**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Електронен дневник

Дипломант: Научен ръководител:

*Методи Пандуров Иван Драганов*

СОФИЯ

2 0 2 1

**Електронен дневник**

**УВОД**

С напредването на технологиите, те все повече започват да изместват хартията като основен носител на информация. Вече сме станали свидетели на електронни книги, графични таблети, масово вече комуникацията е чрез eлектронни пощи, чатове и др. Въпрос на време е и масово училищата да започнат да използват и електронни дневници и бележници.

Една от основните причини да се опитам да създам такъв дневник е фактът, че нашето учлище все още използва хартиените му предци. Плюсовете, които би донесъл електронният дневник са: по-удобно внасяне на информацие, по-сигурно запазване на информация, възможност за достъп до тази информация по всяко време и много други.

Изисквания:

- Административната част е десктоп приложение

- Потребителската част е мобилно приложение

- Поддръжка на различни потребители - администратор, ученик, учител

- Имплементиране на OATH2 provider за потребителските акаунти

- Използване на OATH2 provider-а от мобилните приложения

- Поддръжка на ученици, учители, предмети, оценки от текущи контролни и годишни

**ПЪРВА ГЛАВА**

**МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА СОФТУЕРНИ ПРИЛОЖЕНИЯ**

* 1. **Основни принципи, технологии и развойни среди за реализиране на десктоп приложения**

Десктоп приложението е приложение, което работи самостоятелно на даден лаптоп или настолен компютър. То може да се разглежда и като локална версия на услига или уеб приложения, които предоставят опцията за взаимодействие на потребителя с приложението, без наличието на интернет връзка или услига.

Този тип приложения са придобили огромна популярност, тъй като предоставят крос-платформени решения за доста обемисти уеб услуги, както и предимството да не се засегне цялата архтектура при наличието на грешка или проблем.

Десктоп приложенията се инсталират на личния или работния компютър и тези инсталации са необходими на всеки компютър поотделно. Основните недостатъци свързани с тях е, че актуализирането на такива приложения е много трудно и трябва да бъде правено потделно на всеки компютър на който е инсталирано. Най-голямото предимство на настолното приложение се крие във факта, че скоростта и производителността му надвишават уеб приложенията, тъй като е независимо от интернет. Също така те имат и възможност за изпълнението на множество функции едновременно.

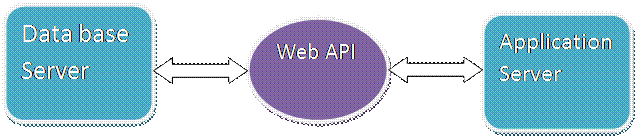
* 1. **Основни принципи, технологии и развойни среди за реализиране на Web API**

В програмирането интерфейсът за приложно програмиране (API) представлява набор от дефиниции на подпрограми, протоколи и инструменти за изграждане на софтуер и приложения.

API е вид интерфейс, който има набор от функции, които ти позволяват достъпа до специфични функции или данни на приложение, операционна система или други услуги.

С други думи, ако искате да взаимодействате с компютър или система за извличане на информация или изпълнение на функция, API ви помага да съобщите какво искате на тази система, за да може тя да разбере и да изпълни заявката.

Web API, както подсказва името, е API, който се намира в онлайн среда и може да бъде достъпен чрез HTTP протоколи. Можем да изградим Web API, използвайки различни технологии като Java, .NET и др.



Фиг.1.1: Визуализация на логиката зад Web API

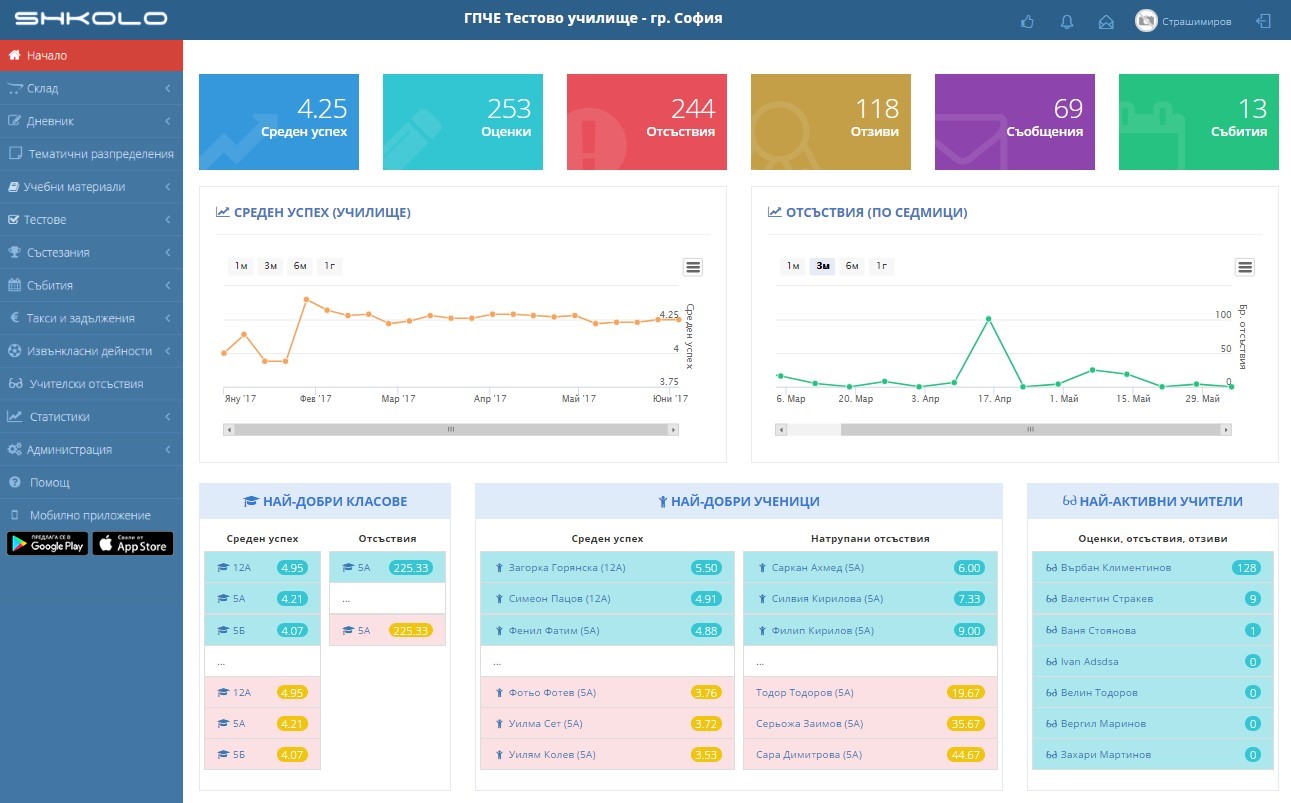
Както е показано на Фиг.1.1 Web API служи като посредник между десктоп/мобилното приложение и базата данни. Това олеснява работата със самата база данни и самите процеси стават по лесно и сигурно.

* 1. **Основни принципи, технологии и развойни среди за реализиране на** **мобилно приложение**

Мобилното приложение е вид приложен софтуер, предназначен да работи на мобилно устройство, например смартфон или таблет. Те често служат за предоставяне на потребителите на услуги, подобни на тези, достъпни на компютри.Мобилните приложения обикновено са малки, отделни софтуерни единици с ограничена функция.

Мобилните приложения са отдалечаване от интегрираните софтуерни системи, които обикновено се срещат на компютрите. Вместо това всяко приложение предоставя ограничена и изолирана функционалност като игра, калкулатор или мобилно сърфиране в мрежата. Въпреки че приложенията може да са избегнали мултитаскинга поради ограничените хардуерни ресурси на ранните мобилни устройства, тяхната специфичност сега е част от тяхната желателност, тъй като позволяват на потребителите да избират ръчно какво могат да правят техните устройства.

* 1. **Съществуващи решения и реализации – Школо(Shkolo)**

Идеята за електронни дневници не е нещо напълно ново. От няколко години насам много от училищата в България имплементираха тази технология във воденето на училищната си документация. Водещият и масово наложил се такъв софтуер е така нареченото Школо(Фиг.1.2).

Фиг.1.2: Потребителският интерфейс на Shkolo

Школо е създадено през 2016 година, като първата версия е пусната през януари 2017 година. Към 2019 г. софтуерната платформа Школо се използва в над 1000 български [училища](https://bg.wikipedia.org/wiki/%D0%A3%D1%87%D0%B8%D0%BB%D0%B8%D1%89%D0%B0" \o "Училища) от близо 697 000 потребители.

Образователният софтуер на Школо има всички реквизити на стандартното хартиено издание – имена на учениците, оценки, отсъствия, забележки, и прочие. Разполага с мобилна версия и е своеобразна електронна мрежа за учители, родители и ученици.

Платформата интегрира множество модули, сред които са електронен дневник, администриране на извънкласни дейности, статистически справки, училищен календар и управление на събития, управление на складови наличности и инвентаризация, модул за разплащане (много популярен сред частните училища), мобилно приложение и други.

Предоставя и набор от образователни ресурси: електронни уроци, карти, тестове, учебни пособия, публикации и др.

**ВТОРА** **ГЛАВА**

**ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА ЕЛЕКТРОННИЯТ ДНЕВНИК**

**2.1 Функционални изисквания към електронният дневник**

- Административната част е десктоп приложение

- Потребителската част е мобилно приложение

- Поддръжка на различни потребители - администратор, родител, учник, учител

- Имплементиране на OATH2 provider за потребителските акаунти

- Използване на OATH2 provider-а от мобилните приложения

- Поддръжка на ученици, учители, оценки от текущ контролни и годишни

Цялото приложение е разделено на три части: десктоп приложение, web API и мобилно приложение. Чрез десктоп приложението ще се въвеждат данните на учениците, учителите, както и ще се създават акаунти. Тези данни ще бъдат изпращани към web API. От там мобилното приложевние ще получава нужните му данни според зависи акаунта, който се е логнал и така даденият потребител ще има достъп до информацията, която го касае.

**2.2 Съображения за избор на програмни средства и развойната среда за десктоп приложението**

За разработката на десктоп приложението съм изплозвал C# под структората на WPF. Windows presentation foundation (WPF) се изпозлва за създаването на десктоп клиентски приложения. Платформата за разработка на WPF поддържа широк набор от функции за разработка на приложения, включително модел на приложение, ресурси, контроли, графики, оформление, обвързване на данни, документи и сигурност. Чрез WPF се използва и XAML. XAML е маркъп език за приложения, с помощта на който се създават персонализирани контроли, графики, 3D изображения и анимации, които не са налични в традиционните HTML имплементации.

Десктоп приложението е писано изцяло на Visual Studio. Причината е че Visual Studio, C# и WPF са разработки на Microsoft и си взаимодействат по доста удобен начин, предоставяйки голям набор от готова документация.

**2.3 Съображения за избор на програмни средства и развойната среда за Web API**

Web API е изграден върху APS.NET Core. ASP.NET Core прердставлява междуплатформена, високопроизводителна структора за изграждане на модерни приложения, свързани с облак или интернет. С ASP.NET Core може да се изграждат уеб приложения и услуги, приложения на Internet of Things (IoT) и мобилни бекендове. Едно от най-големите предимства на ASP.NET Core e, че множество версии могат да съществуват заедно на един и същи сървър. Това осначава, че едно приложение може да приеме най-новата версия, докато други, по-стари приложения, могат да продължават да работят на версията върху която са създадени.

Приложенията на ASP.NET могат да се разработват и изпълняват на Windows, Linux, macOS и Docker. Продуктите свързани с Visual Studio имат инструменти за исграждане на .NET приложения на всяка операционна система. Именно за това развойната среда за уеб API е Visual Studio.

**2.4 Съображения за избор на програмни средства и развойната среда за мобилното приложение**

Мобилното приложение е писано на Xamarin.Forms. Xamarin.Forms е атрибут на Xamarin, средата за мобилно програмиране, която разширява платформата за разработчици .NET с инструменти и библиотеки за изграждане на мобилни приложения.

Xamarin.Forms е междуплатформена рамка с отворен код от Microsoft за изграждане на приложения за iOS, Android и Windows с .NET от една споделена кодова база.

С Xamarin приложението може да бъде написано изцяло на C#, от бакедн до достъпа до API. Xamarin разширява .Net и за това може да използват големичт борй набор от пакети и библиотеки предоставени от платформата, което води до улесняване на цялостното писане на приложението.

Както споменах, Xamarin е част от платрофмата .NET.За това развойната среда, която съм избрал, отново е Visual studio.

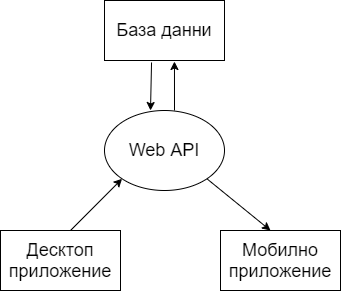
**ТРЕТА** **ГЛАВА**

**ПРОГРАМНА РЕАЛИЗАЦИЯ НА ЕЛЕКТРОНЕН ДНЕВНИК**

**3.1 Подбор на езикът за програмиране**

Цялото приложение и писано на езикът за програмиране C#. C# е главно обектно ореантиран език (ООП) за работа в мрежа уеб разработка. C# e разработен от [Microsoft](https://bg.wikipedia.org/wiki/Microsoft), като част от софтуерната платформа [.NET](https://bg.wikipedia.org/wiki/.NET). Стремежът още при създаването на C# езика е бил да се създаде един прост, модерен, обектно-ориентиран език с общо предназначение. Основа за C# са [C++](https://bg.wikipedia.org/wiki/C_Plus_Plus), Java и донякъде езици като [Delphi](https://bg.wikipedia.org/wiki/Delphi), [VB.NET](https://bg.wikipedia.org/w/index.php?title=VB.NET&action=edit&redlink=1) и [C](https://bg.wikipedia.org/wiki/C_(%D0%B5%D0%B7%D0%B8%D0%BA_%D0%B7%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D0%BD%D0%B5)). Той е проектиран да балансира мощност ([C++](https://bg.wikipedia.org/wiki/C_Plus_Plus)) с възможност за бързо разработване ([Visual Basic](https://bg.wikipedia.org/wiki/Visual_Basic) и Java). Езикът представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. Програмите на C# представляват един или няколко файла с разширение .cs., в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на C# (csc) до изпълним код и в резултат се получават асемблита – файлове със същото име, но с различно разширение.

Приложението главно се състои от три части: десктоп, мобилна и онлайн.



Фиг.3.1:Изображение на логиката зад проекта

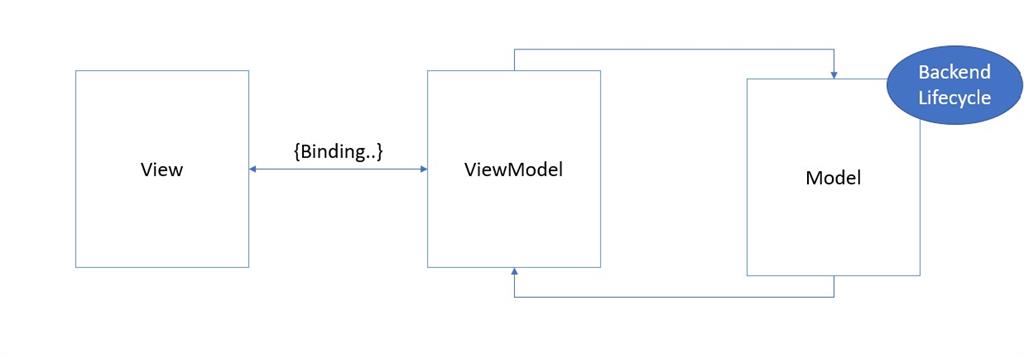
Логиката зад самото приложение е следната:

-след въвеждането на данните в десктоп приложението и натискане на бутон за запазване, те се избращат към Web API, който от своя страна си комуникира със базата данни.

-при отваряне на мобилното приложение, след влизане в съответния акаунт(създаден отново през десктоп приложението), потребителят има достъп до данните, които се отнасят до него.

**3.2 Програмна реализация на десктоп приложението**

Приложението е писано на C# използвайки архитектурата на Microsoft – WPF. За по-добра четимост и удобство е имплементриан и моделът за писане на код MVVM(Model-View-ViewModel).

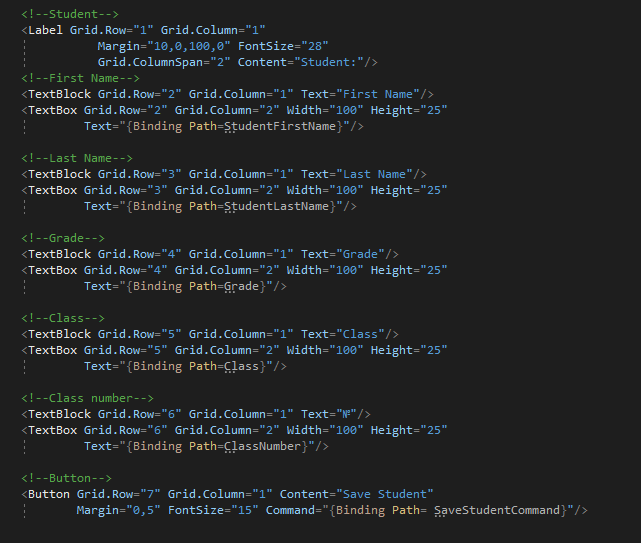
Фиг.3.2: Нагледно представяне на логиката на MVVM

Model (Моделът) представлява модела на домейна, който също се счита за бизнес логика, логика за достъп до данни или можем абстрактно да го дефинираме като бекенд на приложението. В него се съхраняват данни за приложенията.

View (Изглед) представлява потребителския интерфейс на приложението. Потребителят взаимодейства с него и това е частта от приложението, която е достъпна до него.

ViewModel представлява логиката, стояща зад View-то. ViewModel се нарича още презентационна логика. View и ViewModel комуникират помежду си. Заявката от ViewModel се препраща към Model / бизнес логически слой / слой за достъп до данни. Тя позволява споделяне на изчислени или получени данни във View-то.

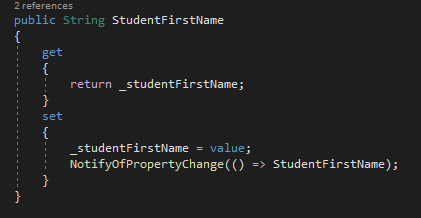
В нашият случай в папката Model има два класа: Student и Teacher. Те съдържат само наименованията на данните характерни съответно за учителите и учениците.

В папката View се намира файлът ShellView. Това е дизайнът на потребителският интерфейс. Също така в него се осъществяват и връзките чрез които се изпращат нужните данни. Това се осъществява посредством Binding функцията, която праща новата стойност към дадената променлива.

Фиг.3.3: Нагледно преставяане за използването на *Binding*

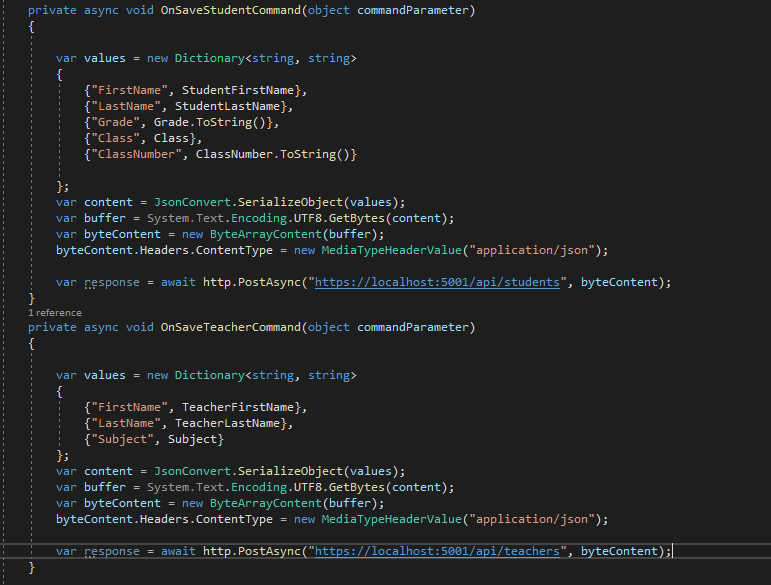
Във ViewModel папката и по-точно в класът ShellViewModel се намира логиката зад цялото приложение. Там са създадени всички нужни пропъртита и команди нужни на приложението за да се осъществят връзките между View, ViewModel и Model.

В set метода на всяко пропърти се извиква и функцията *NotifyOfPropertyChange().* Благодарение на нея(както и на *Binding* функцията във View частта) данните могат въведени от потребителя се запазват на съответните променливи. Ролята на самат функция е при наличие на промяна (въвеждане на нова стойност от потребителят) във View частта съответното пропърти да бъде оведомено и да приеме новата стойност. Цялата връзка няма да работи ако не присъстава *Binding* функцията във View частта и *NotifyOfPropertyChange()* функцията във ViewModel частта.



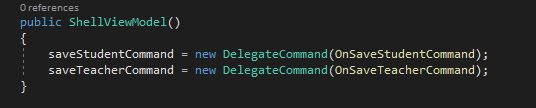
Фиг.3.4: Пропъртито за името на чуеникът с функцията *NotifyOfPropertyChange()*

Други важни функции във ViewModel класът са *OnSaveStudentCommand()* и *OnSaveTeacherCommand().* Посредством тях при натискане на Save бутоните от потребителя във визуалната част(виж Фиг.3.3) въведените данни се изпращат към Web API. И двете команди работят по еднакъв начин като първо въведените данни се записват в Dictionary. След това Dictionary-то се превръща в Json формат посредством JsonConverter, добавят му се нужните хедъри и се изпраща към url-ът на API.



Фиг3.5 Командите *OnSaveStudentCommand()* и *OnSaveTeacherCommand()*

В проекта съществъва и още една папка Commands. В нея е класът DelegateCommand. В този файл се наследява интерфейсът ICommand, който ни помага за по-лесна и проста работа с нужните ни команди. Във файлът ShellViewModel има две пропъртита от тип DelegateCommand: *SaveStudentCommand()* и *SaveTeacherCommand()*. Те се създават във констукторът на класа като вземат логиката на описана в *OnSaveStudentCommand()* и *OnSaveTeacherCommand().* След това посредством *Binding* функцията се „закачат“ за съответните бутони.

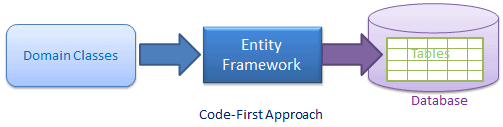


Фиг.3.6: Инициализирането на двете променливи във конструкторът на класа

**3.3 Програмна реализация на Web API**

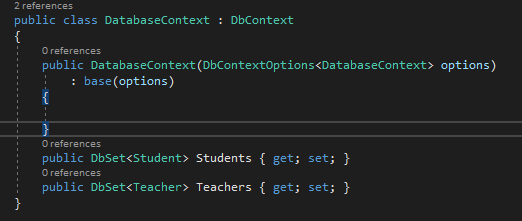
Приложният интерфейс е писан на C#, като е изполвана главно архитектурата на ASP.NET Core. Зад него седи CodeFirst база данни. Към нея се изпращат получените данни от десктоп приложението и съответно се изпращат нужните данни към мобилното.

CodeFirst за пръв път е въведен с излизането на Entity Framework 4.1. От своя странна Entity Framework e съвремменен картограф на бази данни предоставен от .NET.

При CodeFirst базите с данни фокусът пада върху домейна на приложението и се създават класове за обекта на самият домейн, вместо първо базата с данни да се проектира и след това да се създават класовете, които съответстват на дизайна на базата данни. (Фиг.3.7)

Фиг.3.7: Логиката зад CodeFirts база данни

Web API се състои от няколко папки. Първата папка е Context папката. В нея е класът DatabaseContext. DatabaseContext е основният клас, който координира функционалността на Entity Framework за модел на данни. DatabaseContex също така наследява и класът DbContext. Класът DbContext представлява комбинация от шаблони за Unit Of Work и Repository, така че може да се използва за заявки от база данни и да групира промените, които след това ще бъдат записани обратно в хранилището като единица. В DatabseContext също така се декларират двете таблици: Teachers и Students.



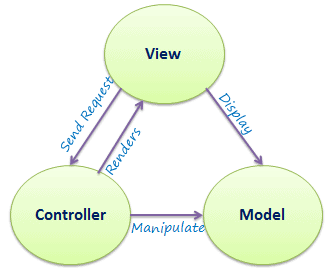
Фиг.3.8: Класът DatabaseConstext

В папката Controllers се намират два класа : StudentsController и TeahcersController, които съответно са контролерите на таблиците.

Фиг.3.9: Класът StudentsController(подобно изглежда и класът TeachersController)

StudentsController, както и TeachersController наследяват класът ControllerBase. ControllerBase представлява базов клас за MVC контролер без поддръжка на визувалнба част.

MVC е архитектура, наподобяваща MVVM, която не съкратено означава Model-View-Controller. MVC разделя приложението на три компонента - Model, View и Controller.



Фиг.3.10: Нагледно преставяне на MVC

Model представлява формата на данните. Клас в C # се използва за описание на модел. Обектните модели съхраняват данни, извлечени от базата данни. При нас в папката Models са дефинирани атрибутите на самите таблици за учениците и учителите в два отделни класа: съответно Students и Teachers.

View в MVC е потребителски интерфейс. Чрез него се предоставя достъп на потребителя до данните, както и възможността за тяхното променяне. В момента липсва визуална част, защото тя не ни е нужна. Използваме Web API само за съхраняване на информацията достъп до нея и променяне получава от мобилното и десктоп приложението.

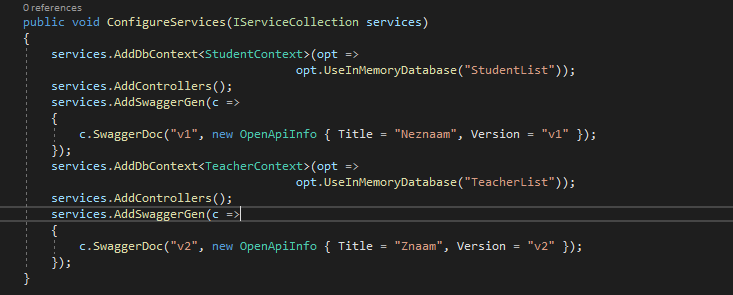
Контролерът обработва заявката на потребителя. В контролер класовете се опсва логиката и поведението на самите таблици спрямо данните. В нашият случай са описани две заяваки – Post и Get.

Посредством Post заяваката данните се избращат от десктоп приложението към съответната таблица.

С помощтта на Get заявката необходимите данни се изпращат към мобилното приложение.

В тях са описани съответните Post и Get заявки. Благодарение на тези контролери базата данни знае при извикването на съответната заявка, кои данни да приеме и съответно кои да изпрати.

С класът Startup се конфигурират онслугите и заявките на/към приложението. ASP.NET Core приложенията използват клас Startup, който по конвенция се нарича Startup. Startup класът съдържа ConfigureServices метод отговярящ за конфигурирането на услигите на приложението. Услугата е компонент за многократна употреба, който предоставя функционалност на приложението. Услугите се регистрират в ConfigureServices и се консумират в приложението чрез инжектиране на зависимости (DI) или ApplicationServices. Startup класът още съдържа и метода Configure чрез който се създават заяваките и наборът от данните.



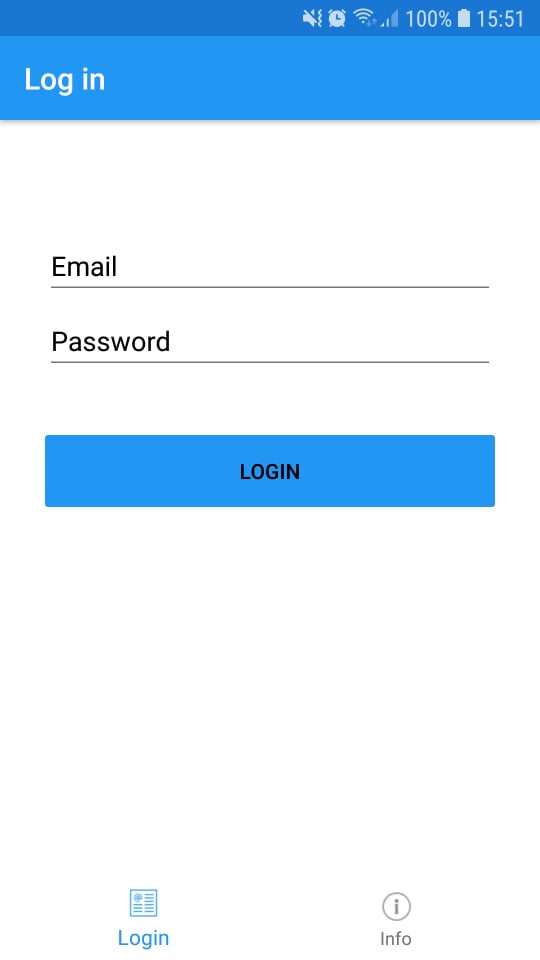
Фиг.3.11: ConfigureServices методът в класът Startup

В нашето приложение в методът ConfigureServices за регистрирани две услуги съответно за таблицата на учениците и учителите.

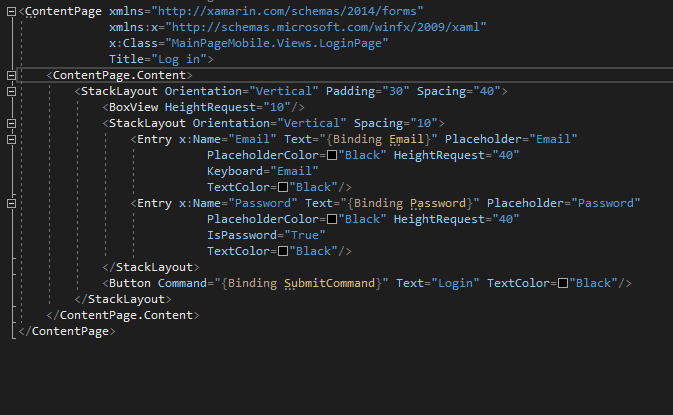
**3.4 Програмна реализация на мобилно приложение**

Мобилното приложение е писано отново на C# като е имплементирана архитектурата на Microsoft – Xamarin. Xamarin се използва за разработване на cross-platform мобилни приложения. Един от големите му плюсове е, че той позволява голяма част от кода да бъде споделен между различните платформи. Xamarin използва C# и собствени библиотеки, базирани на .Net за разработване на приложения на различни платформи. Xamarin също така може да използва най-новият достъп до API и по този начин да използва основните възможности на платформата в приложенията на Xamarin като ARKit за IOS или Android Multi-Window.

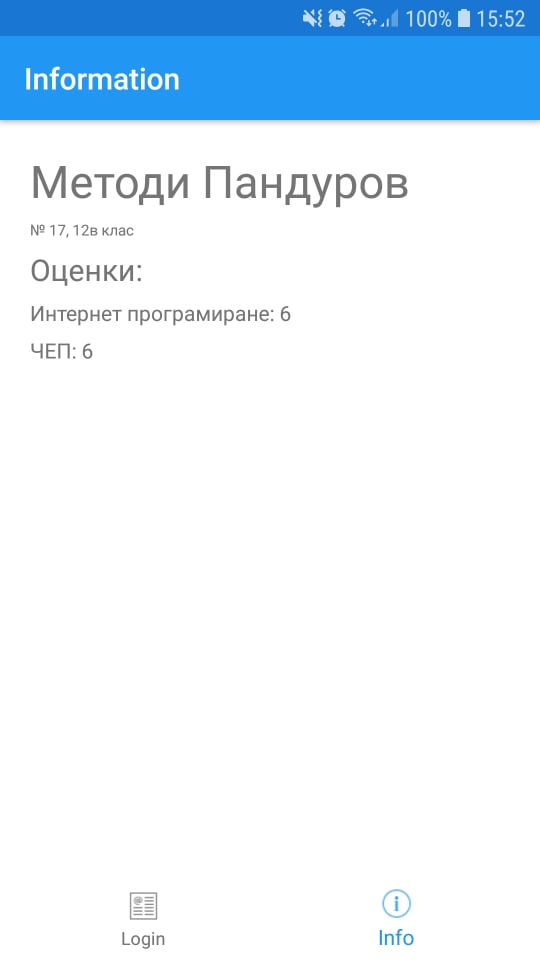
Мобилното приложение се състои от две „страници“. Едната е най-обикновен логин,



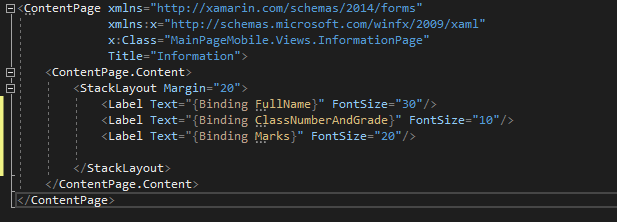
Фиг.3.12: Логин страницата на мибилното приложение



Фиг3.13: Кодът зад визуалната част на логина

а другата е страницата, на която ще бъдат изписани съответните данни след като ученикът/учителят влезе в акаунта си.

Фиг.3.14: Информационната страница на мобилното приложение за ученик



Фиг.3.15: Кодът зад визуалнтата част на ученика

Ако никой не е влязъл в акаунта си, то страницата седи празна.

Преди да има достъп ученикът или учителят до данните си, той трябва първо да е влязъл в своят акаунт.



Фиг3.14: Логиката зад логин(В примерът е hardcode-нат акаунт)

Мобилното приложение отново е писано по моделът на MVVM. Разделено е на три основни папки: Models, Views и ViewModels.

В папката Models се намират моделите съответно на учениците и учителите, съдържащи само пропъртитата им.

В папката Views са визуалниата част на двете страници – Login страницата и информативната страница.

Папката ViewModels съдържа логиката зад цялото приложение като например логиката зад login-a, разпознаването на валиден акаунт и т.н.

**ЧЕТВЪРТА** **ГЛАВА**

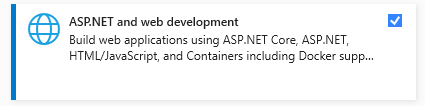
РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ

**4.1 Подготовка на развойната среда**

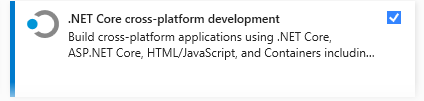
Първата стъпка за стартиране на проекта е инсталирането на Visual Studio 2019 (Community version) от официалният сайт.

Link: <https://visualstudio.microsoft.com/downloads/>.

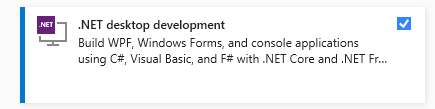
След това е необходимо да се добавят следните разширения:



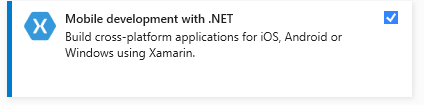
Фиг.4.1: Изображение на разширението за ASP.Net and web development

****

Фиг.4.2: Изображение на разширението за .NET Core cross-platform development



Фиг.4.3: Изображение на разширеноето за .NET desktop development

****

Фиг.4.4: Изображение на разширението за Xamarin

Първите три разширения за свързани главно с .NET платформата. Те са необходими за да може Web API да бъде създаден и да работи по правилен начин.

Последното разширение е за така нареченият Xamarin, чрез който е писано мобилното приложение.

**4.2 Стартиране на Web API**

След инсталирането и добавянето на нужните разширения се отваря папката под името WebApp



Фиг.4.5:Изображение на папката

В папката има два файла. Отваря се файлът WebApp.sln .



Фиг.4.6: Файловете в папката

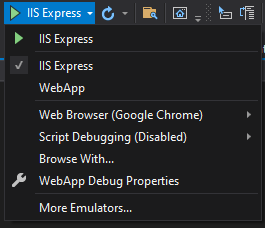
Трябва да се отвори прозорец на Visual Studio 2019, където се вижда самият код на пролжениетозаедно със сапътстващите го папки и файлове.

Следщата стъпка е стартиранетo. В горната част на екрана трябва д аима една зелена стрелка, до която трябва да е изписано и името на файлът.



Фиг.4.7: Бутона за стартиране на програмата

Ако не е зиписано името на файлът, а ISS Express трябва да се натисне малкта стрелка в дянсо и да се избере оцията, ня койято е и името на файлът.



Фиг.4.8: Разширените опции на бутона за стартиране на програмата

След нейното натискане програмата се пуска и така имаме вече работещ API.

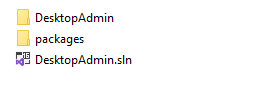
**4.3 Стартиране на десктоп приложението.**

За да пуснем десктоп приложението трябва да отворим папката с име DesktopAdmin.



Фиг.4.9: Изображение на папката

След като я отворим ще намерим три файла. Отваряме файлът под името DesktopAdmin.sln .



Фиг.4.10: Файловете в папката

Отново трябва да се отвори прозорец на Visual Studio 2019. В горната част на екрана пак трябва да има зелена стрелка, на която този път обаче пише Start.

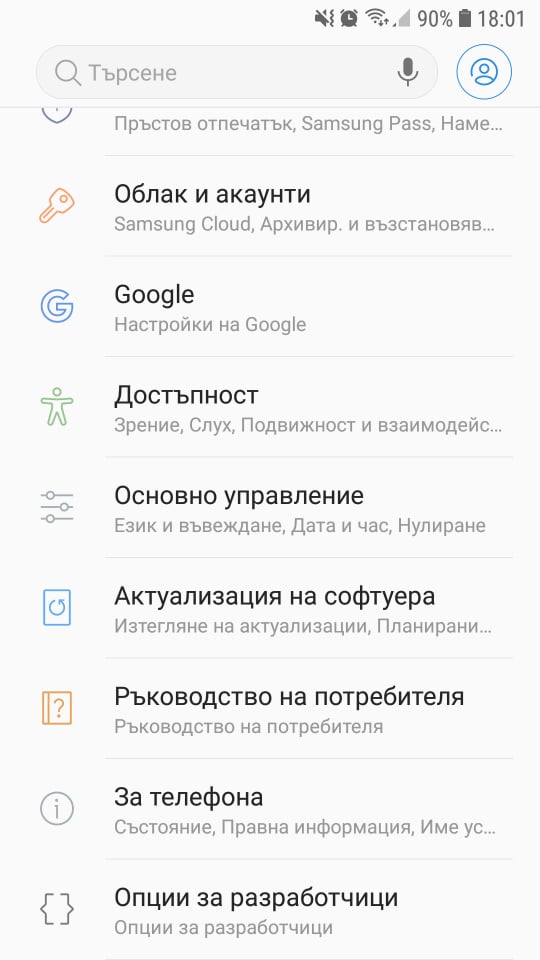


Фиг.4.11: Бутона за стартиране на програмата

След нейнот натискането програмата започва да работи и трябва да се отвори нов прозорец със самото приложение.

**4.4 Стартиране на мобилното приложение**

Преди да пуснем мобилното приложение е препоръчително преди това да сме свързали с компютъра мобилно устройсто(за предпочитане е да е Android). На самият телефон трябва да е пусната опцията за разработчик.



Фиг.4.12: Главното меню на настройките на мобилно устойство с операционна система Android

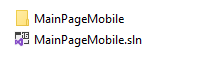
Ако тази опция не е пусната трябва да се отоври от настройките менюто на име „За телефона“. От там трябва да се намери опцията „Сериен номер“(или нещо подобно). След това трябва да се натисне около 7 пъти. Ще се покаже прозорец искащ вашата парола и след въвеждането и оцията за разработчици ще се покаже в главното меню на настройките на телефона.

След като вече сме свързали мобилно устройство с компютъра трябв да отворим папката с име MainPageMobile.



Фиг.4.13: Изображение на папката

В нея ще намрим два файла файла. Отваряме този под името MainPageMbile.sln .



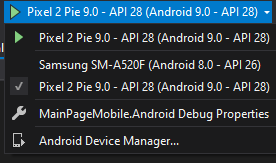
Фиг.4.14: Файловете в папката

След отварянето на прозореца на Visual Studio 2019 отново в горната част на екрана трябва да намерим зелена стрелка. До нея трябва да е изписано името или модела на телефона, свързан с компютъра.



Фиг.4.15: Бутона за стартиране на програмата

Ако е изписан друг модел телефон трябва да натиснем малката сива стрелка до него. Така ще се покажат всички възможни опции за избиране на телефон. След като намерим моделът/името на свързаният телефон го селектираме и натискаме зелената стрелка.



Фиг.4.16: Разширените опции на бутона за стартиране на програмата

След натискането на стрелката приложението ще се стартира и ще се отвори на мобилното устройство, свързано с компютъра.

След изпълнение на всички стъпки и трите приложения работещи едновременно стигаме до цялостният вид на дипломната работа. Сега вече можем да впизваме данните на ученици/учители и да ги четем от мобилното приложение.

ЗАКЛЮЧЕНИЕ

Вариантът на електронен дневник до който сме достигнали е нещо напълно примитивно. В момента единствената му задача е да предоставя на ученика възможността да вижда оценките си по всяко време стига да има достъп до мобилно устройство. Ако се продължи развитието на идеята, е напълно възможно един ден този електронен дневник да се разрастне в нещо много по голямо и да има възможността цялата учлищна администрация да бъде поддържана от него, в електронен, а не книжовен вариант. Лично аз вярвам, че това е пътят към бъдещето и че вече трябва да се налага по-масово използване на подобни технологии, еквивалентни на техните хартиени заместители, за съхраняване на данни. Причините за това са много но едни от най важните са по-голямата сигурнот, удобството и не на поседно място вкологичният фактор.

За да бъде разгърнат пълният потенциал на самата идея в бъдеще целта е да се добавят следните опции:

- създаване и показване на седмичната програма;

-внасяне на отсъствия и закъснения;

-внасяне на забележки;

-поддръжка на родителски акаунти;

-директна комуникация с родителите;

ИЗПОЛЗВАНА ЛИТЕРАТУРА

**Първа глава:**

# What is Desktop Software? - <https://www.educba.com/what-is-desktop-software/>

# What is Web API? - <https://www.tutorialsteacher.com/webapi/what-is-web-api>

# What is a REST API? - <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

## What Does Mobile Application (Mobile App) Mean? - <https://www.techopedia.com/definition/2953/mobile-application-mobile-app>

**Втора глава:**

# C# (C Sharp) - https://www.techopedia.com/definition/26272/c-sharp

# Windows Presentation Foundation (WPF) - <https://whatis.techtarget.com/definition/Windows-Presentation-Foundation-WPF>

What is ASP.NET Core? - <https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core>

# Introduction to ASP.NET Core - <https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0>

What is Xamarin? - https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin

**Трета глава:**

# MVVM Architecture - <https://www.c-sharpcorner.com/article/mvvm-architecture/>

# What is Code-First? - <https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx>

Web API - <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio>

What Is Xamarin? - <https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/>

DbContext Class - <https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?view=entity-framework-6.2.0>

ControllerBase Class - <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controllerbase?view=aspnetcore-5.0>

ASP.NET MVC Architecture - <https://www.tutorialsteacher.com/mvc/mvc-architecture>

App startup in ASP.NET Core - <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-5.0>

СЪДЪРЖАНИЕ

**УВОД** 3

**ПЪРВА ГЛАВА**

**МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА СОФТУЕРНИ ПРИЛОЖЕНИЯ**…………………………………………………………………………………………4

**1.1** **Основни принципи, технологии и развойни среди за реализиране на десктоп приложения** 4

**1.2 Основни принципи, технологии и развойни среди за реализиране на Web API** 5

**1.3** **Основни принципи, технологии и развойни среди за реализиране на** **мобилно приложение** 6

**1.4** **Съществуващи решения и реализации – Школо(Shkolo)** 7

**ВТОРА** **ГЛАВА**

**ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА ЕЛЕКТРОННИЯТ ДНЕВНИК** 9

**2.1 Функционални изисквания към електронният дневник** 9

**2.2 Съображения за избор на програмни средства и развойната среда за десктоп приложението** 10

**2.3 Съображения за избор на програмни средства и развойната среда за Web API** 10

**2.4 Съображения за избор на програмни средства и развойната среда за мобилното приложение** 11

**ТРЕТА** **ГЛАВА**

**ПРОГРАМНА РЕАЛИЗАЦИЯ НА ЕЛЕКТРОНЕН ДНЕВНИК** 12

**3.1 Подбор на език на програмиране**...........................................................13

**3.2 Програмна реализация на десктоп приложението** 14

**3.3 Програмна реализация на Web API** 19

**3.4 Програмна реализация на мобилно приложение** 24

**ЧЕТВЪРТА** **ГЛАВА**

**РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ** 28

**4.1 Подготовка на развойната среда** 29

**4.2 Стартиране на Web API** 30

**4.3 Стартиране на десктоп приложението.** 32

**4.4 Стартиране на мобилното приложение** 33

**ЗАКЛЮЧЕНИЕ** 35

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**.........................................................................................37

**СЪДЪРЖАНИЕ** 39