ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ЗА ДИГИТАЛНИ НАУКИ „СОФТУНИ БУДИТЕЛ“, гр. София

**ДИПЛОМЕН ПРОЕКТ**

На Александър Ивов Боев

ученик/ученичка от XII а клас

професия- код: **481030**, **“Приложен програмист”**

специалност- код: **4810301**, **“Приложно програмиране”**

**Тема: Разработване на софтуер за управление на болници "HealthEdge"**

**Ръководител-консултант:** Мариян Йорданов

Сесия: май-юни 2024г.

Дата:.........................

**СЪДЪРЖАНИЕ**

[**1. Увод 3**](#_3znysh7)

[**2. Изложение 3**](#_2et92p0)

[2.1. Глава 1 (проучвателната част) 4](#_26in1rg)

[2.2. Глава 2 (съдържа изискванията към програмния продукт) 4](#_lnxbz9)

[2.3. Глава 3 (същинската част) 4](#_35nkun2)

[2.4. Глава 4 (ръководство на потребителя) 4](#_1ksv4uv)

[**3. Заключение 5**](#_2s8eyo1)

[**4. Информационни източници 6**](#_17dp8vu)

[**5. Приложения 7**](#_3rdcrjn)

*ВНИМАНИЕ! Всеки раздел и глава трябва да бъдат на нова страница!*

# Увод

"HealthEdge", се цели да бъде създадена среда, където взаимодействието между пациенти и медицински специалисти е улеснено, прозрачно и достъпно, независимо от времето и местоположението. Ангажиментът към иновациите и подобряването на здравеопазването води към разработването на решение, което не само отговаря на съвременните предизвикателства в бранша, но и предоставя стабилна основа за бъдещо развитие и интеграция на нови функционалности.

От друга страна, "HealthEdge" предлага на лекарите и медицинския персонал удобен инструмент за управление на тяхната дейност, включително преглед на заетост, планиране на задълженията и проследяване на пациентската история. Това дава възможност за по-ефективно разпределение на ресурсите и подобряване на качеството на медицинското обслужване.

## **Контекст и значимост на здравеопазването в България**

В контекста на българската здравна система, здравеопазването заема централно място със своята роля за поддържане на общественото здраве и благополучие. Секторът на здравеопазването представлява комплексно поле, което непрекъснато изисква модернизация и иновации за подобрение на услугите и управлението на здравните заведения. Тези усилия са жизненоважни за подобряване на качеството на медицинското обслужване, което влияе директно на здравето и продължителността на живота на населението.

## **Проблеми и предизвикателства в сектора на здравеопазването**

Намирането или предлагането на качествени лекарски услуги може да бъде времеемко и не винаги лесна задача. За да бъде полезно посещението и евентуалното лечение за пациента, той трябва първо да намери достатъчно квалифициран лекар, второ да намери начин да запази час за преглед, трето да съобрази евентуално своя график с графика на лекаря, както и евентуално дали специалистът работи с НЗОК и/или с някой здравноосигурителен фонд. Всяка една от тези нужди, без наличието на “HealthEdge” биха били изключително трудни и ненужно дълги за постигане. В един такъв хипотетичен сценарий, пациентът трябва първо да получи отнякъде контакт на достатъчно квалифициран специалист. След това трябва да намери начин да установи контакт с него, което най-често при старото поколение лекари, които отказват да се модернизират, се случва като отиде на място, изчаква всички хора, които са на опашка преди него и накрая да се разбере с лекаря, обикновено с компромиси от двете страни. Точно от такъв род проблеми решава “HealthEdge”. От друга страна, високо квалифицираните лекари, които искат да помагат на пациентите, биха били обречени на дълги години популяризиране, тъй като методът “от уста на уста” не е най-ефективното решение за популяризиране на лекар. “HealthEdge” отново може да реши този проблем. Всеки лекар има възможността да се представи накратко, както и да разкаже по-подробно за образованието си, за квалификацията си, както и кратка биография. Също така като доказателство за квалитета на лекаря са отзивите от вече посетилите пациенти, които могат да споделят повече за опита си при този лекар, както и да споделят градивна критика към лекуващия.

* 1. **Важността на софтуера за управление на болници**

Уеб приложението "HealthEdge" предлага решения на изброените проблеми чрез интеграция на съвременни технологии, които улесняват диагностиката, лечението и управлението на здравни заведения. Платформата автоматизира множество процеси, което позволява на медицинските специалисти да се съсредоточат повече върху пациентското обслужване отколкото на административни задачи.

## **Цел и обхват на проекта**

Целите на дипломния проект „HealthEdge” са да бъде разработена уеб базирана платформа за управление на лечебни заведения, която да предоставя на потребителите бърз и ефективен начин за намиране на лекар, базирано на оплакванията на пациента.

Специфичните цели на проекта включват следното:

* Разработване на потребителски интерфейс, който да е удобен, лесен за използване и да предоставя ясна информация за всеки лекар, който е наличен в даденото лечебно заведение.

* Интегриране на търсачка, която чрез филтриране на зададени характеристики от потребителя, като например пол, възраст, хронични заболявания, придружаващи заболявания, оплаквания и т.н., да предоставя на потребителите най-добрите специалисти, на бази филтрите и наличните лекари.

* Интегриране на търсачка, чрез която пациентите ще могат да намерят медицинско заведение. Филтрите, чрез които пациентите ще могат да постигнат намирането на медицинско заведение са: **Тип клиника, Населено място или по име на медицинско заведение (опционално)**.

* Интегриране на функционалност, чрез която пациентите ще могат да запазят час при избрания от тях лекар бързо и лесно, без нужда от допълнителна комуникация със специалиста.
* Създаване на AI модел, който ще **анализира** и изпраща информация към МЗ (Министерство на здравеопазването) за **количество заболели, техните диагнози и региона на пациентите** на месечна база, както на тримесечие и на годишна основа. Целта на модела е да помогне на националното здравеопазване за опазването на общественото здраве.

***(Планирано)***

* Интегриране на система за онлайн плащания, която да позволява на пациентите да заплащат своите прегледи, както и назначените медикаменти директно през уебсайта, без посредник.

***(Планирано)***

* Разработване на система за управление на продажбите, която да позволява на администраторите на сайта да упражняват мониторинг и администрация на запазените часове, с цел избягването на спам и недостоверни прегледи, както и  да проследяват статуса на прегледите и плащанията

***(Планирано)***

* Интегриране на чат система, чрез която ще се подобри комуникацията между пациент и лекар. Пациентът ще може да обменя с лекаря информация без нуждата от физическо посещение при специалист.

***(Планирано)***

* Модифициране на архитектурата на “HealthEdge”. Проектът да бъде изграден като **microservice architecture,** но при условие, че проекта се разрасне, съответно започнат да работят голям брой от хора, за да има смисъл от архитектурата. Това е така, тъй като микросървисната архитектура е скъпа за изграждане и разработка и има смисъл от нея за голям екип например. Това е така, тъй като осигурява независимост на екипа. Всеки domain или function може да бъде самостоятелно поддържан от назначен екип.

***(Бъдещо развитие)***

## **Предимства за потребителите на софтуера**

Потребителите на "HealthEdge" ще извлекат значителни ползи, включително по-бърз достъп до пациентски данни, подобрена координация между отделите, намаление на времето за обработка на медицинска информация и повишаване на ефективността при лечението. Софтуерът ще допринесе за намаляване на административните разходи и оптимизация на ресурсите в болницата.

* 1. **Предимства за потребителите на софтуера "HealthEdge" в контекста на намиране и предлагане на лекарски услуги**

Софтуерът "HealthEdge" предоставя множество предимства, които адресират гореспоменатите предизвикателства:

1. **Достъп до информация за лекари:** "HealthEdge" позволява на лекарите да представят своята квалификация, образование и специализации чрез детайлни профили. Това улеснява пациентите в избора на подходящ специалист посредством преглед на образователен и професионален фон.
2. **Електронно запазване на часове:** Платформата предлага функции за електронно запазване на часове, които минимизират необходимостта от телефонни обаждания и лични посещения.
3. **Оценки и отзиви:** Пациентите могат да оставят отзиви за своите преживявания с различни лекари, което помага на други потребители да направят информиран избор, базиран на качеството на медицинското обслужване.
4. **Синхронизация на графици:** Платформата позволява съобразяване на личния график с този на лекарите, облекчавайки процеса на планиране и намалявайки времето за чакане.

Тези функционалности на "HealthEdge" не само улесняват процесите на намиране и запазване на час при лекар, но и подобряват общото качество на медицинското обслужване, като правят здравната система по-достъпна, ефективна и отзивчива към нуждите на пациентите.

* 1. **Възможности за иновации и развитие на здравеопазването в България**

Здравеопазването в България, подобно на много други страни, стои на прага на технологична трансформация, която предлага значителни възможности за иновации и развитие. Софтуерът за управление на болници "HealthEdge" може да играе ключова роля в тази промяна, като внедри нови технологии и процеси, които значително подобряват ефективността, качеството и достъпността на медицинските услуги. Някои от основните направления за иновации включват:

1. **Интегриране на изкуствен интелект (AI) и машинно обучение (Machine learning):**
   * **Диагностика и лечение:** AI може да анализира медицински данни за предоставяне на предложения за диагностика и лечение, ускорявайки тези процеси и повишавайки тяхната точност.
   * **Персонализирана медицина:** Машинното обучение може да помогне за разработването на персонализирани лечебни планове базирани на индивидуалните характеристики на всеки пациент, като генетични данни и здравна история.
2. **Внедряване на блокчейн технологии:**
   * **Сигурност на данните:** Блокчейн може да предостави сигурна и непроменима база данни за медицинските записи, което минимизира риска от фалшификации и злоупотреби.
   * **Транспарентност в сделките:** Технологията може да се използва за проследяване и верификация на медицински трансакции и услуги, гарантирайки тяхната прозрачност и надеждност.
3. **Разработка на мобилни здравни приложения:**
   * **Дистанционно мониторинг:** Приложенията могат да позволят на пациентите да следят своето здраве от дома, предоставяйки данни в реално време на лекарите за по-добър надзор и управление на хронични заболявания.
   * **Подобрена комуникация:** Мобилните приложения могат също да улеснят комуникацията между пациенти и медицински специалисти, ускорявайки процеса на консултации и повишавайки удовлетвореността на пациентите.
4. **Умни медицински устройства:**
   * **Диагностични устройства:** Разработката на умни медицински устройства, които автоматично измерват и записват здравни показатели, може да помогне за ранното откриване и предотвратяване на заболявания.
   * **Роботизирана хирургия:** Въвеждането на роботи в хирургическата зала може да доведе до по-малко инвазивни, по-точни и с по-висок успех операции.
5. **Телемедицина:**
   * **Дистанционни консултации:** Развитието на телемедицински услуги позволява на пациенти в отдалечени или недостъпни райони да получават качествени медицински консултации без нужда от физическо присъствие в медицинско заведение.

Тези иновации не само ще подобрят качеството на здравеопазването в България, но също така могат значително да намалят разходите и да увеличат достъпа до медицински услуги за по-широк кръг от населението. "HealthEdge" предоставя платформата, необходима за интеграция на тези технологии, като по този начин спомага за трансформацията на здравната система в страната.

# Изложение

## **Използвани технологии**

В раздела с „използвани технологии“ ще бъдат разгледани технологиите, заемащи централно място в изграждането на „HealthEdge”. Технологиите са подбрани, за да осигуряват на стабилност, сигурност и ефективност на уеб-базираното приложение, като същевременно гарантират на клиента добро потребителско изживяване. Включени са инструменти за съхранение на данни, стилизиране на интерфейса, потребителска и лекарска автентикация, както и още много други. Съчетавайки всички тези инструменти, те осигуряват функционалност и удобство за крайния потребител.

### **Backend**

Първата стъпка в разработката на уеб приложения е създаването на backend частта. Това е именно частта от софтуерната архитектура на дадена уеб или софтуерна система, която обработва заявки и осигурява връзката между frontend-а (потребителския интерфейс) и базата данни. Той обикновено се състои от различни компоненти и технологии, които работят „зад кулисите“, за да осигурят функционалността и данните, нужни на потребителите. Backend-а се грижи за бизнес логиката на приложението, съхраняването и манипулирането на данни, обработката на заявки и осигуряването на сигурност.

A diagram of a software application

Description automatically generated

Фигура 1: Структурата на уеб приложението (технологии)

## **Какво е ORM и как функционира в Entity Framework?**

ORM е начин за конвертиране на данни между релационна база данни и обектно-ориентиран език за програмиране, чрез създаване на база данни с виртуални обекти, която може да се използва от обектно-ориентирания език за програмиране.

В Entity Framework това се случва чрез помощта на модели и миграции - процес на автоматично създаване и актуализиране на релационната база данни на основата на промените в моделите. Това означава, че Entity Framework Core може да проследява промените в модела на данните и да създава, актуализира или премахва таблиците и колоните в базата данни, когато е необходимо.

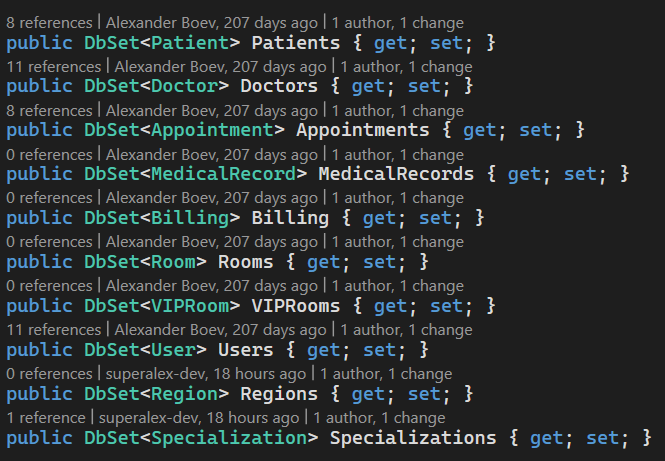
Един модел в Entity Framework представлява клас, който има различни property-та. Те могат да бъдат различни типове, като Entity Framework сам ще ги превърне в типове, които MSSQL server или PostgreSQL да разбере – например тип ***string*** ще бъде превърнат в тип ***nvarchar***, тип ***DateTime*** ще бъде превърнат в ***datetime2,*** тип ***byte*** ще бъде превърнат в тип ***tinyint*** и така нататък.

A screen shot of a computer code

Description automatically generated

Фигура 2: Модел в Entity Framework

Тези модели можем да превърнем в таблици, използвайки т.нар. ***DbSet***, където задаваме и желаното име на таблицата:



Фигура 3: Използване на DbSet в Entity Framework

След това трябва да имаме и конфигурация, която да посочи на Entity Framework как да намери / къде да създаде базата данни, която искаме да създаваме, използваме или модифицираме. Това можем да постигнем чрез метода **OnConfiguring**. Конкретно в случая на “HealthEdge”, connection string-а към базата се намира в application settings (appsettings.json), който бива игнориран от gitignore, чрез което се постига максимална сигурност за връзките в него. След като приложението спре да бъде в режим на development, следващата стъпка, която може да се предприеме, когато приложението е в продукционен режим да се използва криптиране на конфигурационния файл, използвайки **GitCrypt**. Това е инструмент за криптиране на файлове в **Git** хранилища, позволяващ безопасното съхранение на конфиденциални файлове дори в общодостъпни репозитории. При инициализация с **git-crypt init**, се създава криптиращ ключ, който се използва за автоматично криптиране на файлове, определени в **.gitattributes** файл. Криптираните файлове се добавят и комитват в хранилището в защитен вид, като могат да бъдат декриптирани само от потребители с достъп до съответния ключ.

Основните предимства на **GitCrypt** включват висока степен на сигурност, прозрачност в криптирането и декриптирането на файлове, улеснена споделянето на ключове чрез GPG, и лесната интеграция в съществуващи проекти. **GitCrypt** осигурява ефективно решение за управление на конфиденциални данни, като същевременно поддържа контрол и гъвкавост в достъпа до тях.

A screen shot of a computer program

Description automatically generated

Фигура 4: Метод OnConfiguring в Entity Framework

Методът **OnModelCreating** е важен за конфигурирането на модела на данните в приложението, като се дефинират взаимоотношенията, външните ключове и поведението при изтриване на записи. Използването на този метод е изключително нужно за поддържането на целостта на данните и правилното взаимодействие между различните същности в базата данни.

A computer screen shot of a program

Description automatically generated

Фигура 5: Метод OnModelCreating в Entity Framework

В този метод се настройват отношенията между **Doctor**, **Patient**, **Specialization**, и **Region** същностите, което позволява на приложението да управлява комплексни данни и взаимоотношения ефективно и безопасно. Той осигурява на **"HealthEdge"** да може да поддържа реалистични и точни медицински записи, което е от съществено значение за неговата функционалност и надеждност.

След като имаме модел и сме решили, че той ще бъде таблица, задавайки го като DbSet, трябва да направим миграция. Миграцията в Entity Framework се извършва с помощта на инструмента за миграции на Entity Framework, който позволява на разработчиците да генерират т.нар. миграционни скриптове и да ги приложат върху базата данни. Този процес включва изграждане на миграционна история, която записва всички промени в модела на данните, като например добавяне или премахване на таблица или колона. Когато се генерира нова миграция, инструмента за миграции автоматично генерира необходимите SQL команди за актуализиране на базата данни.

A screen shot of a computer program

Description automatically generated

Фигура 6: Миграционен скрипт в Entity Framework

## **Защо C# е добър при разработката на уеб приложения?**

Ето някои от основните причини защо **C#** е сигурен избор при разработването на уеб приложения:

* Обектно-ориентиран подход: **C#** е език със строга типизация, който поддържа обектно-ориентирано програмиране. Това позволява на програмистите да създават модулни и структурирани приложения, които са лесни за поддръжка и мащабиране.
* **ASP.NET: ASP.NET** е платформа за създаване на уеб приложения, която използва **C#** като основен език за програмиране. **ASP.NET** предоставя мощни инструменти за създаване на сигурни и ефективни уеб приложения.
* Голяма общност: **C#** е един от най-популярните езици за програмиране, който има голяма и активна общност от разработчици. Това означава, че има много ресурси, като форуми, уроци и документация на разположение.
* Лесно мащабиране: **C#** езикът и **ASP.NET** платформата предлагат много възможности за мащабиране на уеб приложенията. Това означава, че програмистите могат да добавят нови функции и да променят кода на приложението, без да се налага да го изграждат отново.
* Сигурност: **C#** и **ASP.NET** са добре известни със своята сигурност. Например, **ASP.NET** предоставя вградена защита срещу най-различни видове атаки, като SQL инжекции, кръстосан скриптинг и други.

A purple hexagon with a white circle and a white letter

Description automatically generated

## **Frontend**

След като backend-ът е готов, идва ред на реализацията на приложението в уеб среда, за което ни е нужен Frontend. Frontend-ът в едно уеб приложение е частта, която потребителите виждат и с която взаимодействат.

## **React и защо да използваме него?**

React е JavaScript библиотека за изграждане на потребителски интерфейси, особено подходяща за създаването на бързи и интерактивни едностранични приложения (SPA - Single Page Applications). Разработена от Facebook, React предоставя гъвкава и ефективна модел за разработка на уеб и мобилни приложения и включва следните основни характеристики:

* **JSX (JavaScript XML):** JSX е разширение на синтаксиса, което позволява комбинирането на HTML структура с JavaScript логика. Това улеснява разработчиците да описват интерфейси по декларативен начин и с висока четимост.
* **Components (Компоненти):** React е изграден върху идеята за компоненти – изолирани блокове код, които управляват своето собствено състояние. Те позволяват разработчиците да изграждат сложни UI структури, които са лесни за управление и повторна употреба.
* **One-way data binding (Еднопосочно свързване на данни):** В React, данните текат в една посока от родителски към дъщерни компоненти, което улеснява управлението на сложни приложения и подобрява производителността.
* **Virtual DOM (Виртуален DOM):** React използва концепцията за Виртуален DOM, където промените в интерфейса се прилагат първо във виртуална копия на DOM, което минимизира броя на операциите в реалния DOM и подобрява производителността на приложенията.
* **Lifecycle methods (Методи на жизнения цикъл):** Компонентите в React имат множество „жизнени“ методи, които се извикват в определени моменти от жизнения цикъл на компонента (например при създаването, обновяването или унищожаването му). Тези методи предоставят голяма гъвкавост и контрол върху поведението на компонентите.
* **Hooks (Куки):** Въведени във версия 16.8, куките позволяват на функционалните компоненти да използват състояние и други React характеристики без да се налага да са класове. Това прави кода по-кратък и по-лесен за разбиране.

• **Testing (Тестване):** React може да се интегрира с инструменти като Jest за тестване на компоненти в изолация, което улеснява тестването и поддържането на голям мащаб от приложения.

Използването на React в проектите предлага значителни предимства като ефективност при обработка на данни, голяма екосистема от библиотеки и инструменти, и широката общност от разработчици, които непрекъснато допринасят за нейното развитие. Това го прави предпочитан избор за разработка на съвременни уеб приложения.

## **Компонентите в React**

В React, компонентите са основните строителни блокове на приложението. Компонентите са самостоятелни единици, които обхващат HTML, CSS и JavaScript кода и позволяват на разработчиците да създават многократно използваеми UI Компоненти за приложенията си.

A screenshot of a computer

Description automatically generated

Фигура 7: Компоненти в React.js

В конкретния пример от фигура 7, в папката с компонентите, използвани за view-та, свързани с докторите, имаме 3 компонента. Първият – DoctorCard.js е компонент, който се използва за показване на информация за лекар. Той извлича и показва от сървъра най-ранния свободен слот за среща с лекаря. Ето разбивка на неговата функционалност:

Той приема обект на лекар като реквизит. Този обект трябва да съдържа информация за лекаря, като например неговия идентификатор, име, снимкаUrl, специализация, рейтинг и град.

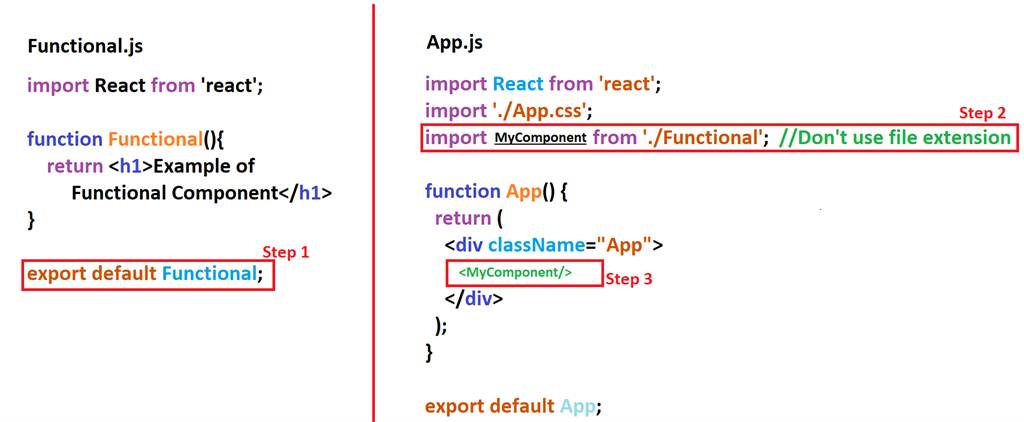
Използва куката useState, за да създаде променлива на състоянието earliestSlot, в която се съхранява най-ранният наличен слот за среща с лекаря.

Куката useEffect се използва за извличане на най-ранния наличен слот от сървъра, когато компонентът се монтира или когато doctor.id се промени. След това изтеглените данни се задават в променливата на състоянието earliestSlot.

Компонентът връща JSX, който показва снимката на лекаря, името, специализацията, рейтингите, града и най-ранния наличен слот за среща. Включва и бутон за резервиране на среща.

StarIcon от @heroicons/react/solid се използва за показване на 5-звездна оценка за лекаря.

Компонентът е експортиран за използване в други части на приложението.



Фигура 8: Пример за принципа на работа на компонентите в React.js

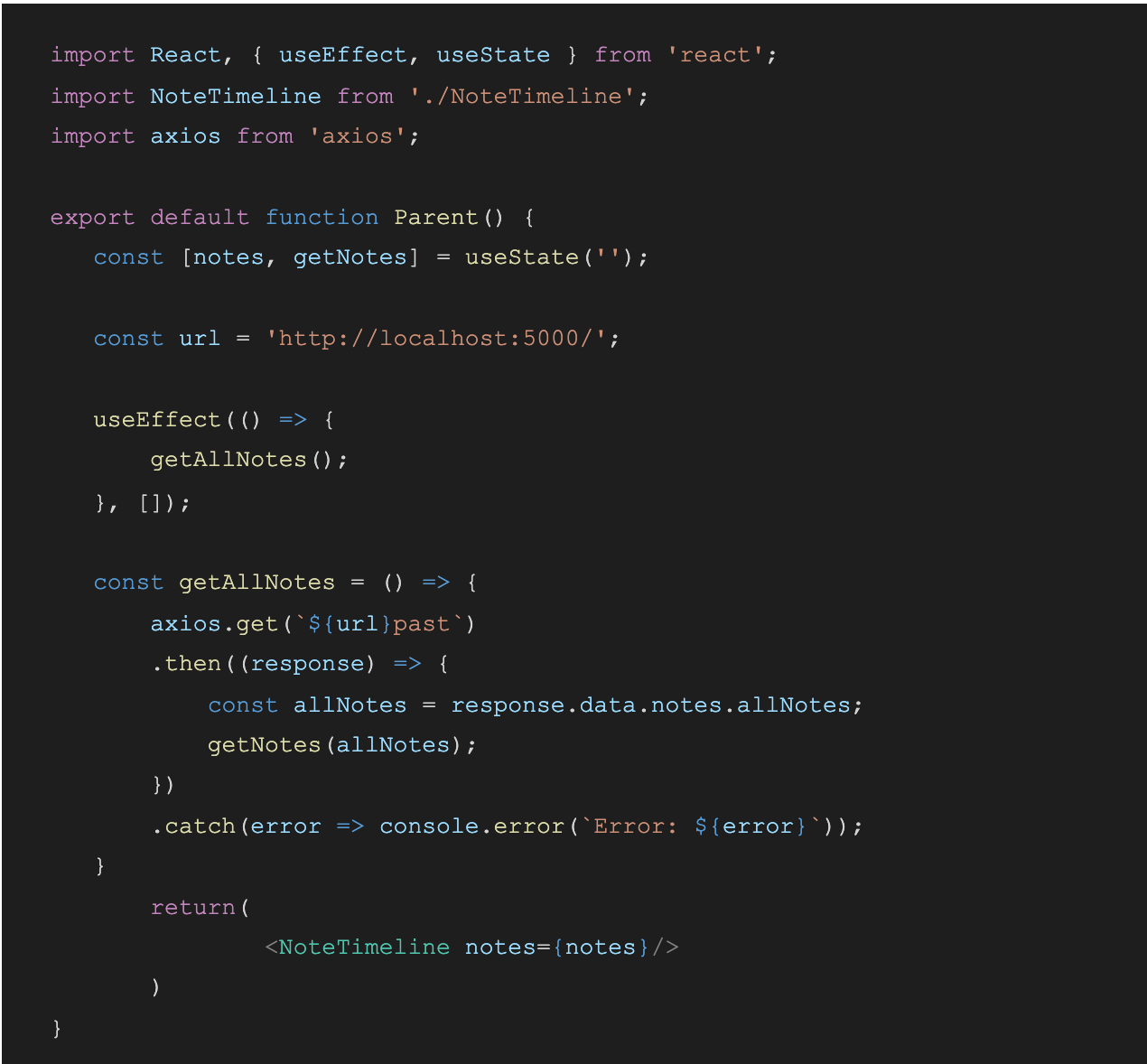
## **Service-и в React**

В React.js, сървисите играят ключова роля за облекчаване на обмена на данни между фронтенда и бекенда на уеб приложението. Ефективното използване на услугите е съществено, тъй като насърчава повторното използване на съществуващия код, което е ключов аспект в оптимизацията в разработката на софтуер.

В типично приложение на React.js, особено когато е интегрирано с бекенд на ASP.NET Core, използващ Entity Framework, услугите се използват за взаимодействие с бекенда чрез HTTP заявки. Ето най-често използваните видове HTTP заявки:

* GET: Извлича информация или данни от сървъра за определен ресурс. Този метод често се използва за получаване на страници, изображения и други файлове от сървъра. В контекста на React.js и ASP.NET Core, заявка GET може да се използва за извличане на данни от база данни чрез операции на Entity Framework, които са предоставени чрез API крайни точки.
* POST: Изпраща данни към сървъра за обработка, обикновено като част от изпращане на формуляр или за създаване на нов ресурс. Например, когато потребител изпраща формуляр в приложение React за добавяне на нов запис в базата данни, заявка POST може да изпрати тези данни към ASP.NET Core API, което след това обработва вмъкването на данните с помощта на Entity Framework.
* PUT: Актуализира съществуващ ресурс на сървъра с нови данни. В интеграцията на React.js с ASP.NET Core, заявка PUT може да се използва за актуализация на запис в базата данни. Бекендът получава новите данни и Entity Framework обработва операцията по актуализация.
* DELETE: Премахва ресурс от сървъра. В сценария, където приложение React трябва да изтрие запис от базата данни, се издава заявка DELETE към ASP.NET Core API, което след това използва Entity Framework за премахване на съответните данни от базата данни.

Разработчиците установяват връзката между фронтенда и бекенда чрез тези заявки, което позволява гладкото протичане на данните и взаимодействието в приложението. Тази структура гарантира, че приложенията на React.js могат ефективно да комуникират с бекенд на ASP.NET Core, използвайки Entity Framework за правилното управление и операции с данни.



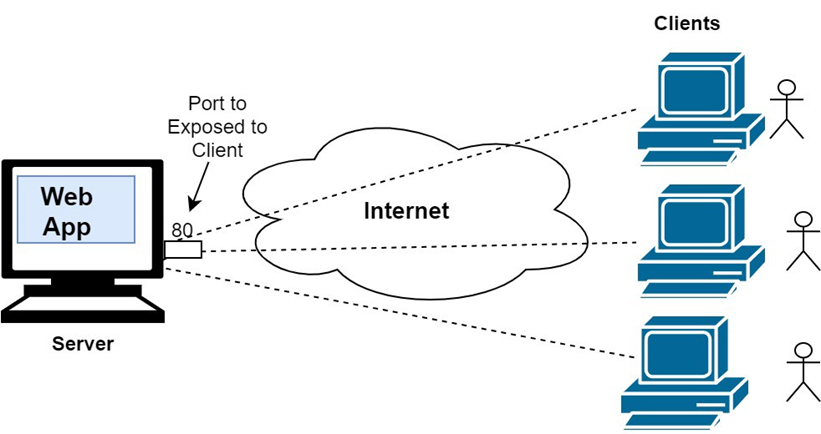
Фигура 9: Пример за service в React.js

## **Проектиране и имплементация**

В тази глава, идва ред на обяснението как е проектирано и имплементирано уеб приложението **“HealthEdge”.** Ще разгледаме логиката в backend-а, data-access layer-a, миграциите, моделите, frontend-а, как се осъществява връзката между клиентската и сървърната част и др.

## **Архитектура на системата**

Архитектурата на проекта е client-server, която изобразена, чрез схема изглежда по следния начин:



Фигура 10: Client-server архитектура

Client-server архитектурата е модел на софтуерен дизайн, който е разделен на два основни компонента: Client (клиент) и server (сървър). Клиентът е софтуер, който изпълнява заявки към сървъра, за да получава информация или, за да извършва операции. Сървърът е софтуер, който отговаря за заявките на клиента и обработката на информация, която е необходима за изпълнението им.

Системата работи като клиентите комуникират със сървъра чрез мрежата, като изпращат заявки за информация или изпълнение на операции. Сървърът получава тези заявки, обработва ги и изпраща отговори (response/s) обратно към клиентите. Това позволява на различни клиенти да работят едновременно с един и същ сървър, като споделят обща информация и ресурси.

Предимствата на този модел включват улеснена съвместна работа и управление на данни, както и по-лесно скалиране на приложенията. В допълнение, този модел предоставя по-голяма сигурност на данните, тъй като те са съхранени централизирано на сървъра и могат да бъдат защитени с подходящи механизми за достъп и сигурност.

## **Имплементация на системата**

В следващите точки и подточки ще бъде показано как е имплементирана система, както в backend-а, така и във frontend-a.

## **Имплементация на Backend системата**

Първа стъпка в имплементацията на backend системата е инициализацията и конфигурацията на базата данни, след това се правят моделите, първите миграции, след това се правят repository-та и controller-и. В репозиториите е логиката, а в контролерите са самите endpoint-и отговорите (response-ите) на сървъра.

## **Инициализация на базата и конфигурация**

След като е направена архитектурата на базата и таблиците на теория, следва инициализацията, която е сравнително проста задача, благодарение на **Entity Framework,** тъй като той генерира голяма част от кода, който е нужен за целта. В следващата фигура е представена първата част от DatabaseContext файлът:

A screen shot of a computer program

Description automatically generated Фигура 11: Пример за първата част от DatabaseContext файлът.

Втората част от DatabaseContext файлът е представена в следващата фигура. Там се намира **OnModelCreating метода:**

A computer screen shot of a program code

Description automatically generated

Фигура 12: Пример за втората част от DatabaseContext файлът.

## **Модели**

Моделите са необходими, за да зададем на Entity Framework скелета, по който да създаде базата данни и таблиците, които ще са необходими. В тези модели съществуват различни анотации, които служат за различни неща, като ограничение на дължината на записа, означаване на Id и т.н.

В „HealthEdge” понастоящем има 13 модела. В следващото изображение са представени всички модели, като част от тях са създадени с цел улеснение на бъдещо развитие.

A screenshot of a computer

Description automatically generated

Фигура 13: Всички модели на HealthEdge към настоящия момент

Конкретен пример за модел от „HealthEdge” за таблица Patients е моделът Patient:

A screen shot of a computer program

Description automatically generated

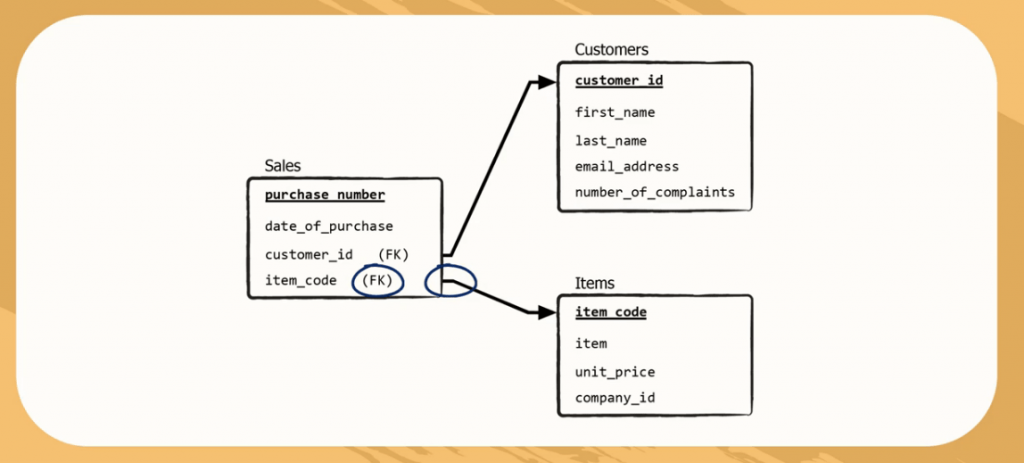
Фигура 14: Модел Patient в „HealthEdge”

Моделът Patient служи за създаване на таблицата Patients. Разликата между двете е името, моделът (entity-то) е в единствено число, докато името на таблицата е в множествено число. Това е така, тъй като в обектно-ориентираното програмиране, моделът обикновено представлява една същност или обект. Например моделът Patient представлява един пациент. Ето защо моделите обикновено се именуват в единствено число. Така става по-ясно, че класът на модела и неговите екземпляри се отнасят до отделни същности. В случая в класа Patient се работи с данните на един пациент. От друга страна, една таблица съдържа множество екземпляри на подобни същности. Следователно именуването на таблица Patients показва, че таблицата съдържа данни за много пациенти, а не само за един. Това наименование в множествено число помага на разработчиците бързо да разберат, че таблицата е колекция от много записи, всеки от които съответства на различен пациент.

В примера с модела на таблицата Patients, можем да разбием модела на няколко основни части.

* **Основни атрибути и свойства**
  + **[Key]: Означава, че свойството Id е основен ключ на таблицата.**
  + **[Required]: Указва, че полето е задължително. Използва се за Id, FirstName, LastName, Email, Gender, BloodType, ContactNumber и Address.**
  + **[DatabaseGenerated(DatabaseGeneratedOption.Identity)]: Показва, че стойността на Id се генерира автоматично от базата данни при добавяне на нов запис.**
* **Свойства за въвеждане на данни**
  + **FirstName, LastName, Email, ContactNumber, Address: Тези свойства съхраняват лична информация и контактни данни за пациента.**
  + **[StringLength(50)], [StringLength(100)], [StringLength(1)], [StringLength(4)], [StringLength(255)]: Задават максималните дължини за съответните низове, управляващи ограниченията на базата данни за различните полета.**
* **Специфични типове и формати**
  + **[Column(TypeName = "Date")]: Указва, че DateOfBirth трябва да се съхранява като тип дата в базата данни, без време.**
  + **Gender: Съхранява пола на пациента, ограничен до един символ.**
  + **BloodType: Съхранява кръвната група на пациента, ограничена до четири символа.**
* **Foreign Keys and Navigation Properties**
  + **UserId и съответното User: Опционални свойства, които свързват пациента с потребителски профил в системата, ако такъв е наличен.**
* **Колекция от данни**
  + **Appointments: Колекция, която съдържа всички медицински назначения/приеми асоциирани с пациента, показва връзката му с различни медицински услуги.**

**Моделът Patient е създаден така, че да предоставя структуриран начин за съхранение и управление на лични и медицински данни за пациентите в рамките на софтуерна система, улеснявайки достъпа до важна информация и поддържайки висока степен на даннова цялостност.**



Фигура 15: Пример за начина на работата на Foreign key

Друг пример, в който може да открием малко по-сложно устройство на таблица можем да открием в Doctor model:

A screen shot of a computer program

Description automatically generated

Фигура 15: Модел Doctor в „HealthEdge”

В примера с модела на таблицата Doctors, Можем да разбием модела на няколко основни части:

* **Атрибути и свойства**
  + **[Key] Този атрибут отбелязва свойството Id като primary key (първичен ключ) на таблицата.**
  + **[Required] Указва, че полето е задължително. Използва се за свойства като Id, FirstName, LastName, ContactNumber и Email.**
  + **[DatabaseGenerated(DatabaseGeneratedOption.Identity)]: Показва, че стойността на Id се генерира от базата данни, когато се вмъква нов запис. Това е типично за полета на основни ключове, които се увеличават автоматично (auto increment).**
  + **[StringLength(50)] или [StringLength(100)]: Задава максималната дължина на символите, позволена в свойства от тип низ, използвани за дефиниране на ограничения в схемата на базата данни.**
  + **Свойства като FirstName, LastName, Username, ContactNumber и Email са пряко разбираеми; те съхраняват лична и контактна информация за доктора.**
* **Foreign Keys and Navigation Properties**
  + **[ForeignKey("FieldName")]: Този атрибут определя кое свойство е чуждестранен ключ във връзка между таблиците. Например, RegionId е чуждестранният ключ, който свързва Doctor с ентити-то Region.**
  + **Navigation properties (Region, Specialization, Insurance)**: Тези свойства представляват връзките за взаимоотношения с други таблиците/ентитети. Те позволяват лесно зареждане на свързаните данни (например, регионът на доктора, специализацията и застраховката).
* **Колекции**
  + **Appointments: Това е колекция, която съдържа списък на всички назначения, асоциирани с доктора. Илюстрира връзка от тип един към много, където един доктор може да има много назначения.**
* **Булеви стойности**
  + **IsPediatrician**: Булево свойство (свойство, което има два типа стойности (true или false), което се използва при търсето на лекар и чрез него се отбелязва дали желаният лекар трябва да бъде педиатър.
  + **Nzok**: Това булево свойство се използва при търсенето на лекар и чрез него се отбелязва дали желаният лекар трябва да работи с НЗОК. (Национална здравноосигурителна каса)
* **Допълнителна логика и характеристики**
  + Този модел конкретно не съдържа методи или допълнителна логика пряко, но има модели, в които се създава допълнителна логика и характеристики, но в конкретния модел - Patient, той предимно дефинира структурата и взаимоотношенията на данните Doctor в базата данни, улесняващи управлението на информацията за докторите ефективно, докато осигурява съответствие на данните и изпълнява бизнес правила чрез ограничения и взаимоотношения.

# Заключение

*Съдържа изводи и предложения за доразвиване на проекта и възможностите за неговото приложение.*

# Информационни източници

*Включва цитираната и използвана в записката на дипломния проект литература. Започва на отделна страница от основния текст. При имената на авторите първо се изписва фамилията. Всички описания в списъка с използваните източници трябва да са подредени по азбучен ред според фамилията на първия автор на всяка публикация*

# Приложения

*Съдържат документация, която не е намерила място в текста поради ограниченията в обема й или за по-добра прегледност подредба. В текста трябва да има препратка към всички приложения*