LOGO_tu_original **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ, ФИЛИАЛ ПЛОВДИВ**

**ФАКУЛТЕТ ПО ЕЛЕКТРОНИКА И АВТОМАТИКА**

**ДИПЛОМНА РАБОТА**

**НА**

Борис Василев Тумбев Фак. **№** 367157

***Специалност:*** Компютърни системи и технологии

***Образователно-квалификационна степен:*** бакалавър

***Тема:***

**Научен ръководител:**

доц. д-р

Пловдив 2020

Задание

Съдържание………………………………………………………………………………………………2

Увод………………………………………………………………………………………………………...4

Глава I. Обзор - състояние на проблема по литературни данни;……………………………….5

1.Анализ на темата. Цели и задачи…………………………………………………………………..5

1.1Анализ на темата…………………………………………………………………………….5

1.2. Цели и задачи……………………………………………………………………………..10

2. Използвани технологии…………………………………………………………………………….10

2.1 Microsoft Visual Studio……………………………………………………………………..10

2.2 Xamarin за Visual Studio…………………………………………………………………..11

2.3 .Net…………………………………………………………………………………………...12

2.4 C#..............................................................................................................................13

2.5 XAML…………………………………………………………………………………………15

2.6 MVVM………………………………………………………………………………………..17

3.Съществуващи решения……………………………………………………………………………19

3.1 Flightradar24……………………………………………………………………………….19

3.2. Flightradar (Live)…………………………………………………………………………23

3.3. The Flight Tracker………………………………………………………………………….24

Глава II. Теоретично решение на поставената задача;………………………………………….27

1. Цялостна архитектура……………………………………………………………………27
2. Архитектура на мобилното приложение………………………………………………29
3. Данни предоставени от информационната система……………………….30

Глава 3 – Описание на софтуерната част;………………………………………………32

1. Контроли…………………………………………………………………………..32
2. Модел на данните и View-та……………………………………………………32

Глава 4 – Функционално тестване;………………………………………………………..37

Глава 5 – Приложимост на дипломната работа;………………………………………41

Глава 6 – Икономическа оценка на резултатите и техническа ефективност;………41

1.Заключение…………………………………………………………………………41

2.Извод………………………………………………………………………………42

Източници………………………………………………………………………………………43

Приложение……………………………………………………………………………………44

**Увод**

Мобилната мрежа е изградена от милиони по-малки мрежи, с различни функции и топология, свързани помежду си чрез сложна система от мрежов хардуер и софтуер. Тя служи за отдалечен достъп до голямо разнообразие от информационни ресурси, услуги и приложения както и за отдалечена телефонна връзка между хората в цял свят. Тя създава едно голямо общество, давайки му възможност за лесна и бърза комуникация.

Тъй като по-голямата част от хората в цял свят вече използват активно мобилната мрежа и мобилните телефони, приложенията за тях стават все по-популярни, широко разпространени и използвани. Мобилните технологии се развиват с изключително бързи темпове, предлагащи по-добри възможности, характеристики и производителност. Това води и до бързото развитие на приложенията за тях, като те стават все по-лесни и интуитивни за използване, с реални графични ефекти и сцени. Развиват се мобилни приложения от всякакво естество и за всякакви нужди.

Затова предметът на текущата дипломна работа е именно мобилно приложение , което е насочено към следенето на дадени полети в реално време. Приложението е предназначено за всеки един потребител използващ мобилно устройство с Android операционна система и желаещ да придобие знания за неговия или друг полет в реално време.

В днешно време мобилните телефони са неделима част от нашето ежедневие .Това води до нуждата от разработване на мобилни приложения от всякакво естество.

Мобилните приложения тип Flight tracker (Проследяване на полети) са от приложения който се използват често когато пътуваме. Съществуват доста приложения от този вид ,които ще разгледаме по-надолу.

**Глава I**

**Обзор - състояние на проблема по литературни данни;**

1.Анализ на темата. Цели и задачи.

В днешно време мобилните телефони са неделима част от нашето ежедневие .Това води до нуждата от разработване на мобилни приложения от всякакво естество.

Съществуват мобилни приложения от всякакви жанрове – бизнес, игри, образование, финанси, спорт, социални, производителност, музика и аудио, пътешествия и местно съдържание и много други.

Мобилните приложения тип "пътешествия и местно съдържание" са подходящи във всяко едно време – докато пътуваме в метрото или автобуса, почиваме си от работа, докато сме на плажа или в планината.

Предметът на текущата дипломна работа е именно мобилно приложение , което е насочено към следенето на дадени полети в реално време. Приложението е предназначено за всеки един потребител използващ мобилно устройство с Android операционна система и желаещ да придобие знания за неговия или друг полет в реално време.

**1.1Анализ на темата.**

Основната цел е разработването на софтуерно решение за събирането обработването на данни за следенето на самолети по целия свят в реално време. Решението се изразява в това да се създаде система която съчетава едновременно събирането на данни за полетите и предоставянето на разработчиците, които да превърнат тези данни в мобилни или уеб приложения за следене на полетите в реално време.

* Системата която прави всичко това комбинира данни от няколко източника на данни, включително ADS-B, MLAT и радарни данни. Данните ADS-B, MLAT и радарите се обобщават заедно с данните за разписанието и състоянието на полета от авиокомпаниите и летищата, за да се създаде уникален полета за проследяване на полета на разработените приложения.
* **ADS-B**

ADS – B е технология за наблюдение, при която въздухоплавателното средство определя позицията си чрез сателитна навигация и периодично я излъчва, което позволява да се проследява. Информацията може да бъде получена от наземните станции за контрол на въздушното движение като заместител на вторичен радар за наблюдение , тъй като от земята не е необходим сигнал за разпит. Тя може да бъде получена и от друго въздухоплавателно средство, за да осигури ситуационна информираност и да позволи саморазделяне .

ADS – B е „автоматичен“, тъй като не изисква пилотен или външен вход. Той е "зависим", тъй като зависи от данните от навигационната система на самолета. [1]

ADS-B е основната технология, която всички приложения за следене на полети използват за получаване на полетна информация, се нарича автоматично зависимо излъчване на наблюдение ( ADS-B ). ADS-B е сравнително нова технология в процес на разработка, което означава, че днес тя рядко се използва от Air Traffic Control (ATC). Приблизително 70% от всички търговски пътнически самолети (80% в Европа, 60% в САЩ) са оборудвани с транспондер ADS-B. За авиацията с общо предназначение този брой вероятно е под 20%. Процентът на самолетите, оборудвани с приемници ADS-B, непрекъснато нараства, тъй като те ще станат задължителни за повечето самолети по света до 2020 г. ADS-B ще замени първичния радар като основен метод за наблюдение, използван от ATC.

Системата има мрежа от повече от 20 000 ADS-B приемници по целия свят, които получават полетна информация от самолети с ADS-B транспондери и изпращат тази информация до нашите сървъри. Поради използваната висока честота (1090 MHz) покритието от всеки приемник е ограничено до около 250-450 км (150-250 мили) във всички посоки в зависимост от местоположението. Колкото по-далече от приемника лети самолета, толкова по-високо трябва да лети, за да бъде покрит от приемника. Ограничението на разстоянието прави много трудно да се обхване ADS-B над океаните.

Сателитно проследяване на полети е стремеж към глобално покритие на ADS-B. Сателитите, оборудвани с приемници ADS-B, събират данни от самолети извън обхвата на нашата наземна ADS-B мрежа и ги изпращат до мрежата на системата. Данните за ADS-B на сателитна основа идват от множество доставчици. Тъй като броят на сателитите, предоставящи данни и тяхното местоположение са динамични, сателитното покритие варира. Като цяло сателитният ADS-B увеличава покритието на полетите над океана, където не е възможно приемане на наземно ниво. Само самолети, оборудвани с транспондер ADS-B, могат да бъдат проследявани чрез сателит.

Когато въздухоплавателното средство излита извън обхвата, системата продължава да изчислява положението на самолета до 2 часа, ако дестинацията на полета е известна. За въздухоплавателни средства без известна дестинация позицията се оценява до 10 минути. Позицията се изчислява въз основа на много различни параметри и в повечето случаи е доста точна, но при дълги полети позицията в най-лошите случаи може да бъде до около 100 км (55 мили).

От съображения за сигурност и поверителност информацията за някои самолети е ограничена или блокирана. Това включва повечето военни самолети и някои високопрофилни самолети.

Покритието и видимостта на въздухоплавателното средство зависи от много параметри, включително тип самолет, тип транспондер на самолета, надморска височина и терен, така че покритието може да бъде различно за различните въздухоплавателни средства, дори и на едно и също място. Ако самолет, който търсите, не се вижда, той или няма съвместим транспондер, или е извън покритие.

* **FLARM**

FLARM е електронна система, използвана за избирателно сигнализиране на пилотите за потенциални сблъсъци между въздухоплавателното средство. Официално не е внедряване на ADS-B , тъй като е оптимизиран за специфичните нужди на леките самолети, а не за комуникация на далечни разстояния или взаимодействие с ATC. FLARM е голям куфар на "полет" и "аларма". Инсталирането на всички устройства FLARM е одобрено като стандартна промяна на Европейската агенция за авиационна безопасност [1] , а PowerFLARM Core специално е одобрена като незначителна промяна от EASA [2] . В допълнение към стандартната промяна, малката промяна одобрява PowerFLARM Core да се използва по време на IFR и през нощта. [3]

Самата технология ADS-B се обяснява най-добре с изображението.



•Самолетът получава местоположението си от източник на GPS навигация (спътник)

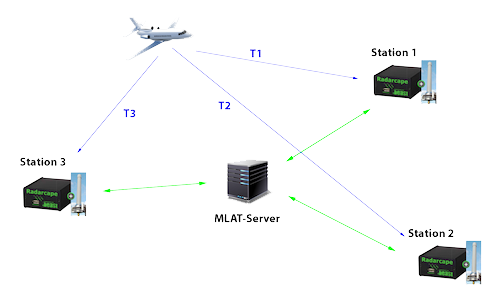
•Транспондерът ADS-B на самолета предава сигнал, съдържащ местоположението (и много повече)

•ADS-B сигнал се приема от приемник, свързан към системата

•Приемникът подава данни към системата

* **MLAT**

Мултилатерацията е техника за навигация и наблюдение, основана на измерването на времето на пристигане (TOAs) на енергийни вълни (радио, акустични, сеизмични и др.) С известна скорост на разпространение. Времето за възникване на TOA е произволно. (По принципа на реципрочност всеки концептуален метод, който може да се използва за навигация, може да се използва и за наблюдение и обратно.) За наблюдение, обект на интерес - при съвместно наблюдение, често превозно средство - предава на множество приемни станции, синхронизирани "часовници". За навигация множество синхронизирани станции предават на потребителски приемник. За да намерите координатите на потребител в n измерения (обикновено n = 2 или n = 3 ), най-малко n + 1 TOA трябва да бъдат измерени. Мултилатералните системи също се наричат хиперболични системи поради причини, обсъдени по-долу.



Приложение на MLAT(Мултилатерацията) са най-различни:

Проследяване на мобилни телефони - използване на множество базови станции за оценка на местоположението на телефона, или от самия телефон (наречен мултилатерация по низходяща линия), или от телефонната мрежа (наречена мултилатерация на връзката нагоре).

Мониторинг на намалени вертикални разделителни минимуми (RVSM) за определяне на точността на информацията за височина на транспондерите на въздухоплавателни средства в режим C / S. Приложението на многостранност към RVSM е демонстрирано за първи път от Roke Manor Research Limited през 1989 г.

Мултилатерация на широка площ (WAM) - система за наблюдение на въздухоплавателни средства, които измерват ТОА на емисиите от транспондера на въздухоплавателното средство (на 1090 MHz); в оперативна служба в няколко страни

1.2 Поставена задача

Проектиране и реализиране на мобилно приложение за проследяване на полети. В днешно време повечето приложения използват информация от интернет. За решаването на дадената задача и постигане на нужните резултати ще използваме информационна система - aviation-edge.comAPI ,която ни дава информация за полетите в реално време. Aviation-edge.com предоставя до някъде безплатен REST API интерфейс с цялата необходима информация относно полетите в реално време. За целите на приложението API интерфейсът е достъпен на адрес :http://aviation-edge.com/v2/public/flights?key=[API\_KEY] .Той предоставя няколко различни адреса за достъп (GET заявки).Също така ще използваме следните технологии за проектиране на приложението което да служи за търсене на полет в реално време.

**2.Използвани технологии**

**2.1** **Microsoft Visual Studio**

Microsoft Visual Studio е мощна интегрирана среда за разработка (на английски: integrated development environment, IDE) на софтуерни приложения за Windows и за платформата .NET Framework. Използва се за разработка на конзолни и графични потребителски интерфейс приложения, както и Windows Forms или WPF приложения, уеб сайтове, уеб приложения и уеб услуги на всички поддържани платформи от Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework и Microsoft Silverlight.

Visual Studio предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване (както за високо така и за машинно ниво), тестване на приложения, дизайн на потребителски интерфейс (форми, диалози, уеб страници, визуални контроли и други), моделиране на данни, моделиране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции. Могат да се добавят и плъгини, които повишават функционалността на почти всяко ниво – включително добавянето на поддръжка за source-control системи (като Subversion и Visual SourceSafe), добавяне на нови инструменти като редактори и визуални дизайнери за domain-specific languages или инструменти за други аспекти (като например: Team Foundation Server, Team Explorer).

Visual Studio поддържа различни езици за програмиране (например: C#, VB.NET, C/C++, F#, XML/XSLT, HTML/XHTML, JavaScript, CSS и други) и различни технологии за разработка на софтуер (Win32, COM, ASP.NET, ADO.NET Entity Framework, Windows Forms, WPF, Silverlight и още десетки други Windows и .NET технологии). Съществуват и отделни езикови версии на Visual Studio, които обаче предоставят по-ограничени услуги за потребителя: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

Microsoft предлага безплатни „Express“ издания на някои от компонентите на Visual Studio 2010. Като например: Visual Basic, Visual C#, Visual C++, Visual Web Developer и други. Чрез програмата DreamSpark, Microsoft предоставя на студенти напълно безплатно някои от своите продукти. Чрез нея могат да бъдат изтеглени Visual Studio 2012, 2010, 2008 и 2005 Professional Editions, заедно със специфичните езикови версии (Visual Basic, C++, C#, J#) на Visual Studio Express 2010 и други.

**2.2 Xamarin за Visual Studio**

Xamarin твърди, че е единствената IDE, която позволява създаването на собствени приложения за Android, iOS и Windows в Microsoft Visual Studio. Xamarin доставя добавки към Microsoft Visual Studio, които позволяват на разработчиците да изграждат приложения за Android, iOS и Windows в рамките на IDE, използвайки попълване на код и IntelliSense. Xamarin за Visual Studio също има разширения в рамките на Microsoft Visual Studio, които осигуряват поддръжка за изграждане, внедряване и отстраняване на грешки на приложения на симулатор или устройство. [38] В края на 2013 г. Xamarin и Microsoft обявиха партньорство, което включваше по-нататъшна техническа интеграция и клиентски програми, за да направи възможно създаването на съвместни бази за разработчици за всички мобилни платформи. [39]В допълнение, Xamarin сега включва поддръжка за Microsoft Portable Class Libraries [40] и повечето функции на C # 5.0 като async / чакат. Изпълнителният директор и съосновател на Xamarin, Нат Фридман, обяви съюза при старта на Visual Studio 2013 в Ню Йорк.

На 31 март 2016 г. Microsoft обединява целия софтуер на Xamarin с всяка версия на Microsoft Visual Studio, включително Visual Studio Community, и това добави различни функции на Xamarin, които трябва да бъдат предварително инсталирани във Visual Studio, като емулатор на iOS.

**2.3 .Net**

Microsoft .NET Framework е [платформа](https://bg.wikipedia.org/wiki/Платформа_(компютри)), създадена от [Microsoft](https://bg.wikipedia.org/wiki/Microsoft), която предоставя [програмен модел](https://bg.wikipedia.org/w/index.php?title=Програмен_модел&action=edit&redlink=1), библиотека от класове ([FCL](https://bg.wikipedia.org/w/index.php?title=FCL&action=edit&redlink=1), Framework Class Library) и среда за изпълнение на написан специално за нея програмен код ([CLR](https://bg.wikipedia.org/w/index.php?title=CLR&action=edit&redlink=1), Common Language Runtime). Тя е ключов елемент от стратегията за развитие на [Microsoft](https://bg.wikipedia.org/wiki/Microsoft), чиято цел е повечето нови приложения за [Windows](https://bg.wikipedia.org/wiki/Windows) да бъдат базирани на .NET Framework.

.NET приложенията се пишат на езици от високо ниво ([C#](https://bg.wikipedia.org/wiki/C_Sharp), [VB.NET](https://bg.wikipedia.org/w/index.php?title=VB.NET&action=edit&redlink=1), [C++/CLI](https://bg.wikipedia.org/w/index.php?title=C%2B%2B/CLI&action=edit&redlink=1) и други) и се компилират до платформено-независим междинен език, наречен [CIL](https://bg.wikipedia.org/w/index.php?title=CIL&action=edit&redlink=1) (Common Intermediate Language). По време на изпълнение CIL кодът (т. нар. „управляван код“) бива автоматично компилиран от CLR за конкретната хардуерна платформа и операционна система, с която работи потребителят.

.NET Framework се разпространява с [Windows Server 2003](https://bg.wikipedia.org/w/index.php?title=Windows_Server_2003&action=edit&redlink=1), [Windows Server 2008](https://bg.wikipedia.org/wiki/Windows_Server_2008), [Windows Vista](https://bg.wikipedia.org/wiki/Windows_Vista) и [Windows 7](https://bg.wikipedia.org/wiki/Windows_7) като може да бъде инсталиран и на по-стари версии на Windows. Целта на проекта [Mono](https://bg.wikipedia.org/w/index.php?title=Mono_(софтуер)&action=edit&redlink=1), спонсориран от [Novell](https://bg.wikipedia.org/w/index.php?title=Novell&action=edit&redlink=1), е да даде възможност програми, написани на .NET Framework, да работят и на други операционни системи, като Linux, Solaris, Mac OS, BSD и др.

В основата на философията на .NET Framework са залегнали следните основни цели и идеи:

- Платформена независимост – Една от основните цели на .NET Framework е да даде възможност за създаването на платформено независими приложения, т.е. на приложения, които могат да работят на различни операционни системи и хардуерни конфигурации. До този момент Microsoft са създали имплементации на .NET Framework, които работят на x86 и x64 персонални компютри, Windows CE базирани устройства ([PDA](https://bg.wikipedia.org/wiki/PDA), [SmartPhone](https://bg.wikipedia.org/w/index.php?title=SmartPhone&action=edit&redlink=1), [Tablet PC](https://bg.wikipedia.org/w/index.php?title=Tablet_PC&action=edit&redlink=1)), както и игровата конзола [XBox 360](https://bg.wikipedia.org/w/index.php?title=XBox_360&action=edit&redlink=1). Спецификациите на CLI (Common Language Infrastructure, което обхваща BCL, CTS и CIL), както и езиците C# и C++/CLI са предадени на [ECMA](https://bg.wikipedia.org/w/index.php?title=ECMA&action=edit&redlink=1) и [ISO](https://bg.wikipedia.org/wiki/ISO) и са отворени стандарти. По този начин независими разработчици могат да поставят .NET Framework и прилежащите му езици на други софтуерни и хардуерни платформи.

- Съвместимост с вече съществуващите технологии – .NET Framework дава възможност за достъп до функционалност, реализирана в приложения извън .NET средата. Пространствата System.Runtime.InteropServices и System.EnterpriseServices дават възможност за достъп до [COM](https://bg.wikipedia.org/w/index.php?title=Component_Object_Model&action=edit&redlink=1) компоненти, докато с [P/Invoke](https://bg.wikipedia.org/w/index.php?title=Platform_Invocation_Services&action=edit&redlink=1) може да се извиква неуправляван код (native code).

- Common Runtime Engine – Програмните езици, които се поддържат от .NET Framework биват компилирани в [междинен код](https://bg.wikipedia.org/w/index.php?title=Междинен_код&action=edit&redlink=1) известен като [CIL](https://bg.wikipedia.org/w/index.php?title=CIL&action=edit&redlink=1) (Common Intermediate Language) а по-рано и като [MSIL](https://bg.wikipedia.org/w/index.php?title=MSIL&action=edit&redlink=1) (Microsoft Intermediate Language). При изпълнението си този междинен код не бива интерпретиран като при други виртуални машини, а вместо това компилиран по начин, известен като [JIT](https://bg.wikipedia.org/w/index.php?title=JIT&action=edit&redlink=1) (Just In Time) компилация в платформено-зависим машинен код (native code).

- Езикова независимост – Приложения за .NET Framework могат да бъдат създавани на няколко съвместими [програмни езици](https://bg.wikipedia.org/wiki/Програмни_езици), сред които [C#](https://bg.wikipedia.org/wiki/C_Sharp), [VB.NET](https://bg.wikipedia.org/w/index.php?title=VB.NET&action=edit&redlink=1) и [C++/CLI](https://bg.wikipedia.org/w/index.php?title=C%2B%2B/CLI&action=edit&redlink=1). Това е възможно благодарение на съвместимостта на типовете данни, които отделните езици поддържат. [CTS](https://bg.wikipedia.org/w/index.php?title=CTS&action=edit&redlink=1) (Common Type System) дефинира всички базови типове данни, както и начинът, по който те могат да бъдат конвертирани един в друг. Тези типове са споделени между всички .NET езици и са стандартизирани в CLI.

- Улеснена инсталация – Програмите, създадени на .NET Framework, както и техните компоненти, могат да бъдат инсталирани с просто копиране в желаната директория – процес, известен като XCopy Deployment.

- Сигурност – Програмният код, написан на .NET, се нарича управляван код, а също и защитен код, тъй като е изолиран от хардуерната среда, в която оперира, и това го предпазва от някои видове програмни грешки, които правят кода уязвим за атаки, например препълването на буфера (buffer overflow). Също така .NET Framework предлага система за сигурност, която може да се използва от всички .NET приложения.

**2.4** **C#**

C# се използва за разработване на софтуер, който работи на устройства Linux, Android и iOS.

Първата версия на C# е разработена от [Microsoft](https://bg.wikipedia.org/wiki/Microsoft) в периода 1999 – 2002 г. и е пусната официално в употреба през 2002 г., като част от [.NET](https://bg.wikipedia.org/wiki/.NET) платформата, която има за цел да улесни съществено разработката на софтуер за [Windows](https://bg.wikipedia.org/wiki/Windows) среда чрез качествено нов подход към програмирането, базиран на концепциите за „виртуална машина“ и „управляван код“. По това време езикът и платформата [Java](https://bg.wikipedia.org/wiki/Java), изградени върху същите концепции, се радват на огромен успех във всички сфери на разработката на софтуер и разработката на C# и [.NET](https://bg.wikipedia.org/wiki/.NET) е естественият отговор на [Microsoft](https://bg.wikipedia.org/wiki/Microsoft) срещу успехите на [Java](https://bg.wikipedia.org/wiki/Java) технологията. Едно от най-големите предимства на .NET Framework е вграденото автоматично управление на паметта. То предпазва програмистите от сложната задача сами да заделят памет за обектите и да търсят подходящия момент за нейното освобождаване. Това сериозно повишава производителността на програмистите и увеличава качеството на програмите, писани на C#.

Едно от най-големите предимства на [.NET Framework](https://bg.wikipedia.org/wiki/.NET_Framework) и C# е вграденото автоматично управление на паметта. То освобождава програмистите от сложната задача сами да заделят памет за обектите и да търсят подходящия момент за нейното освобождаване. Това сериозно повишава производителността на програмистите и увеличава качеството на програмите, писани на C#.

За управлението на паметта в [C#](https://bg.wikipedia.org/wiki/.NET_Framework) се грижи специален компонент от CLR, наречен „система за почистване на паметта“ (garbage collector). Основната задача на системата Garbage collector-а е да следи кога заделената памет за променливи и обекти вече не се използва, да я освобождава и да я прави достъпна за последващи заделяния на нови обекти.

C# се използва за разработване на софтуер, който работи на устройства Linux, Android и iOS.

Той и неговите приложения трябва да осигуряват поддръжка на принципите за софтуерно инженерство, като „strong type“ проверка и откриване на опити за използване на неинициализирани променливи. Популярността на софтуера, издръжливостта и производителността за програмистите са важни.

C# е предназначен да бъде подходящ за писане на приложения както за хоствани, така и за вградени системи, вариращи от най-големите, които използват сложни операционни системи, до най-малките – със специални функции.

C# просто казано представлява 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 (.exe или .dll).

Ето няколко причини да работим с него:

– Езикът притежава приятен и лесен за научаване синтаксис;

– Силна близост с Java, C, C++, заради което е подходящ за тези, които владеят някои от посочените езици;

– Широко приложение. Позволява разработването на сървърна back-end логика, уеб с ASP.NET MVC, мобилни приложения с Xamarin, десктоп приложения с WPF, игри, Augmented Reality и Mixed Reality приложения с Unity game engine, embedded приложения за Windows 10 IoT и т.н.

– Особено подходящ за начинаещи;

– Наличие на огромен брой от технологични рамки, библиотеки и инструменти за разработка на езика;

– Популярен и гъвкав;

**2.5 XAML**

XAML („замъл“)(от [английски](https://bg.wikipedia.org/wiki/Английски_език): *Extensible Application Markup Language*) e декларативен markup език използван за иницализиране на структурирани стойности и обекти. Базиран е на [XML](https://bg.wikipedia.org/wiki/XML), създаден от [Microsoft](https://bg.wikipedia.org/wiki/Microsoft) с цел опростяване на създаването на потребителски интерфейс ([UI](https://bg.wikipedia.org/w/index.php?title=UI&action=edit&redlink=1)) за .NET приложения.

Езикът е представен през 2006 г. заедно с [WPF](https://bg.wikipedia.org/w/index.php?title=WPF&action=edit&redlink=1) (Windows Presentation Foundation) в .NET 3.0. WPF е графична подсистема за [рендериране](https://bg.wikipedia.org/wiki/Рендериране) на потребителски интерфейс в Windows-базирани приложения, която ползва XAML. Впоследствие през 2007 г. е представен [Silverlight](https://bg.wikipedia.org/w/index.php?title=Silverlight_Silverlight&action=edit&redlink=1), който е фреймуърк изпълняващ функции, много близки до тези на [Adobe Flash](https://bg.wikipedia.org/wiki/Adobe_Flash). През 2011 г. е представен и [Windows Runtime XAML Framework](https://bg.wikipedia.org/w/index.php?title=Windows_Runtime_XAML_Framework&action=edit&redlink=1), който върши същото за програми в новия [„Metro“](https://bg.wikipedia.org/w/index.php?title=Metro_(език_за_програмиране)&action=edit&redlink=1) стил.

Основните предимства на XAML са че разделя UI от [бизнес логиката](https://bg.wikipedia.org/w/index.php?title=Бизнес_логика&action=edit&redlink=1) и че позволява едновременно да са работи върху логиката и интерфейса на една програма с различни инструменти ([Visual Studio](https://bg.wikipedia.org/wiki/Visual_Studio) и [Microsoft Expression Blend](https://bg.wikipedia.org/w/index.php?title=Microsoft_Expression_Blend&action=edit&redlink=1)).

Елементите на XAML са директно свързани със [CLR](https://bg.wikipedia.org/w/index.php?title=CLR&action=edit&redlink=1) обекти, а атрибутите – със CLR свойства и събития върху тези обекти. XAML файловете могат да бъдат редактирани с инструменти за визуален дизайн като Microsoft Expression Blend и Microsoft Visual Studio, както и със стандартни [текстови редактори](https://bg.wikipedia.org/wiki/Текстов_редактор), кодови редактори като [XAMLPad](https://bg.wikipedia.org/w/index.php?title=XAMLPad&action=edit&redlink=1)или графични редактори като [Vector Architect](https://bg.wikipedia.org/w/index.php?title=Vector_Architect&action=edit&redlink=1).

Всички графични елементи в XAML са Direct3D апликации (използват видео картата за рендериране), което води до висококачествен и разнообразен потребителски интерфейс. Освен това графиките са вектор-базирани, което позволява оразмеряване на обектите без загуби в качеството им. XAML поддържа множество от функции, като анимации с таймери или свързани с действия на потребителя, видео и аудио във всички формати в които операционната система поддържа, както и връзки между тези функции. XAML имплементира логическа пикселна система използваща за стойности floating-point (числа с плаваща запетая), както и поддържа 32 битови [RGBA](https://bg.wikipedia.org/w/index.php?title=RGBA&action=edit&redlink=1) цветове.

Всичко създадено или имплементирано с XAML може да бъде направено и с по-традиционен .NET език като [C#](https://bg.wikipedia.org/wiki/C) или [Visual Basic.NET](https://bg.wikipedia.org/w/index.php?title=Visual_Basic.NET&action=edit&redlink=1). Понеже е базиран на XML, обаче, XAML е по-прост, а и девелъпъри и дизайнери могат да споделят и редактират съдържанието на един проект свободно помежду си без да е нужна компилация.

Синтаксисът на XAML реално е чист XML и силно наподобява HTML – идеята му е да се правят потребителски интерфейси за различни програми.

Накратко XAML се състои от Елементи (Button, Template, Grid) и Свойства(нещата които различават елементите, характеристики – Width, Height), като Елементите могат да притежават Свойства.

Всеки XAML файл има точно един елемент, който е неговия „корен“ – той обявява обекта, който ще бъде концепцуалния корен на някаква софтуерна структура (например страница или обектния граф на цялата рънтайм дефиниция на приложението). (Window или UserControl в зависимост дали ползваме WPF или Silverlight)

Елемент се обявява с отварящ и затварящ таг, точно както в HTML (<отв таг> </затварящ таг> <таг />), а вътре в тага се слагат свойствата (размери, местоположение, цвят и т.н.), които получават стойности чрез равно и стойност в кавички (<отварящ Таг свойство1="стойност1" свойство2="стойност2">). Между отварящия и затварящия таг на елемента могат да се намират други елементи.

* **Основни Елементи XAML:**

Панелите се използват за оформяне на лейаут на интерфейса. В тях се поставят другите елементи и целта на панела е да се декларира местоположението на тези елементи.

Canvas – Рамка. Всички дъщерни елементи се позиционират с координати спрямо нея, подадени от програмиста.

StackPanel – Дъщерните елементи се редят в хоризонтален или вертикален ред.

Grid – Решетка. Всички дъщерни елементи се позиционират като в таблица с колони и редове. Button – обявява бутон, копче.

Window, UserControl – основния елемент, в който се намират всички останали елементи на съответния интерфейс

TextBlock – елемент, в който се поставя текст, който може само да се чете от потребителя.

TextBox – Много подобен на TextBlock елемент, в който се поставя текст, който може да бъде редактиран и избран от потребителя. И в двата текстови елемента текстът може да се поставя по два начина **<TextBlock>**ТЕКСТ**</TextBlock>** и

**<TextBlock** Text="ТЕКСТ" **/>.**

**2.6 MVVM**

Шаблонът Model-View-ViewModel (MVVM) може да се използва при всички XAML платформи. Неговата цел е да се осигури чисто разделяне на връзките между контролите от потребителския интерфейс и тяхната логика. В MVVM шаблона има 3 основни компонента: model (модел), view (изглед) и view model (изглед-модел). Всеки от тях изпълнява различна и отделна роля.

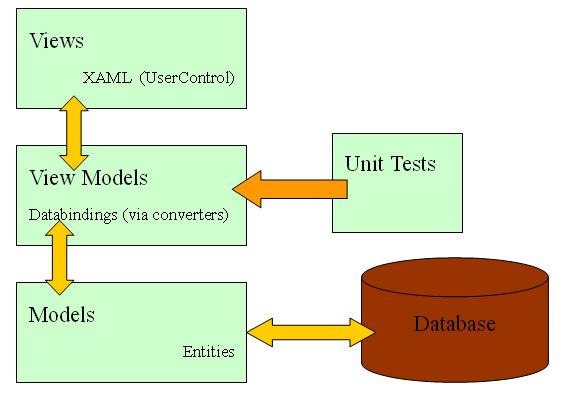
Компонентите са отделени един от друг, което от своя страна позволява:

- Размяна/подмяна на компоненти

- Вътрешната имплементация на компонент да бъде променена без това да засегне останалите

- Работата по компонентите да се извършва независимо

- Индивидуално тестване на компонентите

За да се разбере по-добре какви са отговорностите на трите компонента е важно да се схване и как те си взаимодействат помежду си. 

Слоят View, бидейки най-високо ниво в архитектурата знае всичко за ViewModel, a ViewModel от своя страна знае за Model. Но Model не е наясно за ViewModel и ViewModel не знае за View. ViewModel изолира класовете на View от тези на Model и позволява на Model да се развива независимо от View. View слоят е отговорен за създаването на структурата, подредбата и външния вид на това, което вижда потребителя на екрана. View се дефинира изцяло с XAML, с ограничен code- behind, който обикновено не съдържа бизнес логика.

Едно View може да има собствен ViewModel или да наследява този на своя родител. Всяко View получава данните си от собствения ViewModel чрез връзки (bindings) или чрез извикване на методи от ViewModel. По време на изпълнение на програмата View се променя, когато UI отговорят на ViewModel свойства, повдигащия събития за известяване за промяна (change notification events).

Тук има различни възможности за изпълнение на код в ViewModel в отговор на взаимодействията във View, като например натискане на бутон или избор на елемент. Ако контролата е източник на команда (Command Source), свойството ѝ Command може да бъде свързано с ICommand свойство на ViewModel. Когато се извика (invoke) командата на контролата се изпълнява кода във ViewModel.

Model Модела в MVVM е имплементация на основния модел на приложението, който включва модел на данните, заедно с бизнес и валидационната логика. Примери за обекти на модела са хранилища (repositories), бизнес обекти, data transfer objects (DTOs), Plain Old CLR Objects (POCOs) и генерирани entity и proxy обекти.

Слоят ViewModel работи като посредник между View и Model и е отговорен за обработката на логиката на изгледа. Обикновено ViewModel взаимодейства с Model като извиква методи от неговите класове. След това ViewModel предоставя данни от модела в подходящ за изгледа формат. При това ViewModel може да преформатира извлечените от модела данни, като например ги опрости и ги направи по-лесни за работа в изгледа. Той също така предоставя и имплементация на команди, които потребителя на приложението инициира в изгледа. Например, когато потребителя натисне върху бутон в UI, това действие може да инициира команда във ViewModel. ViewModel може да бъде отговорен също за дефинирането на промени в логически състояние, което засяга някои аспекти на визуализацията във View, като например като например индикация, че някоя операция е предстояща.

**3.Съществуващи решения**

**3.1 Flightradar24**

Flightradar24 е обществена услуга, която позволява наблюдение на положението на въздухоплавателното средство в реално време. Проследяване чрез позицията на обслужване на въздухоплавателното средство е възможно, само ако тя е оборудвана с транспондер като ADS-B и един включена.

Услугата показва координатите, височината и скоростта на самолета, а също така показва на картата разстоянието, изминато от мястото на тръгване. Ако информацията е налична в специализирани източници (например на уебсайтове за споттер ), тя може също така да покаже снимка, вид самолет, номер на борда, принадлежност към авиокомпанията, пункт за отпътуване и качване и редица друга информация. Услугата записва историята на полетите.

Услугата работи в уеб браузъри на настолни компютри и е достъпна и чрез приложения за Mac OS X , iOS - ( iPhone , iPad , iPod Touch ), както и за Android и Windows + Windows Phone 8 .

В уеб браузърите на настолни компютри продължителността на безплатната сесия на услугата е 30 минути, след което на потребителя се предлага платен абонамент за използване на услугата без ограничения или, след като презаредите страницата, можете да отворите нова сесия.

* **Принцип на работа** - Услугата използва ADS-B технология за проследяване и извличане на информация за самолета . Самолетът е оборудван с най-Bed ADS и - транспондера по време на целия режим [ "S"] полет, приблизително всеки втори генерира и изпраща излъчване излъчване (1090 MHz), открит радио съобщение, което съдържа изтече в точка данни на заминаване - тяхната точни координати (определени с помощта на GPS ), текущата ви скорост, надморска височина и друга информация.

Част от основните функционалности на приложението са:

- Търсене по номер (фиг. 1.1).

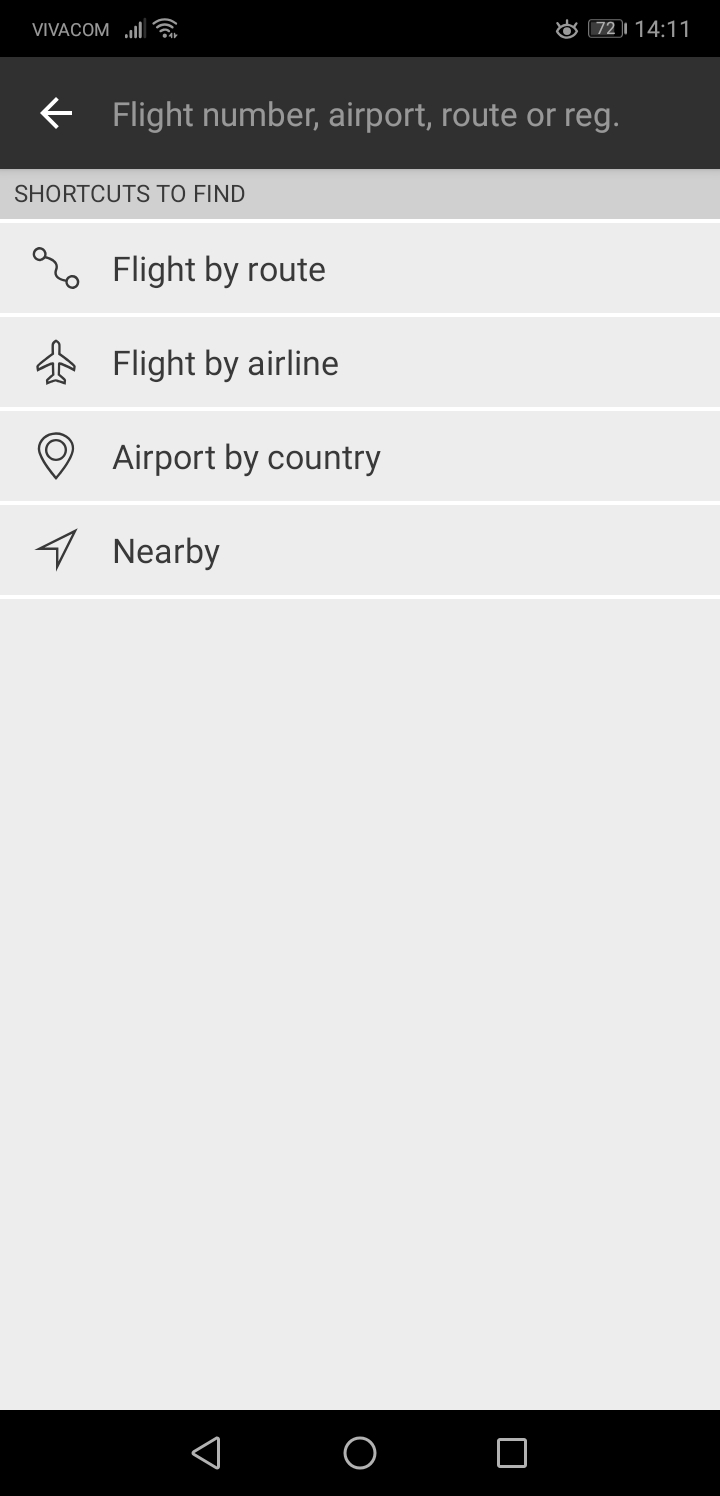
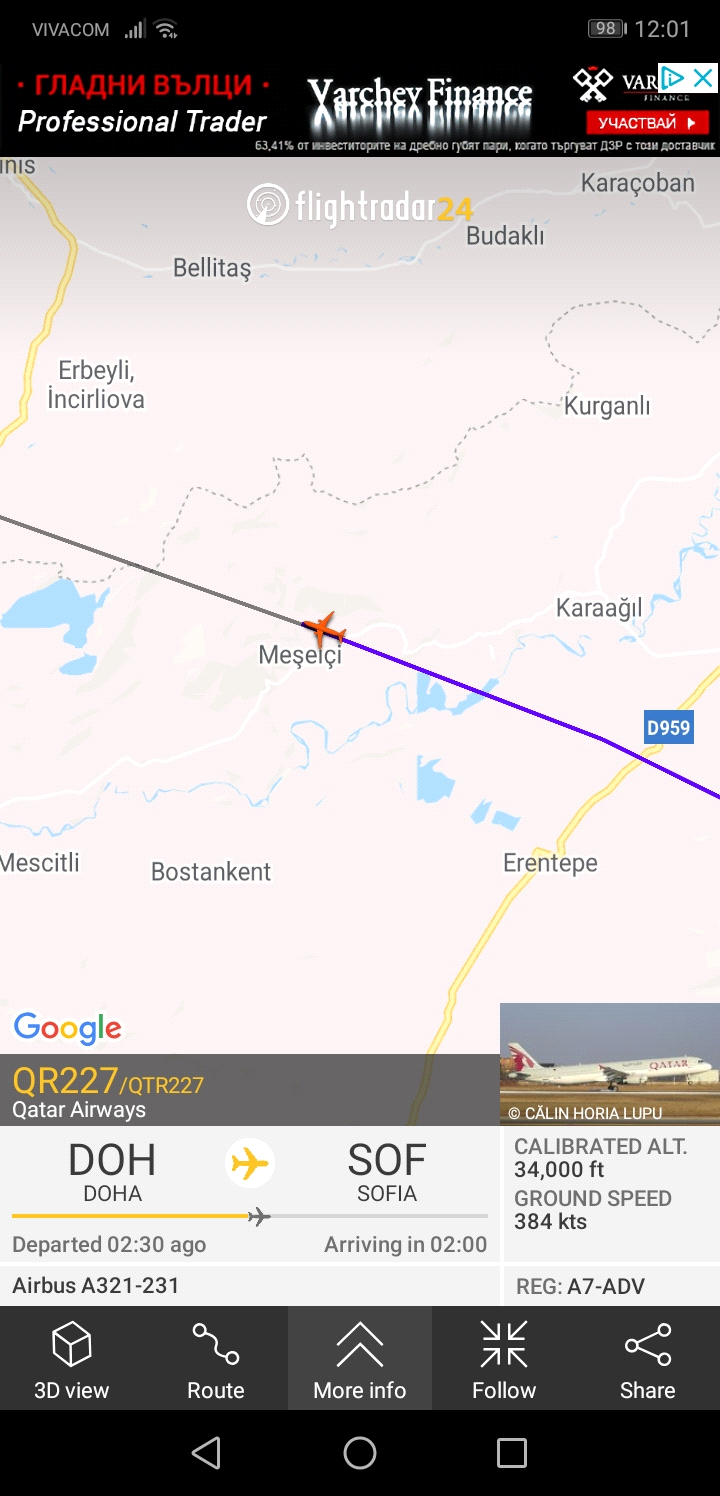
- Търсене по машрут (фиг. 1.1).

- Търсене по авиокомпания (фиг. 1.1).

- Търсене на летища (фиг. 1.1).

- Резултат от търсене(фиг 1.2).

-3D View (1.3).

Търсене (фиг. 1.1). Резултат от търсенето (фиг. 1.2).



3D View (фиг. 1.3).

**Положителни страни на приложението:**

-Много начини на търсене

-Утвърдено приложение с много потребители

-3D view

**Негативни страни на приложението:**

- Повечето от функционалностите са платени.

**3.2. Flightradar (Live)**

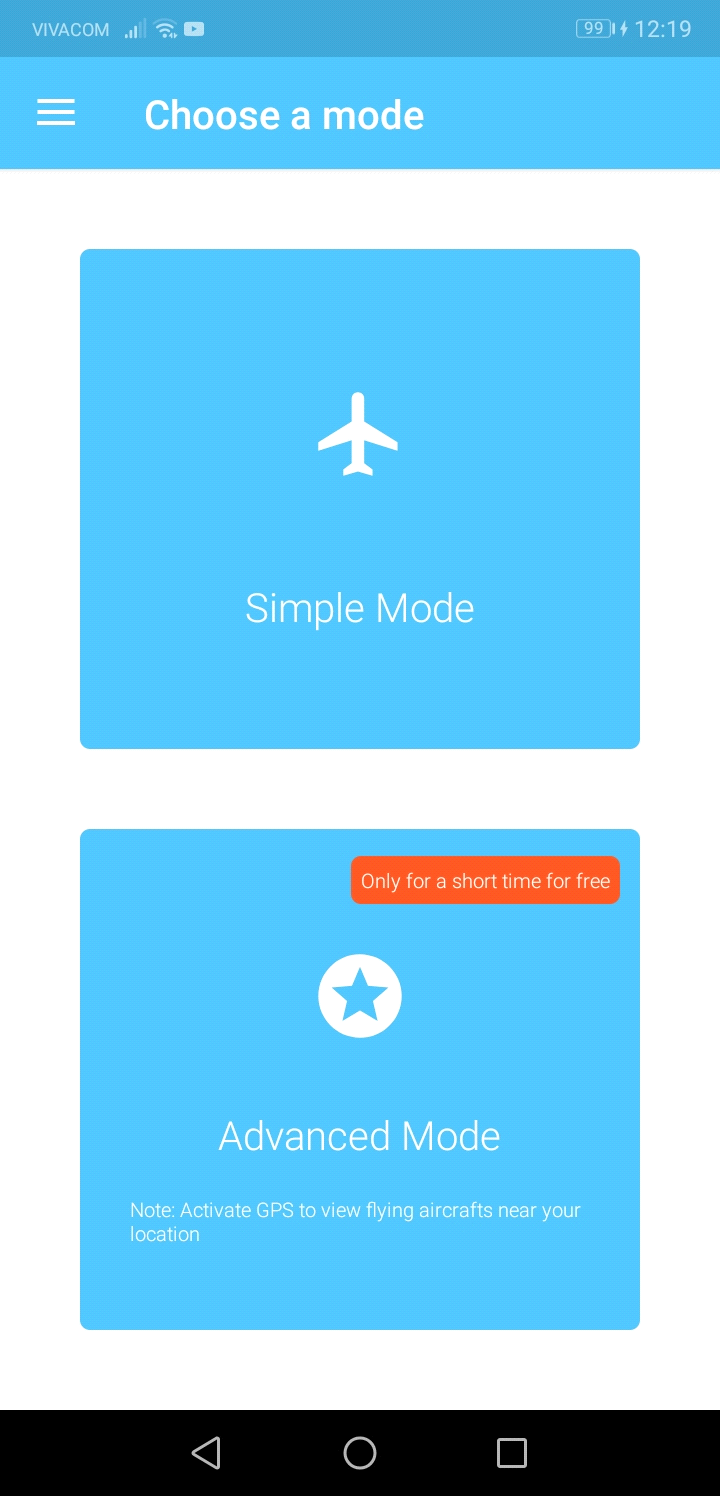
Мобилно приложение което e мултиплатформено . При стартитане на самото приложение ние трябва да изберем с коя от двете версии да работим . Първата е simple версията която е безплатна версия.В безплатната потребителя няма право на търсене по каквито и да е били полета.(фиг.)

В платената версия която предоставя приложението advanced (платена версия).Потребителя има функция за търсене и само това.

Част от основните функционалности на приложението са:

- Избор на два вида (Mode) обикновен (Simple) и напреднал (Advanced). (фиг. 1.4).

- Търсене по номер налично само в платената версия (Advanced mode). (фиг. 1.5).

Избор на работа (фиг. 1.4). Търсене Advanced mode (фиг. 1.5)

**Положителни страни на приложението:**

-информация в реално време

**Негативни страни на приложението:**

- Бавен интерфейс

- Търсенето е възможно само в платената версия

- Изкачане на много реклами

**3.3. The Flight Tracker**

Приложението получава данни от системите за контрол на въздушното движение в над 45 държави, мрежата на FlightAware от наземни станции ADS-B в 195 страни, глобалната космическа базова станция ADS-B на Aireon и datalink (сателит / УКВ) чрез всеки основен доставчик, включително ARINC, SITA , Satcom Direct, Garmin и Honeywell GoDirect. Приложението е предназначено за мобилни телефони с операционна система Android. Oсновната цел е проследяване на даден полет.

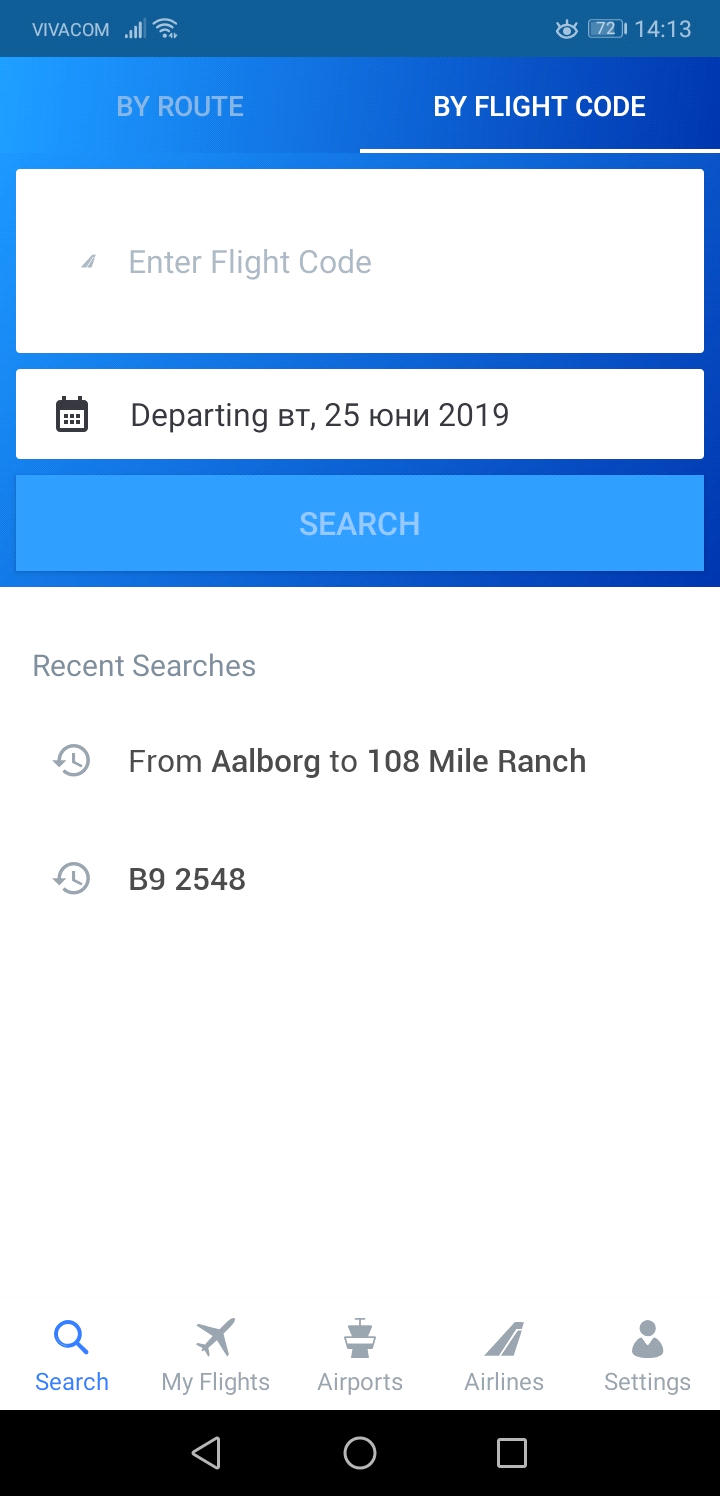
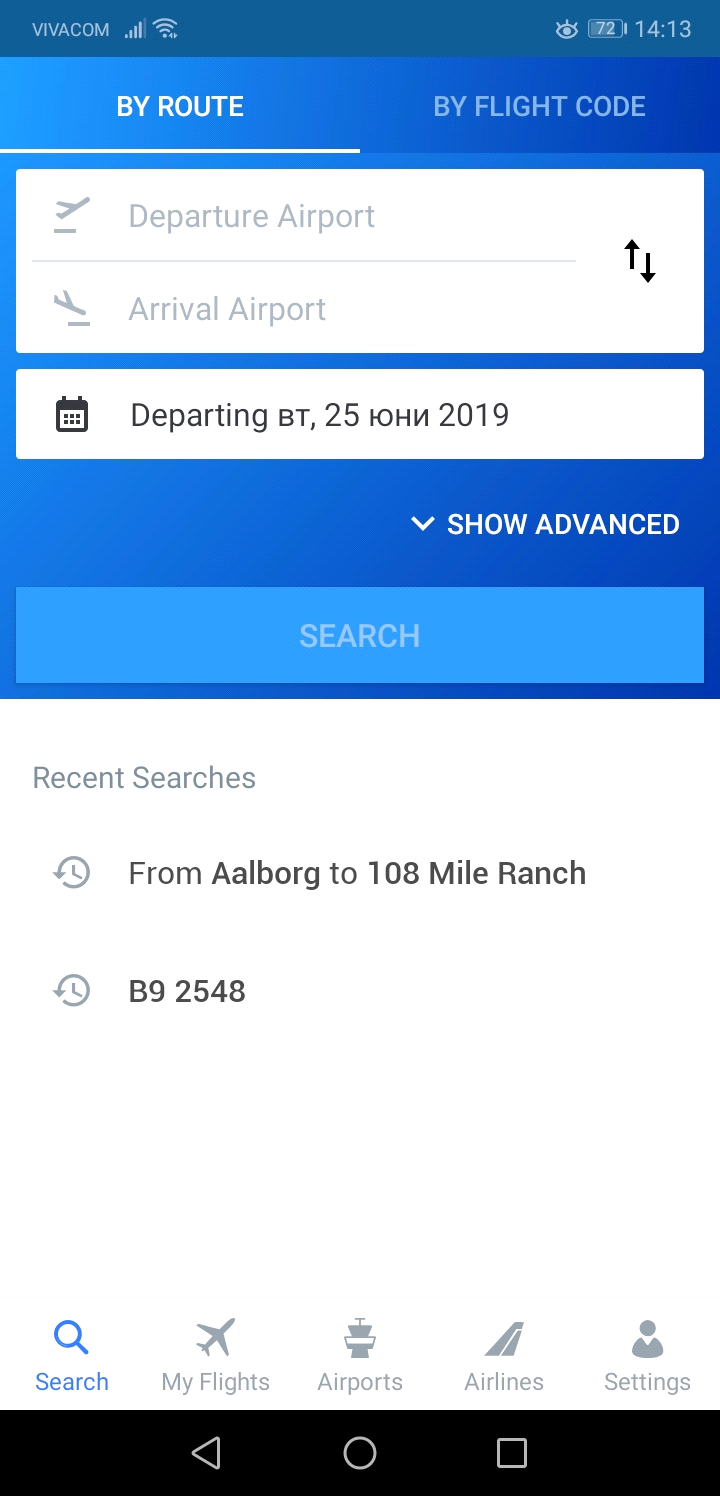
Част от основните функционалности на приложението са:

- Търсене по номер и търсене по маршрут (фиг. 1.6).

- Търсене по авиокомпания (фиг. 1.7).

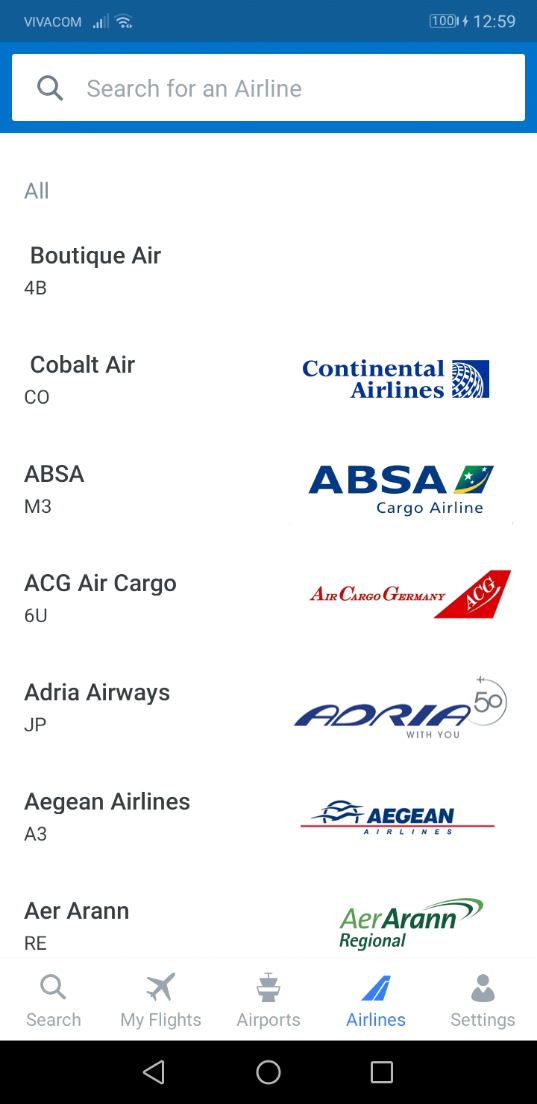
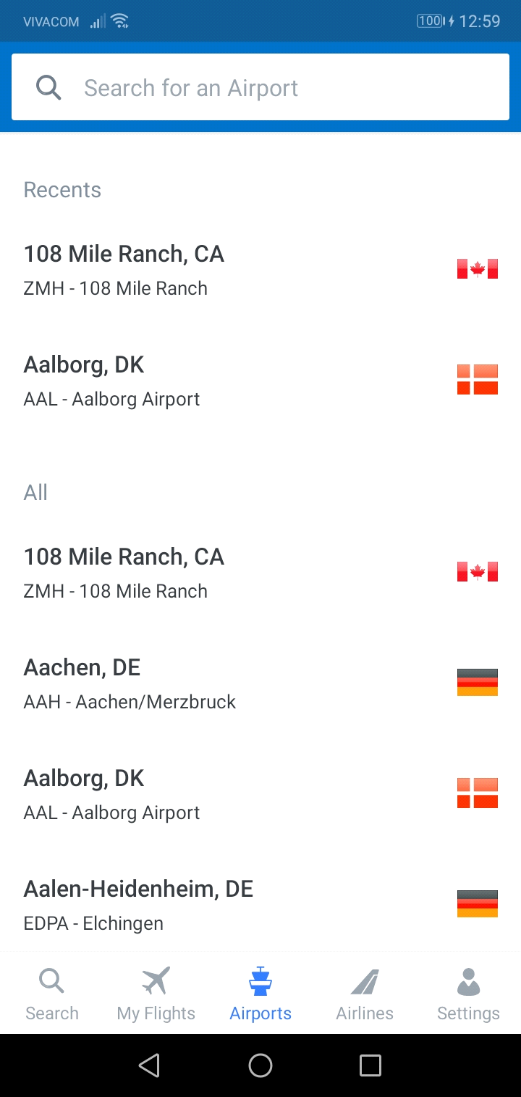
- Търсене на летища (фиг. 1.8).

Както се вижда на снимките по-отдолу интерфейса който предлага приложението е изчистен структуриран и лесен за работа от потребителя. Също така има пази и наши предишни търсения за дадени полети. Програмата получава разписания на авиокомпаниите месеци преди полета.

Търсене по номер и търсене по машрут (фиг. 1.6).

Снимките показват наличието за търсене не само по номер а и по авиокомпания и летище .

  Търсене по авиокомпания (фиг. 1.7). Търсене на летища (фиг. 1.8).

**Положителни страни на приложението:**

- Изчистен и структориран интерфейс

- Бързо и лесено тъсене

**Негативни страни на приложението:**

- Не намирам

Глава 2 – Теоретично решение на поставената задача;

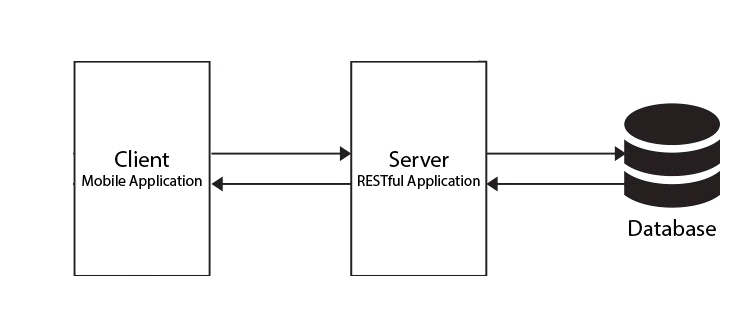
За решаването да дадената задача използваме информационната система на aviation-edge.comAPI , която е безплатна и дава данни на даден полети в реално време.

Свързването към API-то става единствено чрез ключа API-KEY, който дава директен достъп до всички данни на полета на живо. API-то за проследяване на полети позволява на клиентите да показват в своите приложения или на своите уебсайтове и други платформи действителното проследяване на полетите, включително цялата им подходяща информация. Тази услуга е централна за авиационни данни и се поддържа постоянно в база данни. Тъй като данните са на живо, актуализациите се случват в кратки интервали от време(няколко минути). Функцията на API-то е да събира и поддържа пълните авиационни данни , така че да могат да се създават функционални инструменти за ползването им.

Както се запознахме по-нагоре данните на API-то за проследяване на полета се събират чрез ADS-B системи, както и използването на полетни графици за предвиждане на местоположението и попълване на пропуските, когато е необходимо. Също така получава данни от авиационни партньори и комбинира всичко това в централната система.

**1.****Цялостна архитектура**

Цялостната архитектура на приложението е изградена от мобилно приложение (клиент), RESTful приложение (сървър) и база данни. Мобилното приложение комуникира с RESTful приложението чрез HTTP заявки. Връзката между RESTful приложението и базата осигурява извличането и записването на данни, съхранявани в базата данни.



Най-използваните HTTP заявките са GET,POST,PUT,DELETE.

**HTTP GET**

GET заявки са за да изтегляне и само представяне на информация / информация - и да не я променяте по никакъв начин. Тъй като GET заявките не променят състоянието на ресурса, те са така наречените безопасни методи . Освен това, GET API трябва да бъде idempotent , което означава, че отправянето на множество еднакви заявки трябва да произвежда един и същ резултат всеки път.

**HTTP POST**

POST е за създаване на нови подчинени ресурси , например файл е подчинен на директория, която го съдържа или ред е подчинен на таблица на база данни. По отношение на REST, POST методите се използват за създаване на нов ресурс в колекцията от ресурси.

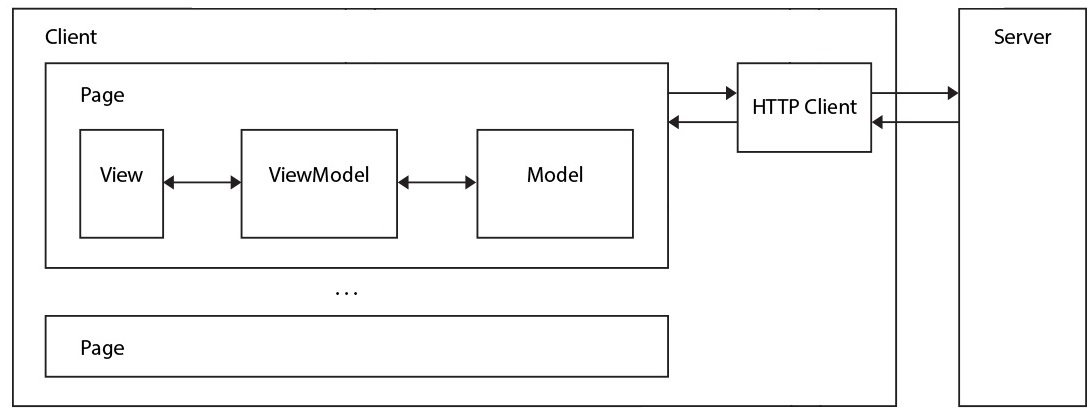
**HTTP PUT**

Методът PUT замества всички текущи представяния на целевия източник със съдържанието. Извикването на една и съща заявка PUT многократно винаги ще доведе до един и същ резултат.

**DELETE**

Методът DELETE изтрива указания източник. на заявката.



**2.Архитектура на мобилното приложение**Архитектурата на мобилното приложение е изградена главно на принципа на страници, всяка от които съдържа View, Model и ViewModel. Навигирането между страниците става възможно с помощта на команди и т.н. Binding на контролите към съответната команда.Страниците използват HTTP Client за връзка между мобилното приложение и сървъра, получавайки данни под формата на JSON обекти получени от сървъра. (Архитектура на мобилното приложение).

Комуникацията се между приложението и информационната система се осъществява чрез приложно-програмния интерфейс, като извлечените данни се използват от приложението за предоставяне на потребителите по удобен за тях начин. Използваният текстови формат за отговор на заявките е JSON. Основната информация за реализиране на приложението се предоставя от информационната система.. За целите на приложението API интерфейсът е достъпен на адрес :http://aviation-edge.com/v2/public/flights?key=[API\_KEY] .Той предоставя няколко различни адреса за GET заявки. Отговорите са в JSON формат и са показани по-долу като примери.

**3. Данни предоставени от информационната система.**

Данни, включени в API за проследяване на полети са:

География: Информация за местоположението като географска ширина, дължина, надморска височина и посока.

Пример: geography": {

"latitude": 43.5033,

"longitude": -79.1297,

"altitude": 7833.36,

"direction": 70 }

Скорост: вертикална и хоризонтална скорост на въздухоплавателното средство.Пример : "speed": {

"horizontal": 833.4,

"isGround": 0,

"vertical": 0

}

Отпътуване и пристигане: IATA кодове и ICAO кодове на мястото на отпътуване и пристигане. Пример: "departure": { "arrival": {

"iataCode": "YHM", "iataCode": "YQM", "icaoCode": "CYHM" } "icaoCode": "CYQM" }

Самолет и полет: IATA и ICAO номер на полета и регистрационен номер, ICAO код и ICAO24 код на въздухоплавателното средство.

"aircraft": { "airline": {

"icaoCode": "B763", "iataCode": "W8",

"regNumber": "CGYAJ", "icaoCode": "CJT" }

"icao24": "C08412" }

Авиокомпания: IATA код и ICAO код на авиокомпанията.

Пример: "airline": {

"iataCode": "W8",

"icaoCode": "CJT"}

Информация за системата: Squawk, статус и последна актуализация.

Пример: "system": {

"updated": 1513148168,

"squawk": "0000"

"status": "en-route"}

Глава 3 – Описание на софтуерната част;

Мобилното приложение е изградено от един проект състоящ се от страници (views), view model-и асоциирани към тях и модели на обектите (models).

Всяка страница има съответно изглед (xaml файл), логически код (cs файл) и view model (cs файл).Навигацията между отделните страници е реализирана с помощта на команди, които са асоциирани към съответните контроли от потребителския интерфейс

**1. Контроли**

Потребителският интерфейс е изграден от страници и контроли вградени в Xamarin.

Всяка страница се дели на Xaml и .cs файлове. В .cs файловете се навира т.н. Code Behind или основната логика и командите, които трябва да се изпълняват. Тъй като проекта следва MVVM архитектурата Code behind e ограничен и всяка страница си има ViewModel, в който са имплементирани всички команди. В Code Behind се оказва единствено съответния ViewModel.

public MainPage()

{

this.BindingContext = new MainPageViewModel();

InitializeComponent();

}

Код . Пример за .cs файл с Code Behind

**2.Модел на данните и View-та.**

Xaml файловете са xml базирани и те изграждат визулната част на страницата. В тях се добавят желаните контроли. Контролите се свързват към команди от съответния ViewModel и при настъпване на съответния event се изпълнява командата.(Кода по горе).

Приложението са състои модели (на данните),View-та и контролери

Моделът представя действителните данни и / или информация, с която имаме работа. Пример за модел може да бъде контакт (съдържащ име, телефонен номер, адрес и т.н.) или характеристиките на публикуваща точка на поточно предаване на живо.

Модела съхранява информацията, но не поведението или услугите, които манипулират информацията.

В приложението са изградени класове за модели за всичките данните който предоставя информационната система която използваме .Това е направено за бъдещо подобрение на функционалностите на приложението. В текущата версия използваме главно два класа.

В класа Geography използваме главно данни за местоположението на полета. Това са Latitude- географска координатна която определя север - юг позицията на точка от земната повърхност , и Longitude - географска координатна което определя изток - запад позицията на точка на на Земята е на повърхността. Тези две полета са основни в намирането на полета на картата. Тъй като данните са динамични обновяването им се случва в периода на една минута.

namespace Flight.Models

{

public class Geography

{

public double Latitude { get; set; }

public double Longitude { get; set; }

public double Altitude { get; set; }

public double Direction { get; set; }

}

}

Другия клас с модел на данните е Flight. Класа е основен за приложението защото в него се съдържат данните за номера на полета ,с помощта на този уникален индификатор може да намерим търсения от нас полет сред хилядите които се извършват в света .

namespace Flight.Models

{

public class Flight

{

public string IataNumber { get; set; }

public string IcaoNumber { get; set; }

public string Number { get; set; }

}

}

Функцията по-долу извършва извличането на данните от информационната система и проверката им, също така и навигирането на страниците в зависимост дали полета съществува или не. При въвеждането за номера на търсения полет приложението извършва търсене в базата данни на информационната система (API).

public async Task<string> SearchRemote()

{

JsonSerializer serializer = new JsonSerializer();

string URL = "http://207.180.203.16/v2/public/flights?key=899238-c80304&flightIata=" + flightNumber;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(URL);

request.ContentType = "application/json; charset=utf-8";

HttpWebResponse response = request.GetResponse() as HttpWebResponse;

using (Stream responseStream = response.GetResponseStream())

using (StreamReader reader = new StreamReader(responseStream, Encoding.UTF8))

{

string jsonString = reader.ReadToEnd();

if (jsonString.Contains(noFlightMessage))

{

await Xamarin.Forms.Application.Current.MainPage.Navigation.PushModalAsync(new NoFlightView());

}

else

{

JArray jsonArray = JArray.Parse(jsonString);

JObject jsonObject = JObject.Parse(jsonArray[0].ToString());

flightInfo = JsonConvert.DeserializeObject<FlightInformation>(jsonObject.ToString());

await Xamarin.Forms.Application.Current.MainPage.Navigation.PushModalAsync(new FlightInformationView());

}

}

return "";

В зависимост от проверката дали има такъв полет приложението навигира кой страници да се покажат (View-та). Приложението се състои от четири на брой View-та.

Първото е така наречено Home View, при стартиране на приложението се появява.Код на XAML файл изграждащ визуалната среда на Home View.

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:local="clr-namespace:Flight"

x:Class="Flight.MainPage">

<StackLayout>

<Image Source="my.png"/>

<Entry Placeholder="Въведете номера на полета пример (W8oIw)"

HorizontalOptions="CenterAndExpand"

FontSize="11"

Text="{Binding FlightNumber}"/>

<Button Text="Търси"

VerticalOptions="Fill"

Command="{Binding SearchCommand}"/>

</StackLayout>

</ContentPage>

В зависимост от проверката дали има такъв полет приложението навигира следващите две View-та. Ако търсения полет не съществува се показва следващото View .

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Flight.Views.NoFlightView">

<ContentPage.Content>

<StackLayout>

<Image Source="no.png"/>

<Label Text="Няма информация за полет с този номер!"

HorizontalTextAlignment="Center"/>

<Button Text="Повторно търсене"

VerticalOptions="Fill"

Command="{Binding BackCommand}"/>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Ако търсения от нас полет съществува в базата на информационната система то тогава приложението отваря следващото View , което е с данните за полета.

<ContentPage.Content>

<StackLayout>

<Label Text="{Binding FlightInfo.аircraft.RegNumber, StringFormat='Редистационен номер: {0:N}'}"

HorizontalTextAlignment="Center" />

<Label Text="{Binding FlightInfo.flight.IataNumber, StringFormat='Номер: {0:N}'}"

HorizontalTextAlignment="Center" />

<Label Text="{Binding FlightInfo.geography.Latitude, StringFormat='Географска ширина: {0:N}'}"

HorizontalTextAlignment="Center" />

<Label Text="{Binding FlightInfo.geography.Longitude, StringFormat=' Географска дължина: {0:N}'}"

HorizontalTextAlignment="Center" />

<Label Text="{Binding FlightInfo.geography.Altitude, StringFormat=' Надморската височина: {0:N}'}"

HorizontalTextAlignment="Center" />

<Label Text="{Binding FlightInfo.speed.Horizontal, StringFormat='Скорост: {0:N}'}"

HorizontalTextAlignment="Center" />

<Button Text="Търси нов полет"

VerticalOptions="Fill"

Command="{Binding NewSearchCommand}"/>

<Button Text="Виж на карта"

VerticalOptions="Fill"

Command="{Binding MapCommand}"/>

</StackLayout>

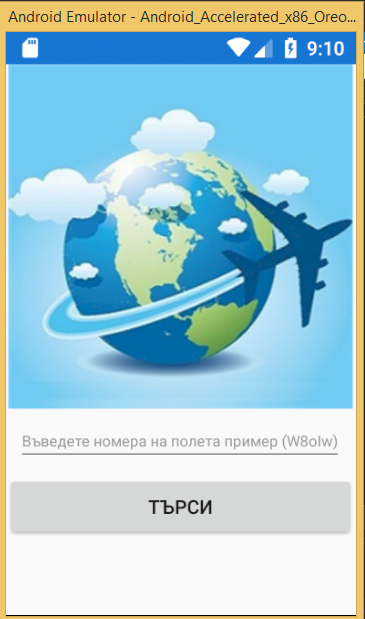
</ContentPage.Content>

</ContentPage>

За визуализация на полета се използва Гоогле мапс. Всяко мобилно устройство с операционна система Андроид разполага с инсталирано от Гоогле приложение Гоогле мапс , което е удобно бързо и лесно за използване.

Глава 4 – Функционално тестване;

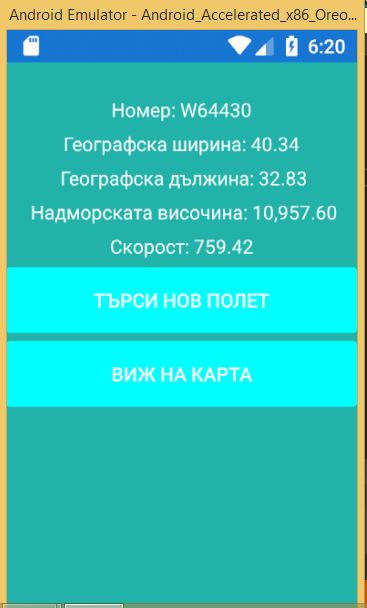
Както се запознахме мобилното приложение се състои от общо четири на брой View-та. При стартиране за мобилното приложение за следене да полети в реално време се появява началната страница фиг. Както се вижда на снимката по-долу потребителя трябва да въведе номер на търсения от него полет.



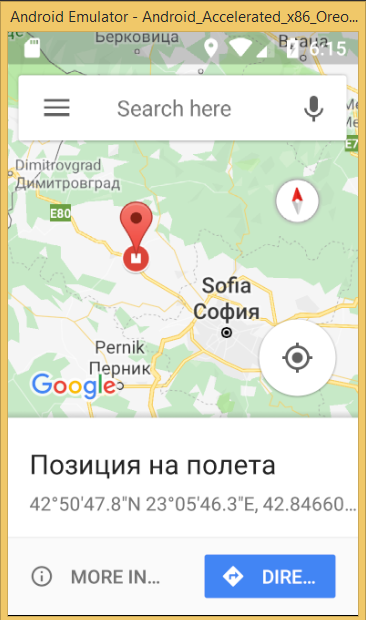
След въвеждането на номера на потребителя и натискането на бутона търси, приложението проверява дали самолет с номера на полет въведен от потребителя съществува. Ако не съществува тогава се отваря следващия прозорец, който указва на потребителя че няма информация за търсения полет.фиг. Както се вижда на снимката по-долу имаме бутон за повторно търсене което ни отвежда на началната страница за повторно търсене.



Ако търсения полет съществува се отваря прозорец с данните на полета в реално време които са номер, географска ширина и дължина, надморска височина и скорост.



Потребителя има две опции за продължение и те са търсене на нов полет или да види точното местонахождение на търсения полет. Ако избере ново търсене се връща на началния прозорец. Ако избере да види самолета на картата се отваря Гоогле мапс и показва точното къде са намира самолета в дадения момент.



Приложението използва услугата на Гоогле (Гоогле мапс) чрез нея след като натиснем бутона „покажи на картата“ се показва картата и на него е отбелязано къде точно се намира дадения полет.

Глава 5 – Приложимост на дипломната работа;

Приложението както се запознахме е разработено да работи на устройства с операционна система Андроид. Приложението може да бъде качено в магазина за приложения на Андроид (Google Play).

Google Play е главното приложение, инсталирано на устройствата, използващи Android. Андроид притежава изобилие от приложения, които потребителят може да изтегли от магазина, който му позволява да ги инсталира, ъпгрейдва и премахва от устройството си.

Броят на новосъздадените приложения продължава да нараства със страхотна скорост. Всяка седмица в Google Play излиза нещо иновативно, което впоследствие бива използвано от милиони хора. Така магазинът за приложения на Google увеличава популярността , Android е най-използваната система за мобилни устройства в света за момента.

Google Play позволява на потребителите да използват приложения, публикувани от Google и някои софтуерни разработчици. През юли 2013 приложенията достъпни за Android в Google Play били около 1 милион, а изтеглените такива към същия момент приближавали 50 милиарда.

Приложението е леко и бързо, което го прави достъпно за голям брой потребители и които искат да разберат данни на търсения от тях полет в реално време, и къде се намира на картата.

Интерфейса е изчистен и максимално опростен за да може да се работи лесно и бързо с него.

Глава 6 – Икономическа оценка на резултатите и техническа ефективност;

1.Заключение

Мобилното приложение реализира най-важната и основна функционалност за едно приложение от подобен тип, и то търсене на полет по номер на полета.

Към момента мобилното приложение е готово за употреба. Част от бъдещо развитие на продукта включва:

* Подобряване на потребителския интерфейс
* Повече информация в реално време.
* Търсене по маршрут
* Търсене на летища.
* Търсене на авиокомпании.

Приложението може да се използва от всеки с мобилно устройство с Android операционна система. Цели да се използва от потребители на всякаква възраст които имат желание да получат информация на полет в реално време било то техния или на приятел или близък човек.

2.Извод

В основата на разработеното от мен мобилно приложение стои .Net Framework, програмният език C# и свързаната с него платформа за разработване на мобилни приложения Xamarin.

Избрах да използвам тази технология, тъй като тя дава възможност за разработка на мултиплатформени мобилни приложения. Xamarin e сравнително нова технология, която се развива с бързи темпове и добива широка популярност. Тя дава възможност за използване на всички най-нови технологии и архитектурни модели, като гореспоменатите XAML, MVVM, връзка с бази данни и други. Също така платформата дава възможност за изполване на така наречените контроли (Xamarin Forms), което улеснява работата и създаването на потребителския интерфейс.

За по-добро оформление на проекта, бъдещи промени, преизползване разделяне на връзките между контролите от потребителския интерфейс и тяхната логика избрах да използвам шаблона MVVM. Той улесни работата по време на развитие на мобилното приложение и ще улесни бъдещи разработчици да разберат по-бързо и лесно приложението.

Източници:

[1] Наков, Светлин, Програмиране за .NET Framework, 2005, издателство „Фабер“

[2] Microsoft, Introduction to the C# Language and the .NET Framework, <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>

[3] Microsoft, Xamarin Platform, https://docs.microsoft.com/en-us/xamarin/

[4] Microsoft, Xamarin Forms, <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/>

[5]Microsoft, Xamarin Android<https://docs.microsoft.com/en-us/xamarin/android/>

[6]Microsoft, Learn about mobile development with Xamarin, <https://docs.microsoft.com/en-us/visualstudio/cross-platform/learn-about-mobile-development-with-xamarin?view=vs-2017>

[7] MVVM Tutorial, <https://www.tutorialspoint.com/mvvm>

[8] C# Serialization, https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/

[9] Xaml overview, https://docs.microsoft.com/enus/dotnet/framework/wpf/advanced/xaml-overview-wpf

**Приложения:**

**MODELS:**

**Flight.cs**

namespace Flight.Models

{

public class Flight

{

public string IataNumber { get; set; }

public string IcaoNumber { get; set; }

public string Number { get; set; }

}

}

**Geography.cs**

namespace Flight.Models

{

public class Geography

{

public double Latitude { get; set; }

public double Longitude { get; set; }

public double Altitude { get; set; }

public double Direction { get; set; }

}

}

**FlightInformation.cs**

namespace Flight.Models

{

public class FlightInformation

{

public Geography geography { get; set; }

public Speed speed { get; set; }

public Departure departure { get; set; }

public Arrival arrival { get; set; }

public Aircraft aircraft { get; set; }

public Airline airline { get; set; }

public Flight flight { get; set; }

public System system { get; set; }

public string status { get; set; }

}

}

**MainPageViewModel.cs**

namespace Flight.ViewModels

{

public class MainPageViewModel : INotifyPropertyChanged

{

public static string flightNumber = null;

private readonly string noFlightMessage = "No Record Found or Flight not currently detected by receivers.";

public static FlightInformation flightInfo = new FlightInformation();

private ICommand searchCommand;

public event PropertyChangedEventHandler PropertyChanged;

public MainPageViewModel()

{

this.SearchCommand = new RelayCommand(async (object obj) => await SearchFlight(obj));

}

public ICommand SearchCommand

{

get

{

return this.searchCommand;

}

set

{

searchCommand = value;

}

}

public string FlightNumber

{

get

{

return flightNumber;

}

set

{

flightNumber = value;

NotifyPropertyChanged();

}

}

public async Task SearchFlight(object obj)

{

var responseString = await SearchRemote();

}

public async Task<string> SearchRemote()

{

JsonSerializer serializer = new JsonSerializer();

string URL = "http://207.180.203.16/v2/public/flights?key=899238-c80304&flightIata=" + flightNumber;

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(URL);

request.ContentType = "application/json; charset=utf-8";

HttpWebResponse response = request.GetResponse() as HttpWebResponse;

using (Stream responseStream = response.GetResponseStream())

using (StreamReader reader = new StreamReader(responseStream, Encoding.UTF8))

{

string jsonString = reader.ReadToEnd();

if (jsonString.Contains(noFlightMessage))

{

await Xamarin.Forms.Application.Current.MainPage.Navigation.PushModalAsync(new NoFlightView());

}

else

{

JArray jsonArray = JArray.Parse(jsonString);

JObject jsonObject = JObject.Parse(jsonArray[0].ToString());

flightInfo = JsonConvert.DeserializeObject<FlightInformation>(jsonObject.ToString());

await Xamarin.Forms.Application.Current.MainPage.Navigation.PushModalAsync(new FlightInformationView());

}

}

return "";

}

private void NotifyPropertyChanged([CallerMemberName]String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

}

**NoFlightViewModel.cs**

namespace Flight.ViewModels

{

public class NoFlightViewModel: INotifyPropertyChanged

{

private ICommand backCommand;

public event PropertyChangedEventHandler PropertyChanged;

public NoFlightViewModel()

{

this.BackCommand = new RelayCommand((object obj) => GoToMainPage(obj));

}

public ICommand BackCommand

{

get

{

return this.backCommand;

}

set

{

this.backCommand = value;

}

}

public void GoToMainPage(object obj)

{

Xamarin.Forms.Application.Current.MainPage.Navigation.PushModalAsync(new MainPage());

}

private void NotifyPropertyChanged([CallerMemberName]String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

}

**FlightInformationViewModel.cs**

namespace Flight.ViewModels

{

public class FlightInformationViewModel

{

private ICommand newSearchCommand;

private ICommand mapCommand;

private FlightInformation flightInfo = MainPageViewModel.flightInfo;

public event PropertyChangedEventHandler PropertyChanged;

public FlightInformationViewModel()

{

this.MapCommand = new RelayCommand((object obj) => GoToMapPageAsync(obj));

this.NewSearchCommand = new RelayCommand((object obj) => NewSearch(obj));

}

public FlightInformation FlightInfo

{

get

{

return this.flightInfo;

}

set

{

this.FlightInfo = MainPageViewModel.flightInfo;

NotifyPropertyChanged();

}

}

public ICommand NewSearchCommand

{

get

{

return this.newSearchCommand;

}

set

{

this.newSearchCommand = value;

}

}

public ICommand MapCommand

{

get

{

return this.mapCommand;

}

set

{

this.mapCommand = value;

}

}

public async void GoToMapPageAsync(object obj)

{

var location = new Location(MainPageViewModel.flightInfo.geography.Latitude, MainPageViewModel.flightInfo.geography.Longitude);

var options = new MapLaunchOptions { Name = "Позиция на полета"};

await Map.OpenAsync(location, options);

}

public void NewSearch(object obj)

{

Xamarin.Forms.Application.Current.MainPage.Navigation.PushModalAsync(new MainPage());

}

private void NotifyPropertyChanged([CallerMemberName]String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}