math_2020_083

Міністерство освіти і науки України

Чернівецький національний університет

Імені Юрія Федьковича

Факультет математики та інформатики

(повна назва інституту/факультету)

Кафедра математичного моделювання

(повна назва кафедри)

Розробка корпоративної системи взаємодії викладачів та студентів факультету математики та інформатики

Виконав:
Студент 6 курсу, 607 групи
Спеціальності 124 – Системний аналіз
(назва спеціальності)
Сучеван Микола Штефанович
(прізвище, ім’я та по-батькові)
Керівник к.ф.-м.н. асистент Іліка С.А.
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:
Протокол засідання кафедри №7
від « 15 » грудня 2020 р.
зав. кафедри _____проф. Черевко І.М.

Чернівці 2020

АНОТАЦІЯ

Магістерська робота присвячена розробці мобільного додатку на платформі Xamarin. Додаток призначений для легкої взаємодії студентів та викладачів факультету математики та інформатики.

ЗМІСТ

ВСТУП

Сьогодення характеризується активним розвитком процесів глобалізації. Інтернет стає головним сегментом спілкування людей. Він став віртуальною територією, де, незважаючи на відстань, завжди можна легко «зустрітися» та поспілкуватися. Інтернет виступає як специфічний засіб спілкування.

Різноманітність соціальних мереж та корпоративних соціальних мереж дедалі зростає. З кожним днем, зробити вибір між ними важче. Кожен використовує те, що йому до вподоби.

На факультеті математики та інформатики студенти та викладачі використовують різні соціальні мережі, або додатки для обміну повідомленнями, для опублікування новин або розповсюдження інформації.

Тому виникла ідея створення мобільного додатку за допомогою якого, студенти та викладачі зможуть взаємодіяти дистанційно, на базі одного мобільного додаткку: проводити анонімні опитування, опубліковувати новини та обмінюватись повідомленнями.

Мета магістерської роботи – реалізувати додаток за допомогою якого, студенти та викладачі факультету математики та інформатики зможуть взаємодіяти дистанційно: проводити анонімні опитування, опубліковувати новини та обмінюватись повідомленнями.

Переді мною було поставлено ряд завдань:

1. Проектування та створення бази даних.
2. Розробка клієнт серверної частини проекту.
3. Створення мобільного додатку на платформі Xamarin.
4. Тестування додатку

При створенні проекту були використані сучасні технології:

- мова програмування C# та Entity Framework;
- мова розмітки XAML;
- Management studio – для створення та адміністрування бази даних;
- для розробки використовувалось середовище Visual Studio 2019 Community;
- сервіс Trello – для організації часу та керування поставлених задач;
- сервіс GitHub – для керування версіями додатку та зберігання резервної копії проекту;

Практичне значення дослідження:

Створений додаток може бути використаний студентами та викладачами факультету математики та інформатики для оголошення різних подій, що дуже актуально коли потрібно зробити якесь оголошення, тому що всім користувачам прийде сповіщення. Додаток також дозволяє швидко переглядати останні новини сайту. Слід зазначити, що зареєструватись зможуть тільки ті користувачі, які мають корпоративну пошту ЧНУ. Це дозволить викладачам та органам студентського самоврядування легко розширювати потрібну інформацію, або розмістити опитування.

§1. Опис використаних технологій.

1.1. Мова програмування C#

C# (читається «*сі шарп*») – об’єктно-орієнтована мова програмування, яка походить від сімейства мов програмування C. Дана мова була розроблена компанією Microsoft. Вона зручна тим, що програмісти які працювали з такими мовами C, C++, Java та JavaScript можуть з легкістю освоїти синтаксис мови програмування C#.

1998–2001 роки були визначними для даної мови, оскільки тоді була створена перша версія. *Андрес Гейлсберг* та *Скот Вільтаумот* разом з групою інженерів Microsoft зробили C# основною мовою програмування платформи **Microsoft.Net.**

C# вважається однією з найкращих мов програмування, оскільки вона була розроблена на практичному досвіді використання інших мов програмування, таких як: C, C++, Modula, ObjectPascal, Java. Множинне успадкування класів (яке використовується у мові C++) та деякі проблематичні моделі, що використовувались в інших мовах програмування були виключені.

Мова програмування C# трактується як наступне покоління розвитку C++, а символ # – символізує «++++». Дієз «♯» – англійською «шарп», який очевидно фігурує у назві, але через його відсутність на клавіатурі, було вирішено використовувати знак «#». Помилкою не буде вважатися, якщо у назві даної мови програмування буде використано будь-який з цих символів.

Очевидно, що мова програмування C# дуже схоже на Java, це можна побачити розглядаючи основні поняття цих мов та за їх синтаксисом.

Як було сказано вище, у 1998 році розпочалась розробка мови програмування C#, а саме це відбулось у грудні. Ця мова готувалася до випуску разом продуктами групи Millenium. Проект розроблявся, як аналог мови

програмування Java від компанії Oracle та мав назву COOL(**C**-style **O**bject **O**riented **L**anguage). Основною мовою платформи Microsoft.Net framework стала мова програмування C# у 2000 році. Також перша загальнодоступна бета-версія з'явилася в цьому ж році.

У 2002 році була випущена перша фінальна версія мови програмування C#. В цьому ж році було випущене середовище інтегрованої розробки програмного забезпечення **Visual Studio .Net**.[7]

Аналогічно до мови програмування Java, C# має такі концепції:

- віртуальна машина – платформа .Net виконує програму подібно до віртуальної машини від Java;
- байт-код – програмний код, який компілюється в проміжну мову MSIL(Microsoft Intermediate Language), а вже потім перетворюється в машинну мову, в залежності від платформи на якій запускається програма;
- керований код – оскільки програми написані на C#, виконуються виключно у віртуальному середовищі CLR(Common Language Runtime), таким чином, можемо контролювати виконання програми у будь-який момент та зупинити її коли ми хочемо, а також є можливість контролювання використання пам'яті програми, за необхідності – збільшувати, або видаляти частини пам'яті, які використовувати програма.

Метою C# була розробка мови програмування, яку не тільки легко вивчити, а, яка буде підтримувати сучасні, функціональні можливості для всіх видів розробки програмного забезпечення. C# надає функціональність для підтримки сучасного розвитку програмного забезпечення. C# підтримує потреби в розробці веб-програм, мобільних пристроїв та додатків. Деякі з сучасних функцій мови програмування, які підтримує C#, - це загальні типи,

типи змінних, автоматична ініціалізація типів і колекцій, лямбда-вирази, динамічне програмування, асинхронне програмування, кортежі, узгодження шаблонів, розширена налагодження та обробка винятків тощо.

C# - це безпечна для типів даних мова. Вона не дозволяє перетворювати типи, що може призвести до втрати даних або інших проблем. C# дозволяє розробникам писати безпечний код. [10]

1.2. Microsoft.Net framework

Середовище для підтримки, розробки і виконання розподілених додатків, що базуються на складових (елементи управління) є .Net framework. Особливістю цієї технології є те, що можна розробляти програми на різних мовах програмування, використовуючи цей фреймворк, які його підтримують.

.Net framework забезпечує:

- сумісне використання різних мов програмування;
- безпеку та міграцію програм;
- загальну модель програми на базі платформ Windows.

Якщо дивитись на .Net framework з точки зору програмування, то він має дві основні частини:

- середовище CLR(Common Language Runtime);
- бібліотека базових класів.

CLR, яке є загальномовним середовищем, може розв’язувати різні задачі автоматичного виявлення типів .Net, завантажувати та керувати ними.

Керування пам’яттю, обслуговування додатку, обробка потоків також здійснює середовище CLR. Воно реалізує численні перевірки пов’язані з безпекою, що є дуже важливим у розробці.

Потоки, реалізація баз даних, файловий ввід-вивід – ці примітиви містяться у бібліотеці базових класів.

Виконання коду .Net керується за допомогою виконавчого середовища – CLR. Коли компілюється програма на C#(або на будь-якій іншій мові) генерується файл з псевдокодом (а не виконавчим файлом, як це використовувалось давніше). Microsoft Intermediate Language (MSIL), або Common Intermediate Language (CIL), являються цими псевдокодами, які є проміжною мовою Microsoft.

CLR має таку ціль – перетворення проміжного коду MSIL у виконавчий код під час виконання програми.

Будь-який скомпільований в псевдокод MSIL додаток, має можливість виконуватись в будь-якому середовищі, що підтримує реалізацію CLR. Таким чином .Net framework стає дуже гнучким для використання.

Наступним етапом є те, що псевдокод за допомогою JIT(Just in time)-компілятора у виконавчий код. Така компіляція є компіляцією в процесі виконання програми. За місце розміщення збірки (Assembly) відповідає виконавче середовище CLR. У збірці розташований запитуваний тип. Він зчитується з метаданих двійкового файлу (*.dll, або *.exe). Далі CLR розташовує в пам’яті зчитаний зі збірки тип. Потім CIL-код, який перетворений за допомогою CLR у інструкції. Ці інструкції автоматично налаштовуються під конкретну платформу, все залежить від персонального комп’ютера та операційної системи. Також, на цьому ж етапі здійснюються перевірки, орієнтовані на безпеку програми. Фінальним етапом є виконання програмного коду.

На перших версіях .Net мова псевдокоду називалась Microsoft Intermediate Language (MSIL). На останніх версіях .Net, ця назва була перейменована на

Common Intermediate Language (CIL). Дані абревіатури хоч і різні, але визначають одне поняття та виконують одну функцію.

Після компіляції програми на різних мовах програмування, що підтримують платформу .Net framework створюється проміжна мова CIL (або MSIL). Проміжна мова CIL (або MSIL) є псевдокодом та визначає такий набір інструкцій:

- переносність на різні платформи;
- не мають залежності від типу процесора.

Виходячи з вище сказаного, можемо зробити висновок, що проміжна мова псевдокоду MSIL або CIL, являються мовою переносного асемблеру.

Код програми на таких мовах як: C#, C++/CLI, Visual Basic ті інші, підтримують технологію .Net.

Розробка відбувається в такому порядку: програміст створює проект в середовищі інтегрованої розробки програмного забезпечення, наприклад Visual Studio. Після опису свого коду, компілятор генерує збірку, тобто файл, який має метадані, маніфест та основне CIL-інструкції.

Механізм виконання .Net запускається після виконання програми на персональному комп'ютері. .Net framework, очевидно має бути встановленим на пристрої виконання програми. На пристрої маємо можливість встановлювати різні версії .Net, або хоча б одну.

Бібліотеки базових класів зазвичай використовуються у програмному коді. Вони під'єднуються за допомогою завантажувача класів.

Компіляцію збірки з прив'язкою апаратної та програмної платформи пристрою, на якому здійснюється запуск програми виконується за допомогою JIT(Just in time)-компілятора.

Після цих маніпуляцій дана програма буде виконана.

Середовище інтегрованої розробки програмного забезпечення Visuual Studio .Net дає можливість використовувати технологію .Net, але маємо враховувати мову програмування, яка підтримує дану технологію, серед них такі: C#, Visual Basic, C++/CLI, F#, J#.

Для того, щоб мати можливість використовувати у своїх проектах .Net технологію, потрібно встановити такі програмні забезпечення: Microsoft.Net framework, Software Development Kit (SDK), або Microsoft Visual Studio.[7]

1.3. SQL server

SQL Server - це реляційна система управління базами даних (СУБД), розроблена та представлена корпорацією Майкрософт.

Подібно до іншого програмного забезпечення СУБД, SQL Server побудований поверх SQL. SQL - це стандарт ANSI (Американський національний інститут стандартів), але існує багато різних версій мови SQL. SQL – мова структурованих запитів, яка є комп'ютерною мовою для зберігання, маніпулювання та отримання даних, що зберігаються в реляційній базі даних. SQL Server пов'язаний з Transact-SQL, або T-SQL, реалізацією Microsoft SQL, яка додає набір власних конструкцій програмування.

У 1988 році Microsoft випустила свою першу версію SQL Server. Він був розроблений для платформи OS / 2 і розроблений спільно Microsoft та Sybase. На початку 90-х років Microsoft почала розробляти нову версію SQL Server для платформи NT. Поки він розроблявся, Microsoft вирішила, що SQL Server повинен бути тісно поєднаний з операційною системою NT. У 1992 році Microsoft взяла на себе основну відповідальність за майбутнє SQL Server для NT. У 1993 році були випущені Windows NT 3.1 та SQL Server 4.2 для NT. Філософія корпорації Майкрософт у поєднанні високопродуктивної бази даних із простим у користуванні інтерфейсом виявилася дуже успішною. Microsoft

швидко стала другим за популярністю постачальником висококласних програм для реляційних баз даних.

SQL Server працює виключно в середовищі Windows понад 20 років. У 2016 році Microsoft зробила це доступним на Linux.

Коли виконується команда SQL для SQL Server, система визначає найкращий спосіб виконати ваш запит, а механізм SQL з'ясовує, як інтерпретувати завдання. У процес входять різні компоненти. Ці компоненти – це диспетчер запитів, механізми оптимізації, класичний механізм запитів та механізм запитів SQL тощо. Класичний механізм запитів обробляє всі запити, не пов'язані з SQL, але механізм запитів SQL не обробляє логічні файли.

Коли користувачам потрібно отримати дані, вони хочуть їх так швидко, наскільки програмне забезпечення може їм їх надати. Microsoft SQL Server включає кілька варіантів для ввімкнення швидших запитів. Оптимізовані пам'яті таблиці тепер підтримують ще швидші робочі навантаження обробки онлайн-транзакцій (OLTP) з кращою пропускною здатністю в результаті нових паралелізованих операцій.

Кожна база даних SQL Server складається з декількох об'єктів, які використовуються для зберігання, організації та обробки даних. Об'єкти бази даних SQL Server – це таблиці, індекси, подання, обмеження, правила, за замовчуванням, тригери, процедури та типи даних.

Таблиця – це основний об'єкт, який зберігає всі записи, що належать до бази даних (файли таблиці мають розширення .mdf). SQL Server має два типи таблиць – системну та спеціальну. Системні таблиці зберігають інформацію про SQL Server та його об'єкти, а таблиці користувачів зберігають інформацію з первинних документів. Імена всіх системних таблиць починаються з префікса sys. Таблиці включають файл транзакцій (Mf), який створюється автоматично при створенні бази даних і призначений для забезпечення цілісності та

відновлення бази даних у разі помилок (якщо база даних змінена, журнал транзакцій зберігає нові та старі значення рядків таблиці).

Індекси – це файли з розширенням idx, які використовуються для скорочення часу на виконання операцій пошуку та отримання даних з таблиць (перелік файлів індексу визначається користувачем). Індексні файли створюються з номерів записів, розташованих у тому порядку, в якому вони були б, якби вони були відсортовані за певними полями.

Представлення – це SQL-інструкція Select, на основі якої вибираються дані з однієї або декількох таблиць та формується вихідна таблиця. Найчастіше подання використовується для отримання даних з безлічі рядків або стовпців з таблиць, об'єднання стовпців з різних таблиць та обчислення підсумків на основі даних одного або декількох стовпців.

Обмеження гарантують цілісність даних для таблиць, їх зазвичай додають користувачі до таблиці після її створення, і їх можна визначити на рівні стовпців або на рівні таблиці. SQL Server підтримує п’ять типів обмежень цілісності:

1. Первинний ключ (обмеження на первинний ключ) – гарантує, що всі рядки таблиці будуть унікальним ключем, не рівним NULL. Використання обмеження первинного ключа, серед іншого, створює унікальний індекс у таблиці.

2. Зовнішній ключ – пов'язує один або кілька стовпців у таблиці з первинним ключем і гарантує, що вказаний зв'язок існує між двома таблицями.

3. Унікальний – запобігає появі повторюваних значень у будь-якому стовпці. 4. Перевірка (обмеження значення) - забезпечує контроль над значеннями, які можна ввести в стовпець таблиці. Наприклад, можна встановити обмеження для перевірки введення даних у стовпці таблиці від 1 до 100.

5. Не нульовий – використовується для забезпечення того, що стовпець не буде нульовим.

Правила – це, по суті, ті самі обмеження, але більш детальні. Правила базуються на списку значень та логічних виразів (на практиці вони застосовуються рідко).

Значення за замовчуванням – встановлює значення, яке автоматично записується в комірку стовпця таблиці, якщо в це поле не введено значення.

Тригер – це процедура, яка виконується автоматично при зміні таблиці за допомогою інструкцій Оновити, Вставити, Видалити.

Процедура являє собою серію інструкцій Transact-SQLt, які компілюються в спеціальний формат під час її створення, з подальшим використанням для реалізації різних функцій адміністрування баз даних, обробки даних тощо.

Типи даних - визначають тип інформації, яку можна зберігати в певному стовпці таблиці. Типи даних поділяються на числові цілі, числа з плаваючою комою (дробові), текст, дату та час, спеціальні.[11]

1.4. Xamarin

Xamarin - це платформа з відкритим кодом для створення сучасних та продуктивних додатків для iOS, Android та Windows за допомогою .NET. Xamarin надає розробнику інструменти, які можуть допомогти їм у створенні кросплатформенних мобільних додатків. Додатки можуть мати всі власні функції, а також спільно використовувати спільну базу коду одночасно. Згідно зі статистикою Xamarin, понад 15 000 компаній покладаються на свої інструменти, і список включає багато відомих імен.

Інструменти Xamarin доступні для завантаження за допомогою Visual Studio, і можливо безпосередньо створювати програми для Android, iOS та Windows із самої Visual Studio. Найбільш поширені коди написані на C #. Тому

вам не потрібно вивчати Java, Objective-C або Swift для створення додатків, знаючи C#. Якщо програміст новачок, то, використовуючи шлях Xamarin замість звичного навчального процесу, він може навчитися розробляти програми для більш ніж однієї платформи.

Xamarin дозволяє розробникам розподіляти в середньому 90% своїх програм між платформами. Цей шаблон дозволяє розробникам писати всю свою бізнес-логіку однією мовою (або повторно використовувати існуючий код програми), але досягати природних характеристик, вигляду та відчуття на кожній платформі. Програми Xamarin можна писати на ПК або Mac і компілювати в рідні пакети програм, такі як .apk-файл на Android або .ipa-файл на iOS.

Широкий діапазон функцій, що надаються всім набором інструментів, гарантує, що програміст не втратить жодної функціональності. Але в якийсь момент, коли може виникнути бажання копати набагато глибше. Xamarin дозволяє викликати існуючий код, написаний іншими мовами певної платформи, наприклад Java на Android. Але це лише тоді, коли програміст будує щось дуже конкретне, що не може бути реалізоване на різних платформах.

Xamarin також підтримує носимі пристрої. Він надає можливість створити власні програми для Android Wear та Apple Watch. Магазин компонентів Xamarin дозволяє вам додати більше функціональних можливостей до своїх програм, завантажуючи прості плагіни. Можна легко інтегрувати свою програму з найпопулярнішими серверними системами, такими як Microsoft Azure, Parse та іншими. Також можливо додати популярні методи автентифікації. Додаткові модулі також доступні, щоб додати підтримку виставлення рахунків та інші функції. Найпопулярніші плагіни є кросплатформними, але також доступні певні плагіни, такі як плагін підтримки платіжних програм Google Play.

То що в основному робить Xamarin, щоб програміст міг писати загальний код для різних платформ? На самому кореневому рівні Xamarin перетворив весь існуючий Android і iOS SDK на C#, щоб програміст міг програмувати більш звичною мовою. І оскільки можна використовувати C# для програмування на цих платформах, вам потрібно пам'ятати менше синтаксису. За допомогою інструментів Xamarin можна отримати доступ до будь-якого iOS або Android API на C #.

Крім того, Xamarin пропонує прив'язувальні проекти, які дозволяють прив'язувати власні бібліотеки Objective-C та Java, використовуючи декларативний синтаксис. Сучасні мовні конструкції - програми Xamarin написані на C #, сучасній мові, що включає значні вдосконалення в порівнянні з Objective-C та Java, такі як динамічні мовні функції, функціональні конструкції, такі як лямбда, LINQ, паралельне програмування, дженерики тощо.

Xamarin.Forms - це середовище інтерфейсу з відкритим кодом. Xamarin.Forms дозволяє розробникам створювати програми Xamarin.iOS, Xamarin.Android та Windows з однієї спільної кодової бази. Xamarin.Forms дозволяє розробникам створювати користувацькі інтерфейси в XAML з кодом позаду в C #. Ці користувацькі інтерфейси відображаються як ефективні власні елементи управління на кожній платформі.[12]

1.5. Xaml

XAML - це нова мова розмітки, розроблена корпорацією Майкрософт для написання користувацьких інтерфейсів для керованих додатків наступного покоління. XAML - це мова для побудови користувацьких інтерфейсів для Windows та мобільних додатків, які використовують Windows Presentation Foundation (WPF), UWP та Xamarin Forms. Призначення XAML просте - створити користувацькі інтерфейси за допомогою мови розмітки, схожої на XML. Здебільшого, використовується дизайнер для створення XAML, але

можна вільно маніпулювати XAML вручну. XAML використовує формат XML для елементів та атрибутів. Кожен елемент у XAML представляє об'єкт, який є екземпляром типу. Сфера застосування типу (класу, перерахування тощо) визначається як простір імен, який фізично знаходиться у збірці (DLL) бібліотеки .NET Framework.

Подібно до XML, синтаксис елемента XAML завжди починається з відкритої кутової дужки (<) і закінчується закритою кутовою дужкою (>). Кожен тег елемента також має початковий та кінцевий теги. Наприклад, об'єкт Button представлений елементом <Button>. Наступний фрагмент коду представляє елемент об’єкта Button:

<Button> </Button>

Крім того, є можливість використовувати самозакриваючий тег <Button/>. Елемент об'єкта в XAML представляє тип. Тип може бути елементом управління, класом або іншими об'єктами, визначеними в бібліотеці фреймворку.[13]

§2. Програмна частина

2.1. Опис предметної області

Предметною областю називається фрагмент реальності, який описується чи моделюється з допомогою бази даних і його додатків. У предметної області виділяються інформаційні об'єкти – об'єкти реального світу, процеси, системи, основні поняття та дані які зберігаються у базі даних.

Електронні мережі, через нові засоби комунікації, соціальні мережі та інші цифрові платформи з метою обміну інформацією змінили суспільство до незворотної міри. Цей розвиток давно потрапив у компанії. Працівники, що займаються знаннями, все частіше вимагають від своєї компанії та IT-відділу електронних мережних колег та проектних команд та надання єдиних платформ для синхронного огляду та обробки спільних завдань проекту. Корпоративна мережа є дуже корисною для великих підприємств. Вона надає можливість спілкування, обговорення проблем, розбиттям на команди або групи в межах підприємства.

В даній магістерській роботі розроблена база даних для мобільного додатку корпоративної системи факультету математики та інформатики.

Для коректної роботи мобільного додатку, нам потрібно дані користувача та інформація його активності. Кожен користувач характеризується ПІБ, логіном, паролем, корпоративною електронною поштою, датою народження, статусом(викладач або студент) та статусом адміністратора. Користувач має можливість створювати пости та анонімні опитування. Пост повинен містити назву, фотографію, опис, дату опублікування та автора. Опитування описується назвою, датою опублікування та питаннями, які містять текст та відповіді на запитання. Також варто розглянути чати, які характеризуються назвою, фото, користувачами що входять у цей чат та повідомленнями. Повідомлення містять

текст повідомлення, час відправки повідомлення та автора поточного повідомлення.

Виділимо об’єкти даної предметної області.

1. Користувач.
2. Пост.
3. Опитування.
4. Питання.
5. Відповідь.
6. Чат.
7. Повідомлення.

Деякі об'єкти предметної області зв'язані за допомогою зв'язку «багато до багатьох», тому потрібно звести їх до третьої нормальної форми, тобто ввести проміжні таблиці. Таким чином отримаємо такі таблиці:

Answers

- Id (int) – первинний ключ
- Text (ntext)
- QuestionId (int) – зовнішній ключ
- VotesCount (int)

Chats

- Id (int) – первинний ключ
- Name (ntext)
- PhotoPath (ntext)

ChatUsers

- Id (int) – первинний ключ
- UserId (int) – зовнішній ключ
- ChatId (int) – зовнішній ключ

Messages

- Id (int) – первинний ключ
- Text (ntext)
- AuthorId (int) – зовнішній ключ
- SentTime (datetime)
- ChatId (int) – зовнішній ключ

Polls

- Id (int) – первинний ключ
- Name (ntext)
- AuthorId (int) – зовнішній ключ
- PublicationTime (datetime)

PollUser

- Id (int) – первинний ключ
- UserId (int) – зовнішній ключ
- PollId (int) – зовнішній ключ
- Time (datetime)

Posts

- Id (int) – первинний ключ
- Name (ntext)
- Description (ntext)
- PhotoPath (ntext)
- AuthorId (int) – зовнішній ключ
- PublicationTime (datetime)

Question

- Id (int) – первинний ключ
- Text (ntext)
- PollId (int) – зовнішній ключ

Settings

- Id (int) – первинний ключ

- MessageNotification (bit)
- UserId (int) – зовнішній ключ

Users

- Id (int) – первинний ключ
- Login (nvarchar(20))
- Password (nvarchar(20))
- Email (nvarchar(50))
- FirstName (nvarchar(20))
- LastName (nvarchar(20))
- FatherName (nvarchar(20))
- IsAdmin (bit)
- IsTeacher (bit)
- Birthday (date)
- PhotoPath (ntext)

Дана курсова робота повинна реалізувати ряд завдань.

• Створити базу даних.

• Створити таблиці, в яких буде міститись вся інформація, та оптимально розподілити всю інформацію по таблицях.

• Зв’язати таблиці.

Спроектована база даних повинна надавати можливість швидкого та зручного доступу до всіх записів. На базі таблиць можна будувати різноманітні SQL-запити, для різних цілей:

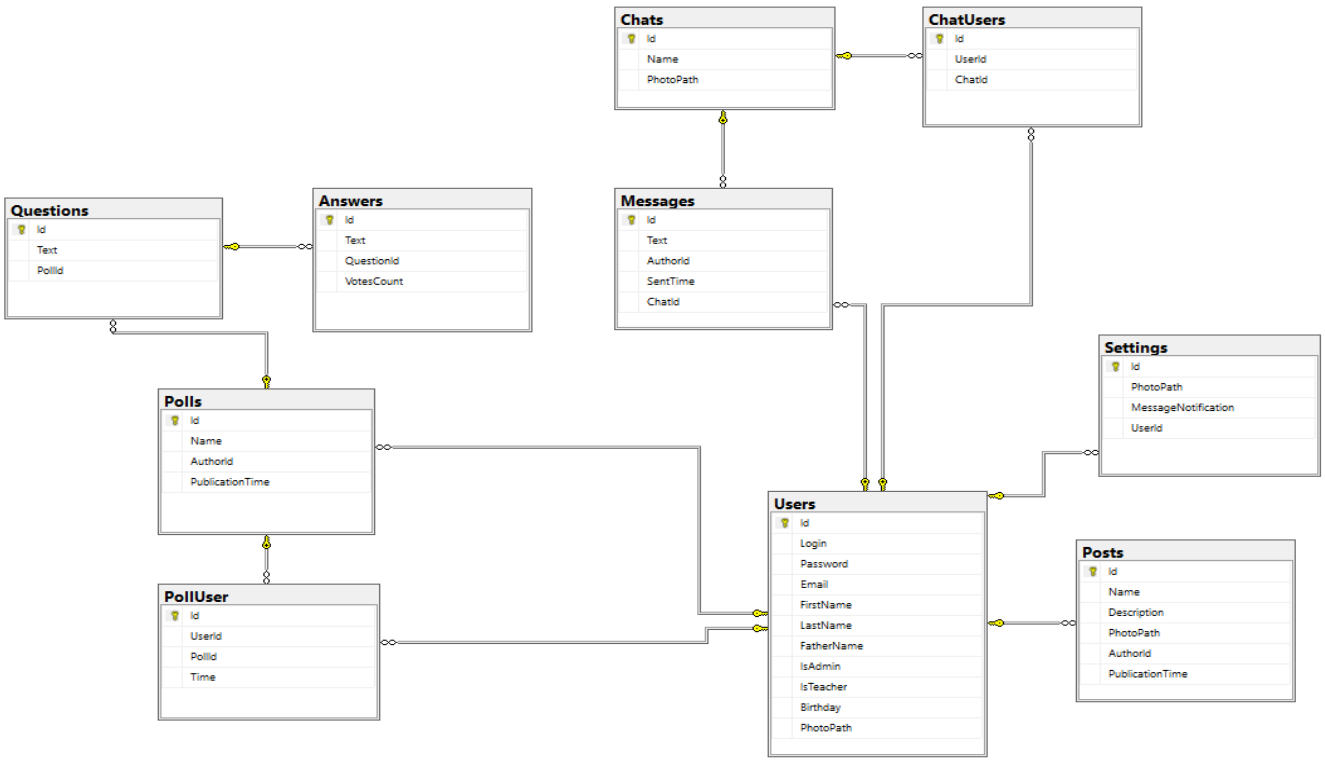
• для пошуку користувачів;

• для створення користувачів;

• для відображення всіх постів та анонімних опитувань;

- для створення постів;
- для створення анонімних опитувань;
- для створення чатів;
- для створення повідомлень;
- для відображення чатів та їх повідомлень;

Схема створеної бази даних:



2.2. Модель життєвого циклу

По суті, життєвий цикл розробки програмного забезпечення є картою для роботи над цифровим рішенням. Іншими словами, модель життєвого циклу – це проект, призначений для команди для створення, обслуговування та виправлення цифрових продуктів. Етапи процесу життєвого циклу розробки програмного забезпечення залежать від розміру проекту та цілей проекту. Кожен крок також містить загальну карту того, як його слід заповнити. У більшості випадків кожна команда розробників створює власний цикл розробки програмного забезпечення або застосовує одну з моделей, яку ми будемо вивчати далі.

Існує безліч різних моделей життєвого циклу розробки програмного забезпечення, які допомагають у різних типах проектів розробки програмного забезпечення, навіть у тих випадках, коли клієнт не має чіткого бачення того, що йому потрібно.

Модель водоспаду – усі етапи слід завершити до початку розробки. Однією з основних передумов моделі «Водоспад» є отримання схвалення на кожному етапі, перш ніж команда зможе перейти до наступного. Цей підхід може бути ефективним для зменшення ризиків у життєвому циклі розробки програмного забезпечення. Тут модель «Водоспад» використовує специфікацію бізнес-вимог, яка допомагає командам оцінювати кожен крок. Хоча деякі компанії з розробки програмного забезпечення все ще пропонують цю модель співпраці, цей тип життєвого циклу розробки програмного забезпечення менш популярний, ніж інші.

Модель V-подібної форми схожа на Водоспад, і її можна розглядати як її продовження. Тому методологічною основою V-подібної моделі є гарантування виконання завдань на одному етапі перед переходом до наступного. Ця модель також розділяє процес розвитку на різні виклики. Ще однією особливістю V-

подібної моделі є постійне тестування, що робить її помітною серед деяких інших моделей життєвого циклу розробки.

Ітераційна модель має безліч циклів розробки програмного забезпечення, які сегментовані на менші цикли. Крім того, ця модель забезпечує надійний старт для програмного продукту за допомогою випробування. Оскільки метод розробки динамічних систем, що використовується в цій моделі, ділить цикл на кілька менших, що дозволяє здійснювати мікроуправління, ця ітераційна модель є одним з найбільш надійних підходів до процесу розробки.

Спіральна модель – це універсальна модель життєвого циклу розробки програмного забезпечення. Подібно до ітеративної моделі, вона підкреслює значення менших циклів у межах більших циклів. Спіральна модель тісно поєднує в собі всі ключові етапи процесу розробки. Ця модель життєвого циклу розробки програмного забезпечення виключає складності будь-якого традиційного життєвого циклу програмного забезпечення. Цей рівень ефективності має свою ціну: розробники витратять більше часу на завдання. Проте, спіральна модель – це один із найкорисніших підходів до поступового вдосконалення продукту серед моделей життєвого циклу розробки програмного забезпечення.

Модель Великого Вибуху взагалі не має вказівок. Ця модель життєвого циклу розробки програмного забезпечення була розроблена, щоб допомогти орієнтуватися в проектах, де клієнт не знає, як виглядатиме кінцевий програмний продукт. Більш конкретно, модель Великого вибуху побудована для проектів, якщо початкова інформація настільки розмита, що сама модель не передбачає певного процесу, за винятком її концепцій або будь-якого планування: команді необхідно розібратися з проектом по ходу. Модель «Великого вибуху» підходить для невеликих зусиль з розробки, невеликих команд розробників, а також може бути придатною для короткострокових експериментів.

Для розробки, було обрано ітераційну модель.

2.3. Web-розробка

Для початку розглянемо концепцію MVC:

Модель–вигляд–контролер (або Модель–представлення–контролер, англ. Model-view-controller, MVC) — це архітектурний шаблон, створений для розробки додатків, зокрема веб-додатків. Як випливає з назви, він складається з трьох основних частин. Традиційний шаблон дизайну програмного забезпечення працює за шаблоном "Вхід – Процес – Вивід", тоді як MVC працює як "Контролер – Модель – Представлення". З появою моделі MVC створення додатка враховує різні аспекти окремо.

- Модель містить чисті дані, пов’язані з додатком. Але модель не має жодної логіки щодо того, як подати дані.

- Представлення використовується для представлення даних моделі користувачеві. Цей елемент стосується того, як пов’язати дані моделі, але не надає жодної логіки щодо того, що це ці дані, або як користувачі можуть використовувати ці дані.

- Контролер знаходиться між моделлю та елементом подання. Він прослуховує всі події та дії, викликані видом, і виконує відповідну реакцію на події.

Після розробки бази даних, було розпочато створення web-додатку засобами мови C#.

Відкриємо заздалегідь встановлене середовище для програмування Visual studio 2019. Створюємо новий проект ASP.NET Web API. Додамо Nuget package Entity Framework Core для роботи з базою даних на базі Sql Server та створимо підключення до бази даних додамо у файл конфігурації Web.config наступний код:

```
<connectionStrings>
    <add name="DataContext" providerName="System.Data.SqlClient"
    connectionString="Data Source = (LocalDB)\MSSQLLocalDB;
    AttachDbFilename=C:\Math_api\Math_api\App_Data\math_db.mdf;
    IntegratedSecurity=True" />
</connectionStrings>
```

Для кожної таблиці клас, в папку Models, який міститиме такі ж поля як відповідна таблиця з відповідними типами даних. Після цього, до цієї ж папки додамо клас, який міститиме колекції раніше описаних моделей таблиць, для роботи зі створеною базою даних за допомогою Entity Framework та назвемо цей клас DataContext.

Створимо необхідні контроллери для відправки даних відповідно до типу запиту та url-адреси. Контроллер це ніщо інше ніж клас який наслідує клас ApiController та має правило іменування «НазваКласуController», так система розпізнає що цей клас є контроллером та відповідає на запити користува.

Додамо до кожного контроллера методи які будуть повертати IHttpActionResult. Цей тип повертає код статусу відправленого користувачем запиту.

Код стану HTTP (англ. HTTP status code) — це 3-значне ціле число, де перша цифра Code-Status визначає клас відповіді, а останні дві цифри не виконують жодної ролі категоризації. Існує 5 значень для першої цифри:

- 1xx: Інформаційний Це означає, що запит отримано і процес триває.
- 2xx: Успіх Це означає, що дія була успішно отримана, зрозуміла та прийнята.
- 3xx: перенаправлення Це означає, що для заповнення запиту необхідно вжити подальших дій.

- 4xx: Помилка клієнта Це означає, що запит містить неправильний синтаксис або не може бути виконаний.
- 5xx: Помилка сервера

HTTP визначає набір методів запиту, щоб вказати бажану дію, яку слід виконати для даного ресурсу. Кожен з них реалізує різну семантику.

GET

Метод GET запитує представлення зазначеного ресурсу. Запити за допомогою GET повинні отримувати лише дані.

POST

Метод POST використовується для створення сутності до вказаного ресурсу, часто спричиняючи зміну стану або побічні ефекти на сервері.

PUT

Метод PUT замінює всі поточні дані цільового ресурсу даними запиту.

DELETE

Метод DELETE видаляє вказаний ресурс.

Назви методів в контроллері повинні відповідати назві відповідному типу HTTP запиту. Відправлятимемо користувачу два основних статуса «OK» та «BadRequest» та відповідні дані якщо це потрібно у відповідності до типу HTTP запиту.

Створені запити можна легко протестувати утилітою Postman. Postman — це потужний інструмент тестування API. Postman наповнена безліччю корисних функціональностей доступних безкоштовно, і в налаштуваннях гнучка.

2.4. Мобільна розробка на базі Xamarin

Для початку розробки мобільного додатку в Xamarin потрібно завантажити всі необхідні компоненти для Visual studio 2019. Це можна зробити за допомогою інсталятора Visual studio. Також потрібно встановити емулятор Android для тестування, або увімкнути режим розробника на Android–смартфоні та дозволити «Debug за допомогою USB». Останній варіант кращий, оскільки одразу можна протестувати додаток на реальному пристрої, але не рекомендується робити такі маніпуляції на сторонніх ПК, дивлячись зі сторони безпеки.

Далі необхідно створити новий проект у середовищі розробки типу «Mobile App (Xamarin.Forms)».

При розробці веб-серверу ми розглядали принцип MVC. Ми створювали лише моделі та контролери відповідно наш мобільний додаток буде представленням(View) у розглянутому принципі MVC. Однак мобільні додатки також мають свої принципи проектування. Найпоширеніший з них є MVVM.

Шаблон MVVM був спеціально розроблений для Windows Presentation Foundation (WPF) та платформ Microsoft Silverlight, і його можна використовувати на всіх платформах XAML

MVVM був уточнений з MVC, і в цьому шаблоні View активний з поведінкою, подіями та прив'язкою даних, а Перегляд синхронізується з ViewModel (що дозволяє відокремлювати презентацію та розкриває методи та команди для управління та маніпулювання Моделлю.

MVVM(Model-View-ViewModel) складається з трьох основних компонентів:

- Модель (представляє дані з перевіркою та логікою бізнесу)

• Перегляд (Вигляд відповідає за визначення структури, компонування та зовнішнього вигляду того, що бачить користувач на екрані. В ідеалі подання визначається виключно за допомогою XAML, з обмеженим кодом, який не містить даних про бізнес-логіку.

• ViewModel (відокремлює вигляд від моделі й розкриває методи та команди для маніпулювання даними (Model).

Перегляд отримує дані від ViewModel (за допомогою прив'язки даних та методів), а під час виконання Перегляд буде змінюватися, коли реагувати на події в ViewModel.

ViewModel опосередковується між View і Model і обробляє логіку View. Він взаємодіє з Моделлю – приймає дані з Моделі та представляє їх для подання для відображення.

Усі ці компоненти відокремлюються один від одного, що дозволяє більшою гнучкістю працювати над ними незалежно, ізолювати блок тестування та замінювати їх, не впливаючи на будь-який інший компонент.

Ця структура дозволяє Моделі та іншим компонентам розвиватися незалежно, дозволяючи розробникам працювати над різними аспектами рішення одночасно. Наприклад, там, де дизайнери працюють над представленням, вони просто генерують зразки даних, не потребуючи доступу до інших компонентів. Це полегшує перепроектування користувацького інтерфейсу, оскільки представлення реалізовано в XAML.

Після створення нашого проекту, додамо одразу представлення для авторизації користувачів, відображення новин, опитувань, чатів, повідомлень та налаштувань.

Головне представлення MainPage перевизначимо типом TabbedPage. Таким чином, ми отримаємо навігацію відповідних представлень у вигляді вкладок.

Додамо до теки Models такі ж моделі як і на серверній частині.

Створимо клас, який буде відповідати за відправлення запитів на веб-сервер, використовуючи HttpClient. Для цього встановимо NuGet пакет System.Net.Http, та Newtonsoft.Json – для трансформування об’єктів у форматі Json та навпаки.

На кожному представленні розмістимо необхідні елементи за допомогою розмітки Xaml.

Для кожного представлення створимо представлення моделі(ViewModel), для кожного представлення та впровадимо прив’язку відповідних полів представлення моделі та відповідного представлення за допомогою реалізації інтерфейсу INotifyPropertyChanged. Таке впровадження дозволить оновлювати автоматично об’єкти в представленні моделі після змін на представленні.

2.5. Інструкція для користувача

При першому запуску додатку користувачу потрібно зареєструватись за допомогою корпоративної пошти (Рис.1).

Після реєстрації користувач перенаправляється на головну сторінку з вкладками. За замовченням відкрита вкладка новини (Рис.2).

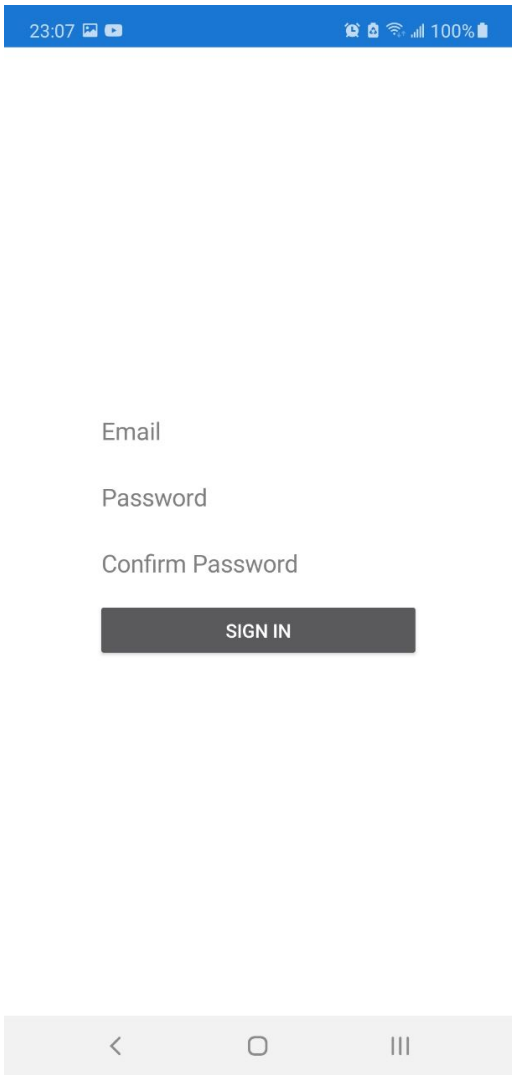


Рис.1

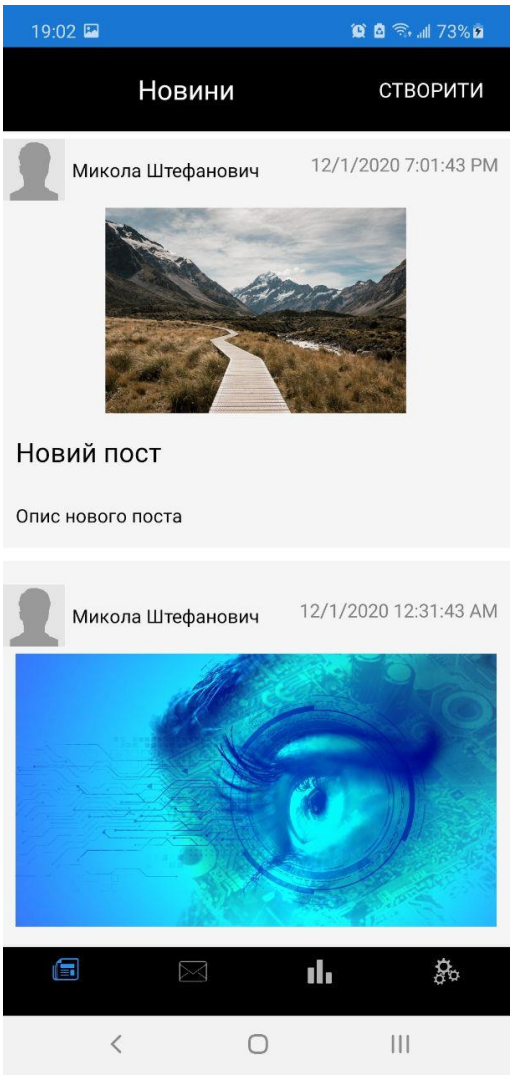


Рис. 2

Біля заголовку «Новини» є кнопка «Створити» яка перенаправляє користувача на сторінку створення постів (Рис.3).

Наступна вкладка на головній сторінці містить список чатів користувача (Рис.4).

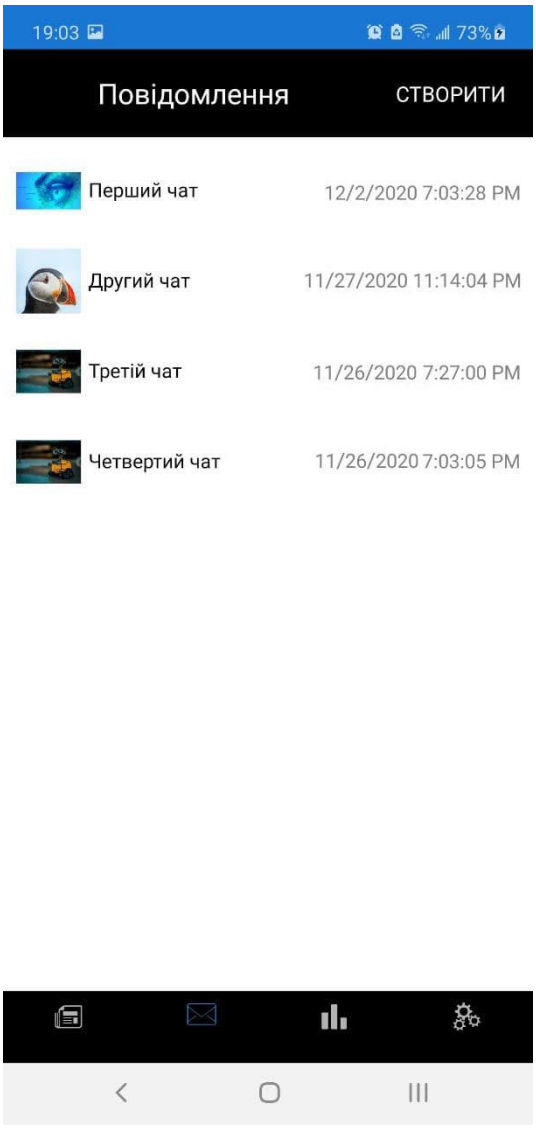


Рис.3

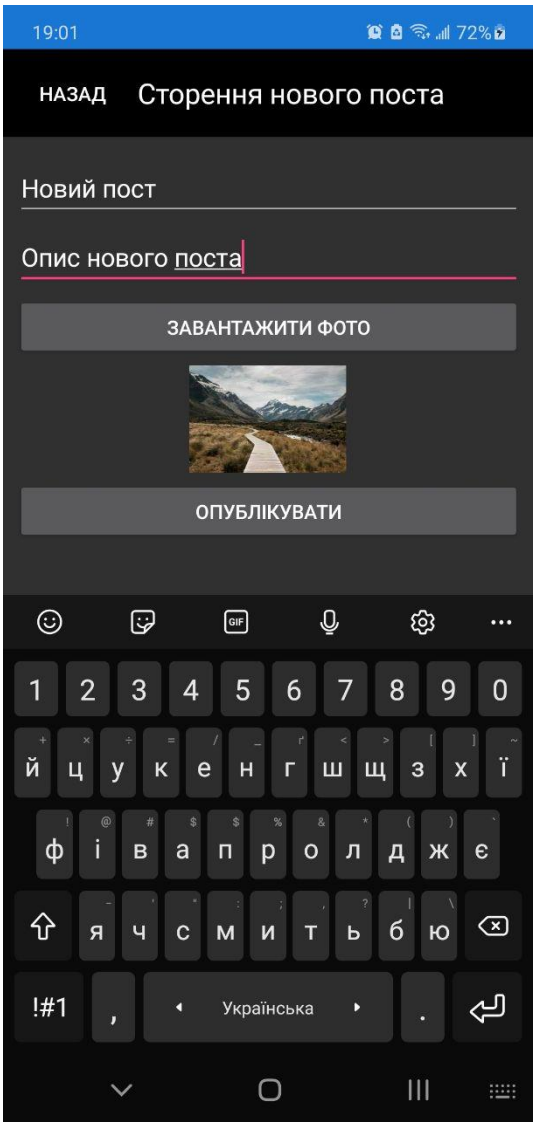


Рис.4

В кожній сторінці чату користувач може переглянути повідомлення чату та відправити своє повідомлення учасникам даного чату (Рис. 5).

Сторінка «Опитування» містить список опитувань (Рис.6).

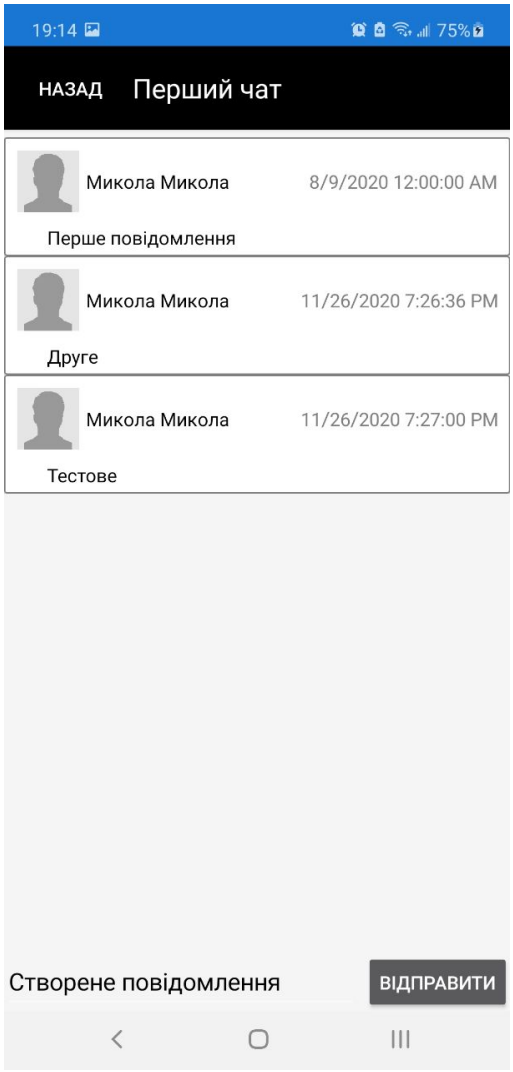


Рис. 5

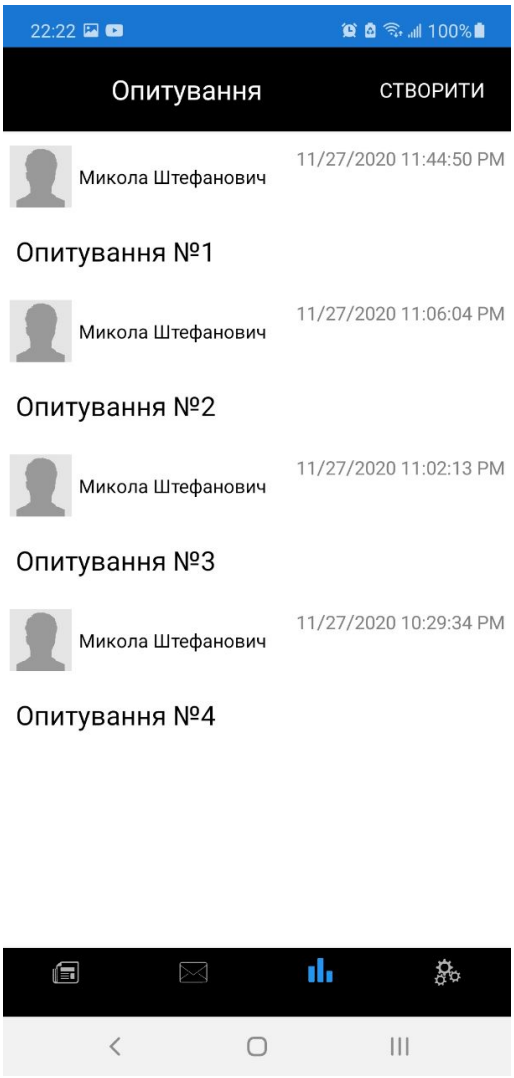


Рис. 6

При натисканні на відповідне опитування, користувач може дати відповіді на питання в опитуванні (Рис.7).

На сторінці «Налаштування» користувач може змінити своє фото, ввімкнути або вимкнути сповіщення повідомлень або змінити пароль (Рис.8).

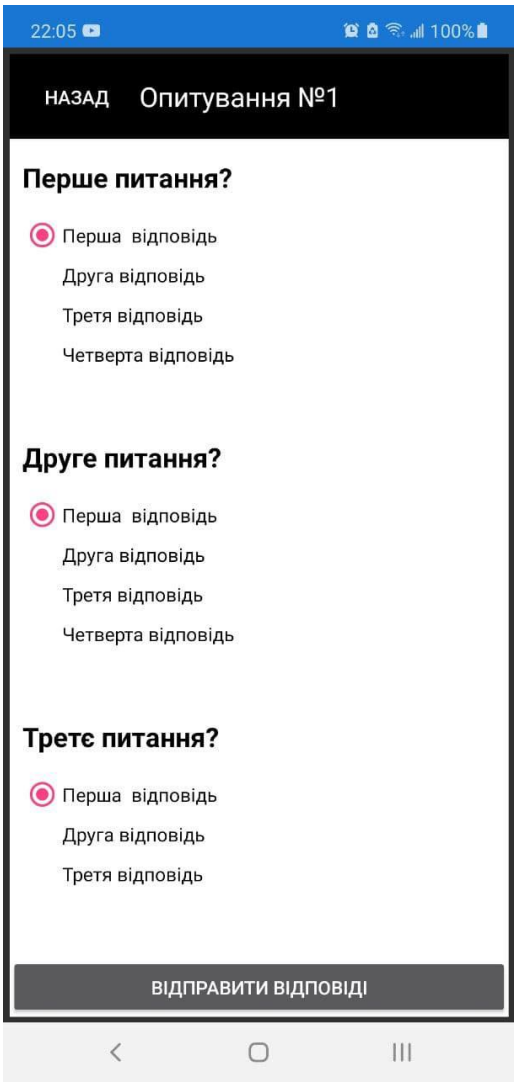

22:05
100%
НАЗАД
Опитування №1
Перше питання?
Перша відповідь
Друга відповідь
Третя відповідь
Четверта відповідь
Друге питання?
Перша відповідь
Друга відповідь
Третя відповідь
Четверта відповідь
Третє питання?
Перша відповідь
Друга відповідь
Третя відповідь
ВІДПРАВИТИ ВІДПОВІДІ

Рис. 7

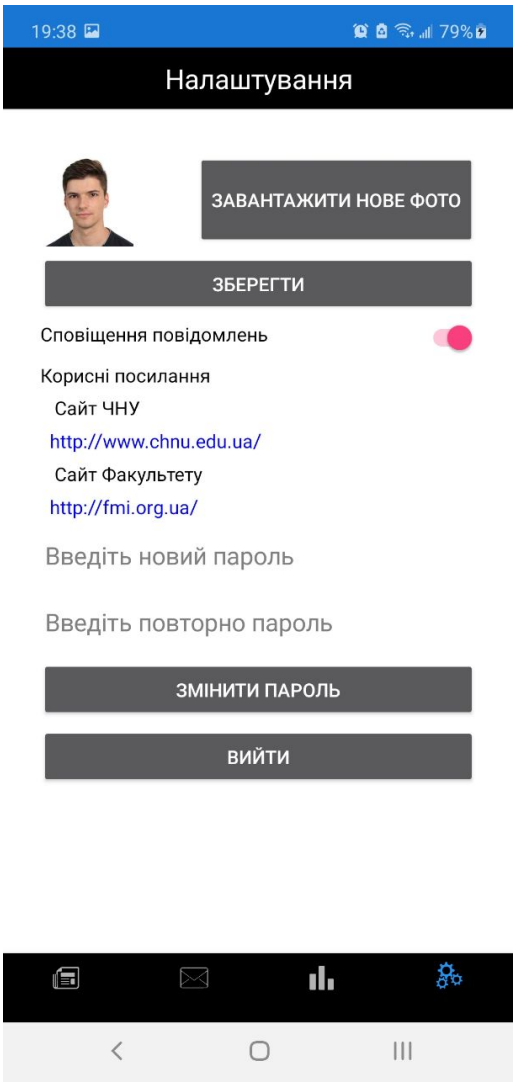

19:38
79%
Налаштування
ЗАВАНТАЖИТИ НОВЕ ФОТО
ЗБЕРЕГТИ
Сповіщення повідомлень
Корисні посилання
Сайт ЧНУ
http://www.chnu.edu.ua/
Сайт Факультету
http://fmi.org.ua/
Введіть новий пароль
Введіть повторно пароль
ЗМІНИТИ ПАРОЛЬ
ВИЙТИ

Рис.8

Висновок

Сучасний розвиток інформаційно-комунікаційних технологій розкриває широкі можливості для використання мобільних технологій. За допомогою яких, студенти та учні можуть отримати доступ до навчальних матеріалів у будь-який час, що стимулює того, хто навчається, до самоосвіти та навчання протягом життя. В процесі написання дипломної роботи були вирішені всі поставлені задачі:

- досліджена сутність корпоративних систем;
- проведено аналіз методів та технологій розробки мобільних додатків;
- обрано технологію розробки для створення мобільного додатку;
- сформовано функціональну структуру мобільного додатку;
- роблено мобільний додаток;
- розроблений додаток було протестовано.

Додаток має корисний функціонал: можливість опублікування новин факультету, проведення анонімних опитувань, обміну повідомленнями. Для розробки мобільного додатку були обрані та засвоєні сучасні технології: HTML5, C#, Xamarin, JSON, OAuth2.0.

Використані технології дуже зручні для розробки, тому що вони є простими та зрозумілими для розробника та водночас швидкими та захищеними.

Переваги використання Xamarin.Forms:

- В процесі розробки створюється єдиний код для всіх платформ.
- Xamarin надає прямий доступ до вбудованих API кожної платформи.
- При створенні додатків ми можемо використовувати платформу .NET і мова програмування C # (а також F #), який є досить продуктивним, і в той же час ясним і простим для освоєння і застосування.

- Xamarin Forms підтримує кілька платформ. Основні платформи: Android, iOS, UWP, Tizen. Додаткові платформи в стані превью: MacOS, WPF, GTK #.

.NET надає ряд переваг, деякі з них:

- Повна підтримка ООП. Бібліотека .NET є бібліотекою класів, а не функцій. Це означає, що при використанні бібліотеки можна створювати об'єкти (екземпляри) цих класів, а так само породжувати нові класи, використовуючи бібліотечні в якості базових.
- Мовна незалежність. У середовищі .NET всі програми, не залежно від того, якою мовою вони написані, компілюються в певний проміжний мову Microsoft Intermediate Language (MSIL). Цим самим досягається мовна сумісність на рівні MSIL. MSIL реалізований в принципах класичного об'єктно-орієнтованого програмування, що допускає тільки одиночне спадкоємство. На рівні MSIL можливо здійснити міжмовне успадкування.
- Загальна система типів - Common Type System (CTS) забезпечує сумісність мов програмування на рівні даних, надаючи погоджений набір основних типів даних.
- Відмова від застосування DLL. DLL в середовищі .NET замінені збірками. Складання мають вбудовані засоби контролю версій. Різні версії збірок можуть співіснувати разом.
- Безпека коду. Середа .NET забезпечує безпеку на рівні коду, строгий контроль типів проміжного мови дозволяє середовищі .NET перед запуском визначити політики безпеки.

- Середовище .NET має вбудовану підтримку і створення web-служб, а також вбудовану підтримку динамічних web-сторінок за допомогою нових технологій ASP .NET.

Література

1. Башмаков А.И. Интеллектуальные информационные технологии: Учеб.пособие./ А.И.Башмаков. – М.: Издательство МГТУ им. Н.Э.Баумана, 2005. – 304 с.
2. Офіційний сайт MySQL [Електронний ресурс]. **–** Режим доступу**:** http://dev.mysql.com
3. Введение в Web-технологии [Електронний ресурс] – Режим доступу до ресурсу: http://bourabai.kz/dhtml/index.html.
4. Основи баз даних: [Навч. посіб.] / І.О. Завадський. — К.: Видавець І.О. Завадський, 2011. — 192 с.: іл.
5. C#/.Net [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp.
6. Xamarin [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/xamarin/.
7. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#, Джеффри Рихтер – СПб: Питер, 2017. – 893 с. – (5).
8. Sql Server tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlservertutorial.net/.
9. ASP.NET Web API 2 [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/aspnet_webapi/.
10. C#Corner [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/what-is-c-sharp/
11. Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Microsoft_SQL_Server/
12. C#Corner [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/basics-of-xamarin/
13. Tutorialspoint [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/xaml/xaml_overview.html

Додатки

Створення таблиць бази даних

```
CREATE TABLE [dbo].[Answers](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Text] [ntext] NULL,
	[QuestionId] [int] NULL,
	[VotesCount] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Chats](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Name] [ntext] NULL,
	[PhotoPath] [ntext] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ChatUsers](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[UserId] [int] NULL,
	[ChatId] [int] NULL,
PRIMARY KEY CLUSTERED
(
```

```
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Messages](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Text] [ntext] NULL,
	[AuthorId] [int] NULL,
	[SentTime] [datetime] NULL,
	[ChatId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Polls](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Name] [ntext] NULL,
	[AuthorId] [int] NULL,
	[PublicationTime] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PollUser](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[UserId] [int] NULL,
	[PollId] [int] NULL,
	[Time] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Posts](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Name] [ntext] NULL,
	[Description] [ntext] NULL,
	[PhotoPath] [ntext] NULL,
	[AuthorId] [int] NULL,
	[PublicationTime] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Questions](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Text] [ntext] NULL,
	[PollId] [int] NULL,
PRIMARY KEY CLUSTERED
```

```
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Settings](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[PhotoPath] [ntext] NULL,
	[MessageNotification] [bit] NULL,
	[UserId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Login] [ntext] NULL,
	[Password] [ntext] NULL,
	[Email] [ntext] NULL,
	[FirstName] [ntext] NULL,
	[LastName] [ntext] NULL,
	[FatherName] [ntext] NULL,
	[IsAdmin] [bit] NULL,
	[IsTeacher] [bit] NULL,
	[Birthday] [date] NULL,
	[PhotoPath] [ntext] NULL,
PRIMARY KEY CLUSTERED
(
	[Id] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[Answers] WITH CHECK ADD CONSTRAINT [FK_Answers_ToQuestions] FOREIGN KEY([QuestionId])
REFERENCES [dbo].[Questions] ([Id])
GO
ALTER TABLE [dbo].[Answers] CHECK CONSTRAINT [FK_Answers_ToQuestions]
GO
ALTER TABLE [dbo].[ChatUsers] WITH CHECK ADD CONSTRAINT [FK_ChatUsers_ToChats] FOREIGN KEY([ChatId])
REFERENCES [dbo].[Chats] ([Id])
GO
ALTER TABLE [dbo].[ChatUsers] CHECK CONSTRAINT [FK_ChatUsers_ToChats]
GO
ALTER TABLE [dbo].[ChatUsers] WITH CHECK ADD CONSTRAINT [FK_ChatUsers_ToUser] FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([Id])
GO
ALTER TABLE [dbo].[ChatUsers] CHECK CONSTRAINT [FK_ChatUsers_ToUser]
GO
ALTER TABLE [dbo].[Messages] WITH CHECK ADD CONSTRAINT [FK_Messages_ToChats] FOREIGN KEY([ChatId])
REFERENCES [dbo].[Chats] ([Id])
GO
ALTER TABLE [dbo].[Messages] CHECK CONSTRAINT [FK_Messages_ToChats]
GO
ALTER TABLE [dbo].[Messages] WITH CHECK ADD CONSTRAINT [FK_Messages_ToUsers] FOREIGN KEY([AuthorId])
REFERENCES [dbo].[Users] ([Id])
GO
ALTER TABLE [dbo].[Messages] CHECK CONSTRAINT [FK_Messages_ToUsers]
GO
ALTER TABLE [dbo].[Polls] WITH CHECK ADD CONSTRAINT [FK_Polls_ToUsers] FOREIGN KEY([AuthorId])
REFERENCES [dbo].[Users] ([Id])
```

```
GO
ALTER TABLE [dbo].[Polls] CHECK CONSTRAINT [FK_Polls_ToUsers]
GO
ALTER TABLE [dbo].[PollUser] WITH CHECK ADD CONSTRAINT [FK_PollUser_ToPolls] FOREIGN KEY([PollId])
REFERENCES [dbo].[Polls] ([Id])
GO
ALTER TABLE [dbo].[PollUser] CHECK CONSTRAINT [FK_PollUser_ToPolls]
GO
ALTER TABLE [dbo].[PollUser] WITH CHECK ADD CONSTRAINT [FK_PollUser_ToUsers] FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([Id])
GO
ALTER TABLE [dbo].[PollUser] CHECK CONSTRAINT [FK_PollUser_ToUsers]
GO
ALTER TABLE [dbo].[Posts] WITH CHECK ADD CONSTRAINT [FK_Posts_ToUsers] FOREIGN KEY([AuthorId])
REFERENCES [dbo].[Users] ([Id])
GO
ALTER TABLE [dbo].[Posts] CHECK CONSTRAINT [FK_Posts_ToUsers]
GO
ALTER TABLE [dbo].[Questions] WITH CHECK ADD CONSTRAINT [FK_Questions_ToPolls] FOREIGN KEY([PollId])
REFERENCES [dbo].[Polls] ([Id])
GO
ALTER TABLE [dbo].[Questions] CHECK CONSTRAINT [FK_Questions_ToPolls]
GO
ALTER TABLE [dbo].[Settings] WITH CHECK ADD CONSTRAINT [FK_Settings_ToUsers] FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([Id])
GO
ALTER TABLE [dbo].[Settings] CHECK CONSTRAINT [FK_Settings_ToUsers]
GO
```

Приклад контролера

```
using MathApi.Models;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

namespace MathApi.Controllers
```

```
{
    [Authorize]
    public class UserController : ApiController
    {
        private DataContext dataContext;
        public UserController()
        {
            dataContext = new DataContext();
        }
        public IEnumerable<User> Get()
        {
            return dataContext.Users.ToList();
        }

        // GET api/controllername/5
        public User Get(int id)
        {
            User user = dataContext.Users.ToList().FirstOrDefault(u => u.Id == id);
            return user;
        }

        // POST api/controllername
        public void Post([FromBody] User user)
        {
            if (user != null)
            {
                dataContext.Users.Add(user);
                dataContext.SaveChanges();
            }
        }

        // PUT api/controllername/5
        public void Put(int id, [FromBody] User user)
        {
            User findUser = dataContext.Users.ToList().FirstOrDefault(u => u.Id == id);
            findUser.Login = user.Login;
            findUser.Password = user.Password;
            findUser.Email = user.Email;
            findUser.FirstName = user.FirstName;
            findUser.LastName = user.LastName;
            findUser.FatherName = user.FatherName;
            findUser.Birthday = user.Birthday;
            findUser.IsAdmin = user.IsAdmin;
```

```
            findUser.IsTeacher = user.IsTeacher;
            dataContext.SaveChanges();
        }

        // DELETE api/controllername/5
        public void Delete(int id)
        {
            User findUser = dataContext.Users.ToList().FirstOrDefault(u => u.Id == id);
            dataContext.Users.Remove(findUser);
        }
    }
}
```

Приклад Моделі (web)

```
using System;

namespace MathApi.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public string Email { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string FatherName{ get; set; }
        public DateTime Birthday { get; set; }
        public bool IsAdmin { get; set; }
        public bool IsTeacher { get; set; }
        public string PhotoPath { get; set; }

    }
}
```

Приклад представлення (мобільний додаток)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:MathMobile.ViewModels"
x:Class="MathMobile.Views.NewsPage"
```

```
Title="{Binding Title}">
<ContentPage.BindingContext>
<local:NewsViewModel/>
</ContentPage.BindingContext>
<ContentPage.Content>
<StackLayout>
<Frame BackgroundColor="black" Padding="10" CornerRadius="0">
<StackLayout Orientation="Horizontal">
<Label Text="Новини" VerticalTextAlignment="Center"
HorizontalTextAlignment="Center" HorizontalOptions="CenterAndExpand"
TextColor="White" FontSize="20"/>
<Button Text="Створити" BackgroundColor="Transparent"
HorizontalOptions="End" TextColor="White" FontSize="15"
Clicked="OnButtonClicked"/>
</StackLayout>
</Frame>
<RefreshView
IsRefreshing="{Binding IsBusy, Mode=OneWay}"
Command="{Binding LoadMessagesCommand}"
Grid.Column="1"
Grid.Row="1"
RefreshColor="Blue">
<CollectionView
x:Name="MessagesList"
ItemsSource="{Binding Posts}"
SelectionMode="Single">
<CollectionView.ItemTemplate>
<DataTemplate>
<StackLayout BackgroundColor="WhiteSmoke" Margin="10, 10, 10, 10">
<StackLayout Orientation="Horizontal">
<Image Source="{Binding AuthorPhotoPath}" HeightRequest="50"
WidthRequest="50"/>
<Label VerticalTextAlignment="center" TextColor="Black" Text="{Binding
AuthorName}"></Label>
<StackLayout HorizontalOptions="EndAndExpand">
<Label
VerticalTextAlignment="center"
HorizontalOptions="EndAndExpand"
TextColor="Gray"
Margin="10"
Text="{Binding Post.PublicationTime}"/>
<!--<Label VerticalTextAlignment="center" TextColor="Gray" Padding="0 10 0
0">22.05.2020</Label>-->
```

```
</StackLayout>
</StackLayout>
<Image Source="{Binding Post.PhotoPath}"
HorizontalOptions="CenterAndExpand"/>
<Label
FontSize="Medium"
TextColor="Black"
Padding="10"
Text="{Binding Post.Name}"/>
<Label
Padding="10"
TextColor="Black"
Text="{Binding Post.Description}"/>
<StackLayout Padding="5" BackgroundColor="White">
</StackLayout>
<StackLayout.GestureRecognizers>
<TapGestureRecognizer
NumberOfTapsRequired="1"
Command="{Binding Source={RelativeSource AncestorType={x:Type
local:MessagesViewModel}}, Path=ChatTapped}"
CommandParameter="{Binding .}">
</TapGestureRecognizer>
</StackLayout.GestureRecognizers>
</StackLayout>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>
</RefreshView>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

Приклад Моделі-Представлення(мобільний додаток)

```
using MathMobile.HttpHelpers;
using MathMobile.Models;
using MathMobile.ServerModels;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Net;
```

```
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace MathMobile.ViewModels
{
   class NewsViewModel : BaseViewModel
   {
      private ObservableCollection<PostUser> posts;
      public Command LoadMessagesCommand { get; }
      public Command<PostUser> ChatTapped { get; }

      public ObservableCollection<PostUser> Posts
      {
         get
         {
            return posts;
         }
         set
         {
            posts = value;
            OnPropertyChanged(nameof(Posts));
         }
      }
      public NewsViewModel()
      {

         LoadMessagesCommand = new Command( async () => await
ExecuteLoadMessagesCommandAsync());
      }
      public async Task ExecuteLoadMessagesCommandAsync()
      {
         IsBusy = true;

         try
         {
            HttpClient client = new HttpClient();
            HttpRequestMessage request = new HttpRequestMessage
            {
               RequestUri = new Uri(Constants.Constants.ApiPost),
               Method = HttpMethod.Get
            };
```

```
            request.Headers.Add("Accept", "application/json");
            HttpResponseMessage response = await client.SendAsync(request);
            string json = "";
            if (response.StatusCode == HttpStatusCode.OK)
            {
               HttpContent responseContent = response.Content;
               json = await responseContent.ReadAsStringAsync();
            }
            Posts =
JsonConvert.DeserializeObject<ObservableCollection<PostUser>>(json.ToStrin
g());
            for (int i = 0; i < Posts.Count; i++)
               Posts.Move(Posts.Count - 1, i);
         }
         catch (Exception ex)
         {
            Debug.WriteLine(ex);
         }
         finally
         {
            IsBusy = false;
         }
      }
   }
}
```

Приклад Моделі (мобільний додаток)

```
using System;
using System.Collections.Generic;
using System.Text;
namespace MathMobile.Models
{
   class Post : BaseModel
   {
      public int? Id { get; set; }
      public string Name { get; set; }
      public string Description { get; set; }
      public string PhotoPath { get; set; }
      public DateTime PublicationTime { get; set; }
      public int AuthorId { get; set; }
   }
}
```