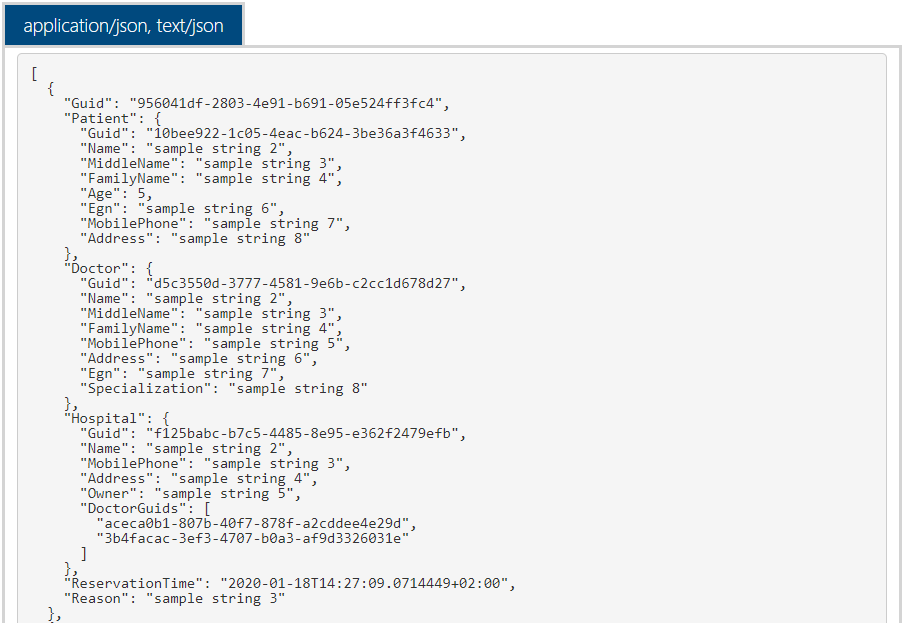
****

****

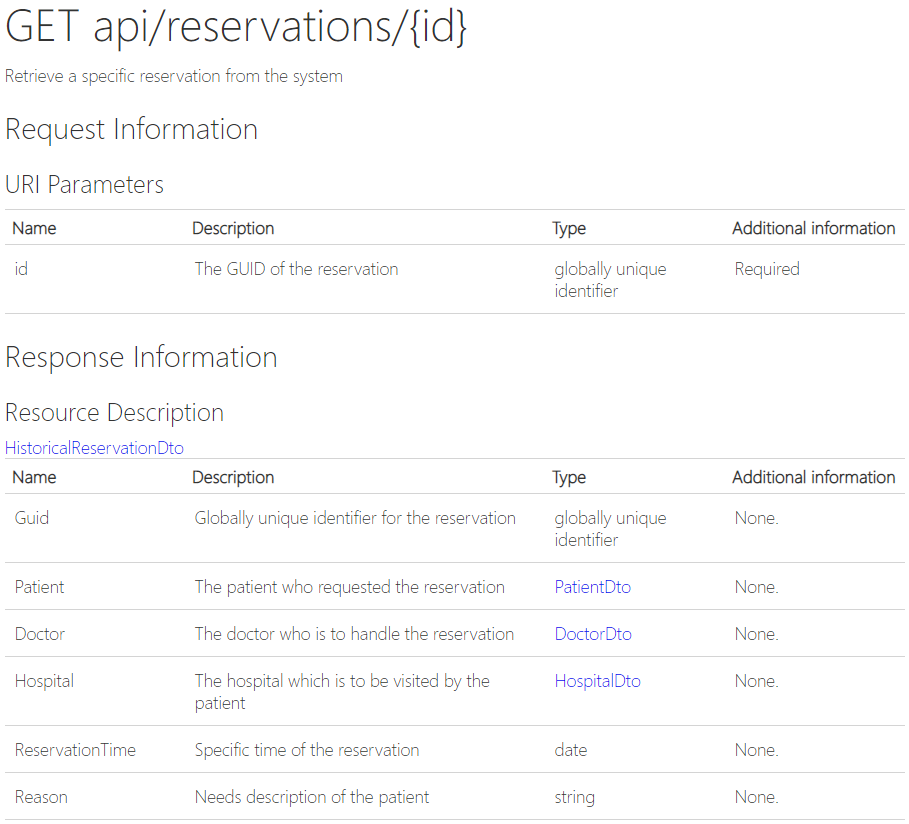
****

* 1. **Reservations API**

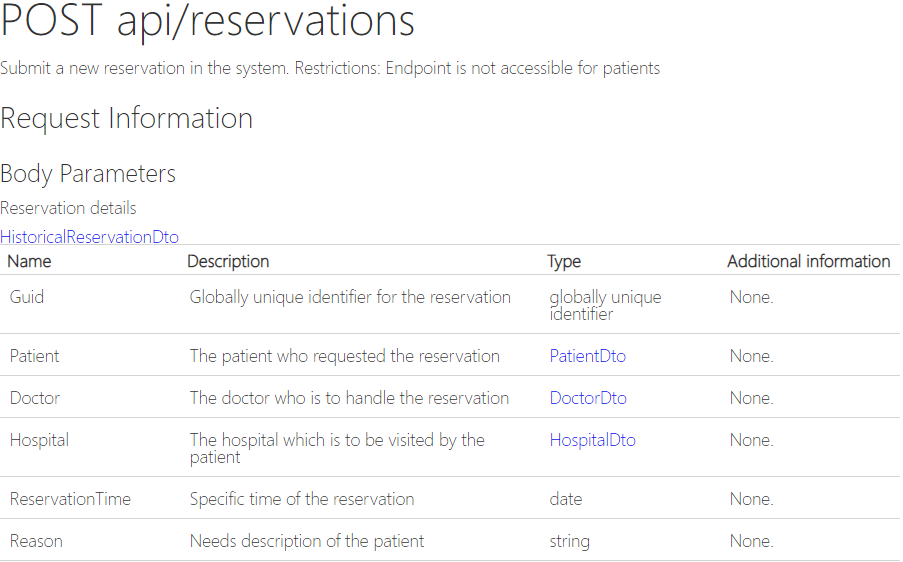
****

****

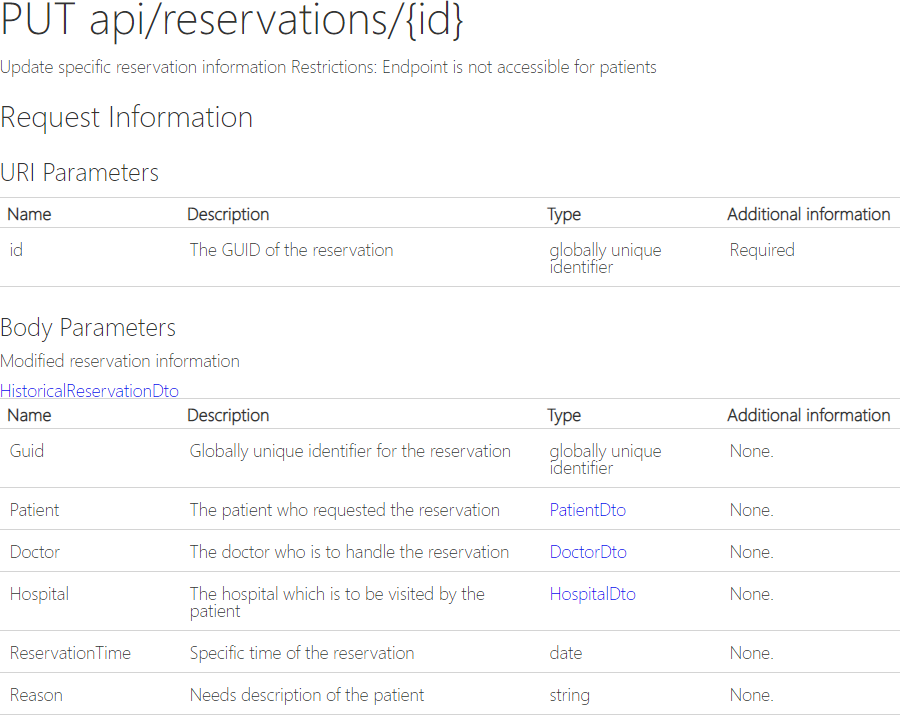
****

****

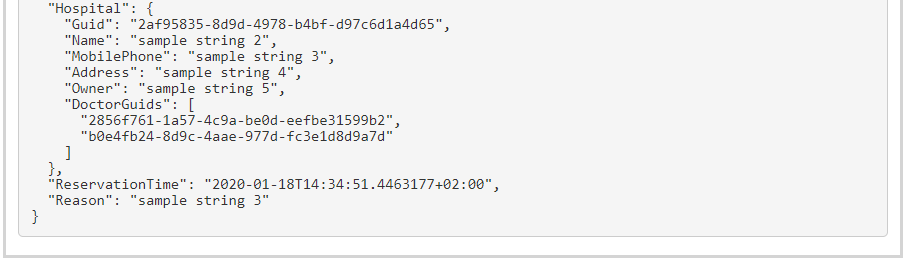
****

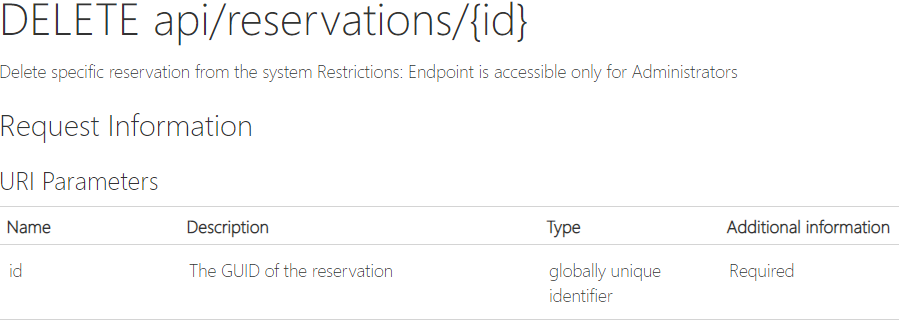
****

****

****

****

****

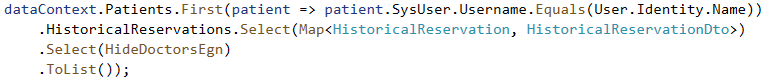
****

1. **AutoMapper**

Тъй като базата данни не винаги рефлектира точно данните които REST API-то трябва да връща, не е възможно да връщаме същите обекти.

За целта сме използвали DTO (Data Transfer Object), и извършваме Mapping между обектите от базата данни и DTO.

Това е видимо в по-предишния пример за работа с LINQ to SQL:

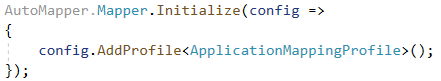


Тук .Select(Map<HistoricalReservation, HistoricalReservationDto>) ще извърши mapping.

Посредством AutoMapper това става лесно без нуждата от писане на код от наша страна.

Единственото изискване е конфигурирането на AutoMapper, а именно да уточним как се извършва този Mapping с конкретни полета.

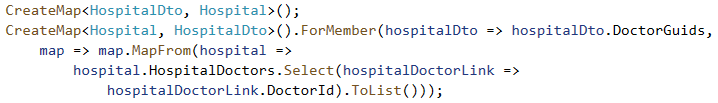
За целта е необходимо дефинирането на така наречен Profile и инициализирането на AutoMapper с него:



Самият клас ApplicationMappingProfile, създаден от нас наследява Profile:



И извършва създаването на Map дефиниция:



**Заключение**:

Нашият продукт е готов за употреба и среща нуждите на нашите потребители. Смятам, че достатъчно конкурентноспособен на база на вечете разработени функционалности.

Литература:

<https://dbup.readthedocs.io/> - DbUp, automated database creation/migration using scripts

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/> - LINQ to SQL

<https://www.tutorialspoint.com/linq/linq_sql.htm> - LINQ to SQL

<https://docs.automapper.org/en/stable/> - Automapper

<https://stackoverflow.com/>