НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект
на здобуття ступеня бакалавра
з напряму підготовки 6. 050201 «Системна інженерія»
на тему: «Соціальна мережа для навчального закладу»

Виконав:
студент IV курсу, групи ІА-52
Комарницький Олександр Сергійович _____

Керівник:
Ст. вик. Моргаль О.М. _____

Рецензент:
_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 рік

АНОТАЦІЯ

Комарницький О.С. Соціальна мережа для навчального закладу. КПІ ім. Ігоря Сікорського, Київ, 2019.

Проект містить 63 с. тексту, 28 рисунків, посилання на 17 літературних джерел, додатки та 4 конструкторських документа.

Ключові слова: соціальна мережа, C#, ASP.NET Core, Entity Framework Core, база даних.

Об’єктом розробки є соціальна мережа для навчального закладу.

Мета розробки – покращення навчального процесу в закладах освіти.

Дипломна робота присвячена розробці соціальної мережі для навчального закладу з використанням мови програмування C#. Результатом роботи став додаток, розроблений за допомогою технологій ASP.NET Core та Entity Framework Core. Були досліджені також такі підходи, як ASP.NET, ASP.NET MVC та ASP.NET Web API, та технології для доступу до баз даних Dapper та ADO.NET. Було виділено основні недоліки та переваги вищезазначених технологій в розробці додатків. Також було розглянуто основні патерни програмування та розробки, правила та рекомендації щодо побудови архітектури таких додатків.

В результаті була спроектована та побудована соціальна мережа, яка може бути корисною для навчальних закладів.

SUMMARY

Komarnitsky O.S. Social network for an educational institution. Igor Sikorsky KPI, Kyiv, 2019.

The project contains 63 pages, 28 figures, links to 17 literary sources, annexes and 4 design documents.

Keywords: social network, C#, ASP.NET Core, Entity Framework Core, database.

The object of development is the social network for an educational institution.

The purpose of the development is to improve the educational process in educational institutions.

This work is devoted to the development of a social network for an educational institution using the C# programming language. The result was an application developed by ASP.NET Core and Entity Framework Core. Also, approaches such as ASP.NET, ASP.NET MVC and the ASP.NET Web API, and technologies for accessing databases Dapper and ADO.NET were also explored. The main drawbacks and advantages of the above-mentioned technologies in the development of applications were highlighted. Also discussed were the main patterns of programming and development, rules and guidelines for building the architecture of such applications.

As a result, a social network was designed and built that could be useful for educational institutions.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

SQL	Structured Query Language, мова для взаємодії з базами даних
API	Application Programming Interface, програмний інтерфейс
SMS	Short Message Service, сервіс для обміну повідомленнями
TCP/IP	Transmission Control Protocol/Internet Protocol
XML	eXtensible Markup Language, мова структурованих даних
CLR	Common Language Runtime, віртуальна машина для мови .NET
HTML	HyperText Markup Language, мова для створення веб-сторінок
MVC	Model-View-Controller, паттерн проектування
JSON	JavaScript Object Notation, текстовий формат обміну даними
HTTP	HyperText Transfer Protocol, протокол передачі даних
UI	User Interface, користувацький інтерфейс
EF	Entity Framework, технологія доступу до баз даних
DI	Dependency Injection, паттерн проектування
БД	База даних
ORM	Object-Relational Mapping, створення віртуальних баз даних
СУБД	Система управління базами даних
LINQ	Language Integrated Query, мова запитів до джерел даних

ВСТУП

Активне поширення інформаційних технологій в сучасному суспільстві сприяло зміні традиційних сфер комунікацій. В наш час Інтернет став головним майданчиком спілкування людей на будь-якій відстані, де б людина не знаходилась.

Студентам потрібне спілкування та обговорення матеріалу на семінарах та спільна робота над домашніми завданнями. За допомогою соціальних мереж студенти можуть радитися та питати поради у вчителів та працівників навчальних закладів.

Раніше розповсюджувати та обмінюватися навчальними матеріалами можна було тільки за допомогою копіювального апарату. З тих пір стався глобальний технологічний переворот в сфері комунікаційних технологій, і інтернет швидко та неочіковано увійшов в наше життя. Раніше головною перешкодою перед безмежним спілкуванням були відстань і час. На сьогоднішній день виникла можливість обміну необхідними для навчання матеріалами, задати віртуально питання через різні додатки та соціальні мережі, не чекаючи консультації викладача, або віддалено здати лабораторні та самостійні роботи. Єдине, що стає на шляху безмежного спілкування сьогодні є наша технічна безграмотність.

На сьогоднішній день з повною впевненістю можна сказати, що найбільш універсальним інструментом спілкування і найпопулярнішими онлайн-сервісами є програми для віддалених комунікацій. Найцікавіше те, що подібні сервіси - це потужний інструмент для організації взаємодії студентів і викладачів, їх участі в професійних спільнотах.

Останнім часом, інновації створюються дуже швидко, в порівнянні з минулими століттями. Багато компаній створюють додатки, які дуже допомагають людству вирішувати різні проблеми. Взагалі, багато інформації можна знайти і про те, як модернізувати процес викладання і навчання. Однак, перш ніж почати втілювати свої або чужі ідеї в життя, варто задуматися: інновації повинні технічно поліпшити і прискорити процес отримання знань або вивести

освітній процес на новий якісний рівень? Так як системи освіти спрямовані на людей, то специфіка сфери освіти така, що до неї завжди будуть пред'являтися особливі вимоги. Тому поряд з технологічним прогресом в освітній сфері важливий так само і процес її гуманізації. Якщо мислити глобально, то головною метою системи освіти є, звичайно, підготовка людини до дорослого життя, навчити боротися з різними труднощами та вирішувати складні проблеми. Тому необхідно будувати навчальний процес таким чином, щоб він розвивав потенційні можливості людини, допомагав знайти себе і розвивати свої таланти в професійній і соціальній сфері. Адже під час роботи в інтернеті учень може не тільки отримувати нові знання, а й активно практикувати корисні комунікативні навички завдяки взаємодії з іншими членами професійних онлайн-спільнот. В сучасних умовах свободи слова і права на отримання і поширення інформації суспільство повинно навчитися використовувати можливості масової комунікації з максимальним для себе ефектом. На даний момент, телебачення і інтернет активно беруть участь в процесі становлення суспільної свідомості, також щомиті зростає швидкість інформаційних процесів, а розвиток засобів обробки і передачі інформації йде вперед семимильними кроками. В останні роки телебачення навіть втрачає популярність, адже в інтернеті можна знайти все, що завгодно. Тепер система засобів масової комунікації забезпечує нову і ефективну зв'язність елементів суспільства, його життєдіяльність і психологію, і треба скористатися цим інструментом в освітніх цілях.

1. ПОНЯТТЯ СОЦІАЛЬНОЇ МЕРЕЖІ. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Є різні проблеми ефективності сучасного освітнього процесу. З одного боку, зростає потік інформації з кожним роком, який повинні сучасні покоління вчити та розуміти. З іншого боку, учні та студенти зазвичай мають дуже низьку мотивацію для засвоєння цієї інформації. Тому головним завданням процесу інформатизації системи освіти можна назвати перетворення сучасних інформаційних ресурсів і інформаційно-комунікаційних технологій в ресурс освітнього процесу, що забезпечує формування якісно нових результатів освіти. Мабуть, необхідно прагнути до того, щоб для студентів та учнів процес отримання та засвоєння нових знань став для нас звичним і був успішно інтегрований в наше повсякденне життя. В умовах сучасного світу метою інноваційної діяльності в освіті має стати якісне зміна учня як особистості. Можливо, розвиток вміння самостійно орієнтуватися в потоках інформації, самостійно розвиватися та мотивувати себе на корисні види діяльності, формування творчого підходу до вирішення проблем - навички, які повинен отримати людина, яка закінчила будь-який навчальний заклад. Пробуючи різні передові і нестандартні освітні підходи на практиці, можна дати студентам курс на випереджальний розвиток, що відповідає в перспективі інтересам суспільства, даної особистості і потенційних роботодавців. У реалізації таких цілей викладачам можуть допомогти соціальні мережі і блоги, програми для відеочату і відеоконференцій, системи для 3D коммунікацій, онлайн-сервіси і безліч інших інструментів, які можна використовувати для модернізації навчального процесу, а так само для створення теоретичної бази онлайн-курсу при змішаному або електронному навчанні.

В даному розділі будуть розглянуті основні поняття соціальної мережі, а також – буде проведено розгляд уже існуючих рішень.

1.1 Поняття соціальної мережі

Соціальна мережа - це платформа, онлайн-сервіс або сайт, призначена для побудови, відображення і організації соціальних взаємин, візуалізацією яких є соціальні графи. Принцип роботи різних мереж дуже схожий - у багатьох з них є стіна профілю і новинна стрічка, можливість проводити голосування і опитування, створювати фотоальбоми і відзначати знайомих на фото, вести особисту та групову переписку і обмінюватися файлами. Звичайно, у кожної з них є і свої особливості, які, як правило, і впливають на вибір користувачем конкретної соціальної мережі (більшість людей найбільш часто використовує лише одну).

Сьогодні під терміном «соціальні мережі» розуміють, перш за все, онлайн-сервіси в мережі Інтернет, призначені для створення взаємовідносин між людьми.

Особливості соціальних мереж:

1) надання користувачам можливостей для обміну інформацією;

2) створення профілів користувачів, в яких потрібно вказувати певну кількість персональної інформації;

3) друзями у соціальніх мережах стають переважно не віртуальні, а реальні друзі.

Веб-ресурс соціальної мережі надає можливості:

1) активного спілкування;

2) створення публічного або закритого профілю користувача, що містить персональні дані;

3) організації та ведення користувачем списку інших користувачів, з якими у нього є деякі соціальні відносини;

4) перегляду зв'язків між користувачами всередині соціальної мережі;

5) утворення груп користувачів за інтересами;

6) управління вмістом в рамках свого профілю;

7) синдикації контенту;

8) підключення різних додатків.

1.2 Огляд існуючих соціальних мереж

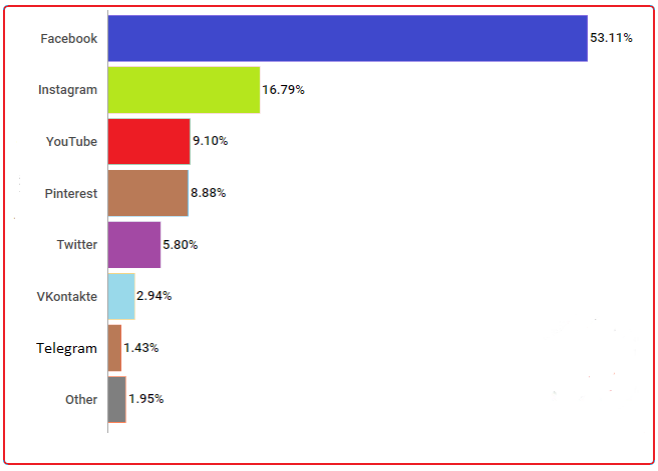

Рисунок 1.1 – Перелік найпопулярніших соціальних мереж в Україні

Найпопулярнішими соціальними мережами в Україні є Facebook, Instagram, YouTube, Twitter, VKontakte, LinkedIn та Telegram. Коротко буде описано про кожен з них, а для деяких буде виділено спеціальний розділ, де вони будуть більш детально описані.

1. Facebook

   Найбільша та найбільш популярна соціальна мережа у світі, що почала працювати 4 лютого 2004 року як мережа для студентів деяких американських університетів. Засновником та головою сервісу є Марк Цукерберг.

2. Instagram

   Ця соціальна мережа орієнтована на обмін відео та фото. З'явилась в 2010 році. Досить швидко зайняла лідируючі позиції. Користувачам сподобалася можливість ділитися відео, фотознімками.

3. YouTube

   Популярний відеохостинг, що надає послуги розміщення відеоматеріалів. Заснований 14 лютого 2005 року трьома працівниками PayPal: Чадом Герлі, Стівеном Чені та Джаведом Карімом.

4. Telegram

   Соціальний мессенджер. Тут можна знайти будь-якого користувача за мобільним телефоном або логіном.

5. Twitter

   Соціальна мережа мікроблогів, яка дає змогу користувачам надсилати короткі текстові повідомлення, використовуючи SMS, служби миттєвих повідомлень і сторонні програми-клієнти.

6. VKontakte

   Соціальна мережа, яка була створена в Росії ще в 2006 році. З того моменту сервіс не втратив своєї популярності. Число акаунтів близько 300 мільйонів, правда не всі акаунти активні. Відомо, що в ВК зареєстровані багато українців, які не мають вільного доступу до цієї соцмережі.

7. LinkedIn

   Соціальна мережа, за допомогою якої можна не тільки обмінюватися повідомленнями та різними файлами, а й знайти нову роботу, укладати угоди, ділитися новинами, інформацією.

1.3 Соціальна мережа Facebook

Рисунок 1.2 – Логотип соціальної мережі Facebook

Facebook — найбільша у світі соціальна мережа, що почала працювати 4 лютого 2004 року як мережа для студентів деяких американських університетів. Засновником та головою сервісу є Марк Цукерберг. За даними Alexa, сайт Facebook.com займає у світі 3 місце за відвідуваністю. Станом на липень 2016 року кількість користувачів становила 1,7 млрд, з яких 1,1 млрд відвідує свої аккаунти щодня. Кількість українських користувачів станом на травень 2014 року становила близько 6 млн. 30 червня 2017 року кількість користувачів досягла 2 млрд чоловік у всьому світі, в Україні 10 млн чоловік.

Головний офіс компанії Facebook розташований в місті Менло-Парк, штат Каліфорнія, США. Компанії також належать сервіси Instagram, WhatsApp та розробник шоломів віртуальної реальності Oculus VR. Чистий дохід Facebook за 2015 рік становив 3,688 млрд доларів США, що на 25 % більше, ніж роком раніше (2,94 млрд доларів у 2014). Ринкова капіталізація компанії станом на квітень 2016 року становила 319 млрд доларів. Facebook є однією з найбільших інтернет-компаній у світі, в списку «Fortune 500» найбільших корпорацій США займає 157 місце.

Facebook дозволяє учасникам створити профілі з фотографіями, обмінюватися повідомленнями, запрошувати друзів, організовувати власні групи.

У 2007 р. ресурс запропонував стороннім програмістам поширювати різні програми і заробляти на цьому, що дозволило впровадити в мережу численні розважальні функції. Аудиторія Facebook (на липень 2013 року згідно Вікіпедії) складає більше 1,2 мільярди користувачів. Тільки в Україні кількість її користувачів перевищує 2 мільйон, а в Росії – 3 мільйони. За оцінками фахівців, вартість компанії Facebook оцінюється приблизно в 90 мільярдів доларів. З метою розширення і надання нових послуг в 2010 р. керівництвом було підписано угоду про інтеграцію в мережу сервісу інтернет-телефонії Skype. В кінці року Марк Цукерберг також офіційно оголосив про запуск власної поштової служби.

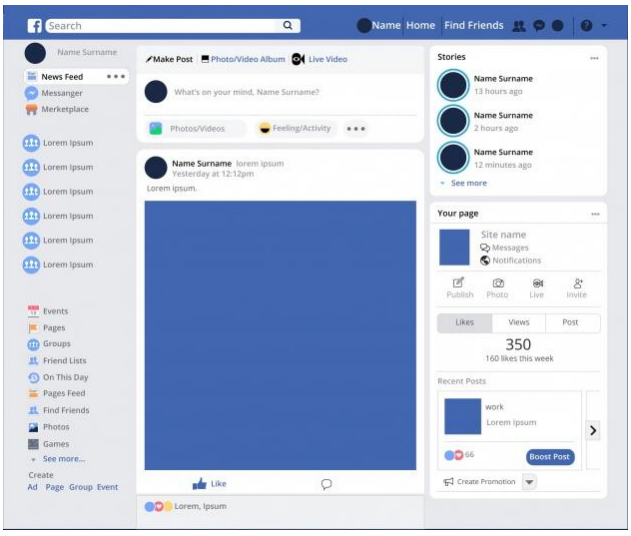

Рисунок 1.3 – Користувацький інтерфейс соціальної мережі Facebook

Днем створення мережі Facebook вважається 28 жовтня 2003, коли Марк Цукерберг, будучи студентом-другокурсником, написав інтернет-сайт Facemash та зламав комп'ютерну мережу Гарвардського університету для копіювання приватних фото. Вже в перші години роботи проекту його аудиторія склала 450 відвідувачів. Через кілька днів адміністрація Гарварду звинуватила Цукерберга в порушенні недоторканності приватного життя і закрила сайт. Пізніше всі звинувачення були зняті, а автор проекту повернувся до початкової ідеї.

Вже в наступному семестрі Цукерберг написав новий код веб-сайту і запустив сервіс за адресою thefacebook.com. Спочатку він був доступний тільки студентам Гарвардського коледжу, але незабаром до проекту приєдналися Едуардо Саверін (фінансовий директор), Ендрю Мак-Коллум (дизайнер), Дастін Московіц (програміст) і Кріс Хьюз, що дозволило істотно розвинути ресурс.

У березні 2004 р. Facebook вже розширився до Стенфорда, Єля і Колумбійського університету, потім відкрився і в інших установах Канади і США. У жовтні 2008 р. мережа оголосила про організацію міжнародної штаб-квартири в Дубліні, а в листопаді 2010 року компанія Facebook стала третьою за величиною веб-компанією в США після Amazon і Google.

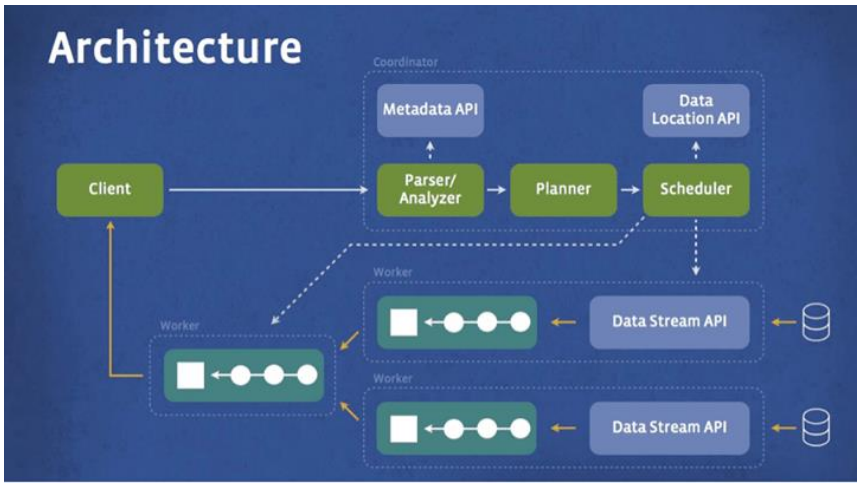

Рисунок 1.4 – Архітектура соціальної мережі Facebook

Понад мільярд активних користувачів роблять Facebook одним з найбільших сховищ даних в світі, що містить більше 300 петабайт. Ці дані використовуються різними способами – від традиційної пакетної обробки до аналітики графів, машинного навчання та інтерактивної аналітики в реальному часі.

Як збільшувати або зменшувати інтерактивного виконання запитів інженери Facebook винайшли Presto, унікальний розподілений движок SQL-запитів, оптимізований для аналізу за запитом. Він використовується тисячами співробітників, які щодня виконують більше 30,000 запитів за допомогою безлічі підключаються бекенд-сховищ даних, таких як Hive, HBase і Scribe.

1.4 Соціальна мережа Instagram

Рисунок 1.5 – Логотип соціальної мережі Instagram

Instagram – це соціальна мережа для обміну та оцінки фотографій і коротких відеороликів. Додаток Instagram сумісний з пристроями iPhone, iPad і iPod Touch на iOS 4.3 і вище, а також з телефонами на Android 2.2 і вище з підтримкою OpenGL ES 2.

У 2012 році Instagram був придбаний компанією Facebook. Ціна купівлі Instagram склала 300 млн доларів грошовими коштами і 23 млн акцій компанії, що в цілому склало $ 1 млрд. За прогнозами експертів, Instagram за 2017 отримав від глобальної реклами близько $ 2,8 млрд. На 2018 рік кількість зареєстрованих користувачів становить 1,1 млрд осіб.

Історія Instagram почалася в Сан-Франциско, коли Кевін Сістром і Майк Крігер вирішили переорієнтувати свій проект Burbn на мобільні фотографії. Додаток з'явилося в магазині додатків App Store компанії Apple в жовтні 2010 року.

Незабаром після випуску додатка до команди приєднався Джош Рідель в якості менеджера спільноти. В цьому ж році до команди приєднався Шейн Суїні як інженер, а в 2011 році - Джессіка Золлман як IT-спеціаліст.

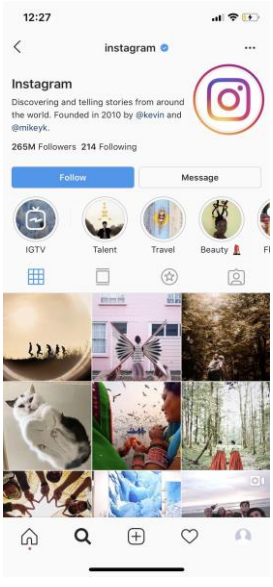

Рисунок 1.6 – Користувацький інтерфейс соціальної мережі Instagram

Пізніше у 2011 році в додаток були додані так звані хештеги для того, щоб було легше знаходити друзів серед користувачів і фотографії знайомих.

У цьому ж році була випущена версія програми 2.0, в якій з'явилися живі фільтри, миттєва зміна нахилу, чотири нових фільтра, фотографії високої роздільної здатності, опціональні кордону, поворот одним кліком і оновлена ікона.

У квітні цього року було випущено версію програми для платформи Android, яке за добу було завантажено більше мільйона разів.

У наступному році було оголошено про покупку мобільного фотододатку Instagram за $ 1 млрд. Власникам фотосервісу Facebook перерахували 300 мільйонів доларів і передасть близько 23 мільйонів своїх акцій. Крім того, соціальна мережа зобов'язалася в разі зриву угоди виплатити Instagram неустойку в 200 мільйонів доларів. У зв'язку з покупкою 25 червня вийшло оновлення, в результаті якого в Instagram був більш пов'язаний з Facebook.

У цьому ж році кожну секунду в Instagram'е ставилося 575 лайків і додавався 81 коментар. У Instagram'е користувачі проводили в середньому 257 хвилин на місяць, що на півтори години більше, ніж в Твіттер'е.

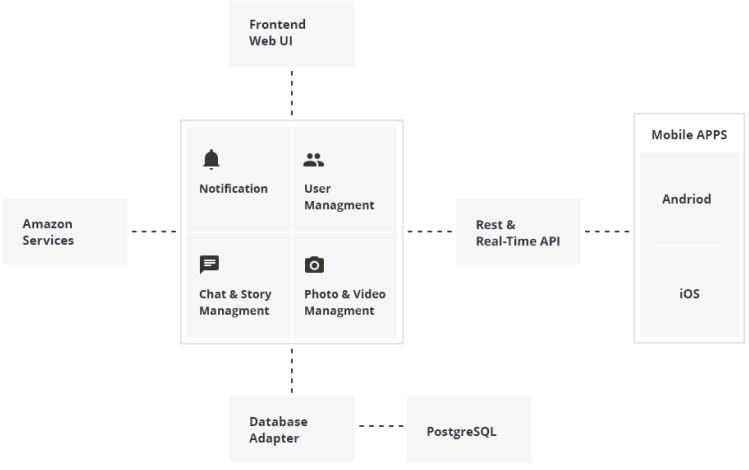

Рисунок 1.7 – Структура соціальної мережі Instagram

В кінці 2012 року Instagram змінив правила користувацької угоди. Найважливішим для людей було те, що Instagram отримує права на використання фотографій, завантажених його користувачами, в тому числі в рекламних цілях. Суспільство сприйняло цю новину дуже агресивно. Було багато поганих відгуків, як серед учасників Instagram'a, так і з боку юристів з авторського права. Багато учасників Instagram'a видалили свої облікові записи. За статистикою, Instagram, можливо, втратив до 25% користувачів - його щоденна відвідуваність знизилася з 16 млн до 12 млн осіб. Зіткнувшись з низкою критики та звинувачень, керівництво сервісу змінило користувацьку угоду, попередньо заявивши, що їхні наміри були неправильно зрозумілі.

У 2013 році Instagram проголосив про початок впровадження своєї нової розробки, яка дозволить користувачам відзначати на фотознімках себе, своїх друзів, цікаві місця, а також відомі бренди. Також було оголошено про можливість налаштовувати систему повідомлень про нові відмітках і в тому числі робити їх приватними.

У цьому ж році Instagram розказав та продемонстрував можливість відеозаписів довжиною в 15 секунд. В кінці 2013 році Facebook домовився про співробітництво з соціальною мережею «ВКонтакте». Пізніше в цьому році було випущено версію Instagram'a для Windows Phone 8.

Через два роки Instagram продемонстрував можливість для брендів створювати фотогалереї і відправляти користувачів на свій сайт за допомогою спеціальної кнопки. В 2015 році Facebook розробив та ввів нову функціональність для брендів: кнопки «Купити зараз» і «Встановити зараз». У липні цього ж року в Instagram стало можливо завантажувати фотографії в значно кращій якості. Влітку цього року в додатку, крім традиційного квадратного формату знімків і відео, з’явився альбомний і портретний формат.

Восени 2015 року в Instagram з'явилась реклама, що складалась з відеозаписів тривалістю до 30 секунд. На початку 2016 року, з’явилась можливість для користувачів заходити в свої облікові записи одночасно з різних пристроїв. Користувачі, які користувались пристроями з операційними системами IOS та Android, могли також користуватися декількома обліковими записами, не виходячи з жодного з них. Для того, щоб ця можливість з'явилась, треба було в налаштуваннях профілю додати додатковий обліковий запис і ввести ім'я або прізвище користувача.

У цьому ж році компанія проінформувала своїх передплатників про зміну структури оновлень в новинах користувача. Тепер новини формувались не на основі хронології, а базувались на користувацьких первагах та закладках. Пізніше Instagram оголосив про плани залишити всі публікації користувачів, проте зі зміної порядку цих публікацій. Нововведення будуть впроваджуватися не відразу, а протягом декількох місяців.

У 2016 році також компанія, яка володіла додатком Instagram, розробила ще одну функцію. Головною ідеєю була зміна тривалості завантаження відеоконтенту, яка значно допомогла користувачам завантажувати власні відеоматеріали. Тривалість відео тепер буде збільшена в чотири рази - з 15 до 60 секунд. Instagram обгрунтував це рішення зростанням інтересу користувачів до відео формату і кількістю опублікованих відеороликів за останні роки.

Наприкінці весни цього року компанія оголосила про зміну дизайну, який став більш стійким та зручним. Деякі символи та кнопки стали набагато зручнішими: чорний колір в їх оформленні змінився на синій. Трохи змінився логотип в кращу сторону.

Влітку Instagram анонсував оновлення додатку під назвою Instagram Stories, яке дозволяло створювати фото і 10-секундні відео з накладенням тексту та інших рукописних позначок. Найголовнішою особливістю даних постів було їх

автоматичне видалення через 24 години, на відміну від звичайних публікацій в стрічках.

На початку 2017 року Instagram розробив нову функцію, яка дозволяла вмістити досить багато фотографій і відеороликів в одному пості. Мультимедійний контент відобразиться у вигляді «каруселі», прокручуваному свайпамі.

Наприкінці 2017 року компанія ввела оновлення, яке сприяло тому, що рекомендовані записи стали з'являтися в основний стрічці.

Влітку наступного року компанія розробила функцію IGTV (InstaGram TV) яка була призначена для довгих відео, розташованих в вертикальному положенні. Дивитися ролики звідти можна буде як в окремому додатку, так і всередині Instagram. Ці відеозаписи могли тривати аж цілу годину, на відміну від Instagram. Люди також могли писати коментарії до відео та ділитися ними з друзями.

В цьому ж році компанія розробила інструмент, який показував користувачам час, проведений в додатку Instagram, який став дуже корисним з часом. Подивитися час, проведений в додатку, можна було в налаштуваннях з назвою «Ваша активність». Користувачі могли контролювати час в соціальній мережі, включаючи нагадування або встановлюваючи ліміт активності кожен сам для себе в додатку Instagram.

Восени 2018 року Кевін Сістром і Майк Крігер оголосили про відставку з компанії соціальної мережі Facebook. Причин для звільнення було багато, проте офіційної версії не було. Наприкінці соціальна мережа Instagram почала тестувати новий інтерфейс мобільного додатка.

На початку 2019 року протягом 10 годин не працювала соціальна мережа Facebook, яка сприяла не працюванню Instagram. Це був перший випадок за всю історію Instagram. Протягом цього часу Facebook, Instagram, Messenger і WhatsApp

виявилися недоступні для багатьох країн по всьому світу. Проблеми виникли через зміни конфігурації сервера. 19 березня 2019 року соціальна мережа почала тестувати нову функцію здійснення онлайн-покупок через свій додаток - Checkout в Instagram.

1.5 Соціальна мережа Twitter

Рисунок 1.8 – Емблема соціальної мережі Twitter

Twitter — це соціальна мережа мікроблогів, яка дає змогу користувачам надсилати короткі текстові повідомлення, використовуючи SMS, служби миттєвих повідомлень і сторонні програми-клієнти.

Власником додатку Твіттер є компанія Twitter Inc, головний офіс якої знаходиться в Сан-Франциско (штат Каліфорнія). Twitter Inc також має сервери й офіси в Сан-Антоніо і Бостоні. За станом на 2012 рік у компанії працює понад 900 співробітників.

Створений Джеком Дорсі в 2006 році, Твіттер незабаром завоював популярність у всьому світі. За станом на 1 січня 2011 року сервіс нараховує понад 200 млн користувачів. 100 мільйонів користувачів проявляють активність хоча б раз на місяць, з них 50 мільйонів користуються Твіттером щодня. 55 % користуються Твіттером на мобільних гаджетах, близько 400 мільйонів унікальних відвідувань отримує за місяць безпосередньо сайт twitter.com. Взимку 2014 році, хакери співтовариства Anonymous нанесли шкоду сайту, зробивши

його небезпечним для користувачів на пару годин, проте проблему було вирішено та сайт почав працювати в попередньому режимі.

У 2010 році дохід компанії Twitter Inc склав 45 мільйонів доларів (із продажів онлайн-реклами).

До 2017 року максимальна кількість символів у повідомленні становила 140 символів. Було збільшенно ліміт символів у повідомленні, що було ковтком свіжого повітря для користувачів. Розробники обгрунтувати це тим, що ця функція допоможе користувачам, які користуються латинськими та символами кирилиці набагато рідше досягати ліміту під час написання повідомлення. Проте для користувачів, які пишуть ієрогліфами, ця функція не мала ніякого значення.

Рисунок 1.9 – Архітектура соціальної мережі Twitter

На початку історії Twitter, проект зіткнувся з масою проблем, пов'язаних з масштабністю. Соціальна мережа частенько давала збої.

Спочатку не було реалізовано жодних форм моніторингу, графіків або статистики, це дуже ускладнювало виявлення та вирішення проблем та недоліків системи, які виникали згодом. Через деяких час були впроваджені Munin і Nagios. Розробники зіткнулися з деякими труднощами при використанні цих продуктів в Solaris. Крім цього був використаний сервіс Google Analytics, але від нього було

занадто мало сенсу, особливо коли сторінки навіть не завантажувались. Розробники додатку почали активно використовувати засобами кешування memcached. Наприклад, якщо підрахунок кількості чого-небудь виконується повільно, набагато ефективніше один раз запам'ятати результат у memcached, ніж кожен раз рахувати його заново.

Також однією незручністю було отримання інформації про статус своїх друзів. Налаштувати дану функцію було непростим завданням. Замість використання запитів інформація про статус друзів відбувалось оновлення в кеші. База даних зовсім не використовувалась. Такий підхід дозволяв отримати передбачуваний час відгуку (обмежене зверху приблизно 20 мілісекундами).

Об'єкти ActiveRecord були настільки великими, що кешування їх було недоцільно. Більшість атрибутів зберігались в хештаблиці, а інша їх частина піддавалось так званому “lazy loading” в момент запиту на доступ.

90% запитів були запитами до API. Таким чином, кешування сторінок або їх фрагментів не мали ніякого сенсу, зате ніхто не заважав їм кешувати самі API запити.

Внутрішня організація роботи з повідомленнями. Повідомлення дуже активно використовуються: виробники генерують повідомлення, вони утворюються в черзі, а потім поширюються по споживачам.

Основна функція Twitter полягає в реалізації своєрідного моста між різними форматами електронних повідомлень (SMS, електронна пошта, сервіси миттєвого обміну повідомленнями і так далі).

Спочатку цей механізм грунтувався на distributed Ruby - бібліотека, що дозволяє відправляти і приймати повідомлення сполучення між віддаленими Ruby-об'єктами по TCP / IP. Але вона була кілька дивною, та й було потенційно слабким місцем з точки зору стабільності.

Згодом сервіс перевели на Rinda, що представляє собою набір загальних для всієї системи черг. Але і у неї були недоліки: всі черги були постійними, а дані губилися при збоях.

Наступною спробою був Erlang. Але одного разу виникла проблема: яким чином зламався сервер може продовжувати працювати, але при цьому в черзі звідкись виникли цілих 20000 очікують користувачів.

Зрештою рішення було розроблено своїми силами: Twitter випустив Starling, розподілений легкий сервер черг, написаний на Ruby і підтримує протокол memcache. Зараз серверна частина Twitter управляється саме їм.

Розподілені черги дозволяють переживати збої шляхом запису їх на диск у критичних ситуаціях. Інші великі інтернет-проекти також часто користуються таким підходом.

Робота з SMS здійснюється за допомогою сторонніх сервісів і послуг, які вони шлюзів. Просто запускаються додаткові сервери з mongrel, більш елегантного рішення поки немає. Всі внутрішні помилки видаються користувачам, якщо обслуговуючий їх mongrel сервер на даний момент замінюється. Всі сервера зупиняються одночасно. Відключення їх по одному з певних причин не використовується.

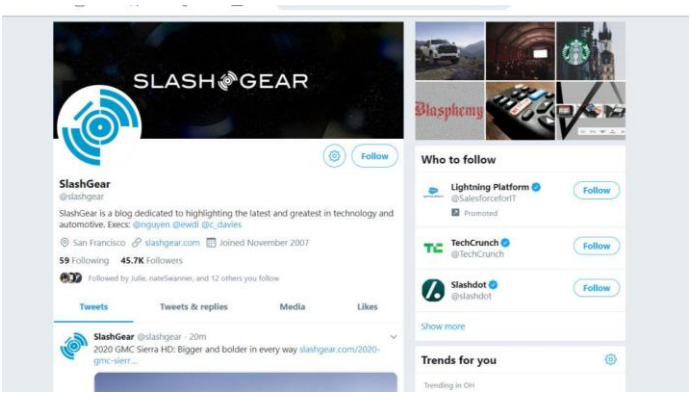

Рисунок 1.10 – Користувацький інтерфейс соціальної мережі Twitter

Багато часу сервіс був недоступний, так як люди проходили спеціальними програмами по сайту з метою додати всіх хто попадався під руку в друзі. 9000 друзів за 24 години. Це просто-напросто зупиняло роботу сайту.

Були розроблені засоби для своєчасного виявлення таких ситуацій. У майбутньому воно буде грунтуватися на часі, а не на користувачах, оскільки запити зазвичай дуже локальні за часом.

Сегментування буде не так просто реалізувати завдяки автоматичному запам'ятовуванню результатів виконання функцій для подальшого повторного їх використання. Ніхто не дасть гарантії, що операції "тільки для читання" насправді будуть такими бути. Запис в slave, що працює в режимі read-only, - не найкраща ідея.

API Twitter генерує в 10 разів більше трафіку, ніж сам сайт. Їх API - найважливіша річ з усіх, що вони розробили.

Простота сервісу дозволила розробникам будувати свої програми поверх інфраструктури Twitter, привносячи все нові і нові ідеї. Наприклад, Twitterrific - гарний спосіб використовувати Twitter в невеликій команді. Моніторинг використовується для зупинки занадто великих процесів.

1.6 Соціальна мережа LinkedIn

Рисунок 1.11 – Логотип соціальної мережі LinkedIn

LinkedIn — соціальна мережа для пошуку і встановлення ділових контактів. У LinkedIn зареєстровано понад 400 мільйонів користувачів (на 2015 рік), що представляють 150 галузей бізнесу з 200 країн. Соціальна мережа LinkedIn була заснована Рейдом Хоффманом у грудні 2002 року, запущена в травні 2003 року. Нинішній керівник компанії — Джефф Вейнер, раніше представник правління Yahoo! Inc. Штаб-квартира компанії знаходиться в Маунтін-В'ю, Каліфорнія, США. LinkedIn має також офіси в Омасу, Чикаго і Лондоні.

Такі фонди, як Greylock, Seqoia Capital, Bain Capital та Bessemer профінансували створення додатку LinkedIn. Весною 2011 року компанія Linkedln неочікувано провела запланованe первинне розміщення акцій (IPO) на Нью-Йоркській фондовій біржі, в ході якого інвесторам було продано 7,84 мільйона акцій по $45 за штуку. Завдяки IPO компанії вдалось залучити $352,8 млн, що зробило це розміщення одним з найбільших в інтернет-секторі США (поступилось лише Google). В перший же день торгів акції LinkedIn користувались шаленим попитом, і завдяки цьому ціна закриття зросла більш, ніж вдвічі — до $94 (найвища зафіксована ціна — $122,7). Таким чином капіталізація LinkedIn склала $8,9 млрд. У багатьох інвесторів є деякий скепсис щодо такої високої оцінки компанії, адже її ціна тепер становить 36 річних доходів ($243 млн). За цим показником вона зрівнялась з Facebook, капіталізація якого на позабіржовому ринку оцінюється в 30-40 річних доходів. LinkedIn станом на березень 2011 року мала 102 млн користувачів. LinkedIn надає можливість зареєс трованим користувачам створювати і підтримувати список ділових контактів. Контакти можуть бути запрошені як з сайту, так і ззовні, проте LinkedIn вимагає попереднє знайомство з контактами.

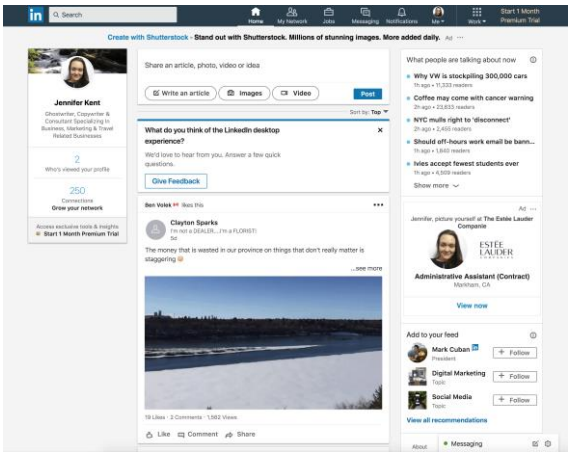

Рисунок 1.12 – Користувацький інтерфейс соціальної мережі LinkedIn

У випадку, коли користувач не має прямого зв'язку з контактом, він може бути представленим через інший контакт.

Користувачі LinkedIn можуть використовувати список контактів у різних цілях:

1. Бути представленими через існуючі контакти і розширювати зв'язки.
2. Здійснювати пошук компаній, людей, груп за інтересами.
3. Публікувати професійні резюме і здійснювати пошук роботи.
4. Рекомендувати і бути рекомендованими.
5. Публікувати вакансії.
6. Створювати групи за інтересами.

1.7 Соціальна мережа Telegram

Рисунок 1.13 – Логотип соціальної мережі Telegram

Telegram — месенджер, програмне забезпечення для смартфонів, планшетів та ПК, яке дозволяє обмінюватися текстовими повідомленнями та різноманітними файлами, зокрема графічними файлами та відеофайлами, а також безкоштовно телефонувати іншим користувачам програми.

Обліковий запис користувача прив'язується до номеру мобільного телефону: щоб авторизуватися, потрібно ввести код авторизації з СМС. Такі коди мають обмежені терміни придатності. Таким чином, користувач позбавляється необхідності запам'ятовувати чи зберігати десь свій пароль.

Проект було створено Павлом Дуровим, засновником соціальної мережі ВКонтакті. В інтерв'ю Нью-Йорк Таймс він розповів, що початкова ідея додатку прийшла йому ще 2011 року, коли до його дверей приходив спецназ. Коли вони пішли, Дуров відразу ж написав своєму братові Миколі. Тоді ж він і усвідомив, що у нього немає безпечного способу комунікації з братом. Сервіс побудований на технології шифрування листування MTProto, розробленій братом Павла Миколою. Сам Telegram спочатку був експериментом компанії Digital Fortress (належить Павлові) з метою протестувати MTProto на великих навантаженнях.

У 2013 році був створений перший клієнт Telegram для пристроїв з операційною системою iOS.

Пізніше в цьому ж році один програміст створив та виклав у відкритий доступ перший додаток для операційної системи Android, сумісний з Telegram (використовує той самий протокол MTProto).

Взимку соціальна мережа розширилась, відкрився веб-сайт і була представлена офіційна версія Telegram під Android з відкритим вихідним кодом (GPL2). Попередня версія додатку доступна під назвою «Unofficial Telegram S».

В кінці 2013 року з'явилися сторонні клієнти сервісу для Windows і OS X з обмеженим функціоналом. Також був розроблений концепт веб-версії клієнта.

На початку наступного року було створено неофіційну веб-версію Webogram від колишнього розробника ВКонтакті Ігоря Жукова. Згодом Жуков став розробником офіційної веб-версії месенджера.

Влітку цього ж року в магазині додатків для пристроїв з операційною системою IOS з'явився додаток Telegram HD, який створила компанія Telegram Messenger LLP.

Новий додаток отримав спеціальну версію для пристроїв з операційною системою IOS і набагато кращу підтримку відео і фотографій високої роздільної якості. Також було додано можливість пересилки анімованих зображень у форматі GIF. Взимку 2014 року в додаток була додана підтримка так званих псевдонімів, за допомогою яких стало можливо знаходити користувачів, навіть не знаючи їх номера телефону, а також з’явилась можливість дивитись профіль користувача. На початку 2015 року в соціальну мережа також була додана підтримка наліпок, якими можна було користуватись під час обміну повідомленнями з іншими користувачами. Спочатку в додатку 14 наліпок, але будь-який користувач міг модифікувати їх або додати свої власні за допомогою цікавих функцій. На відміну від багатьох додатків, у Telegram наліпки повністю безкоштовні. Влітку цього ж року додаток було неочікуванно заборонено у Китаї. В 2016 році власник соціальної мережі Telegram заявив, що додатком користуються вже понад 100 мільйонів людей, при цьому сервіс доставляє близько 15 мільярдів повідомлень щодня. Також він підкреслив, що у 2015 році Telegram передавав 12 мільярдів послань за день.

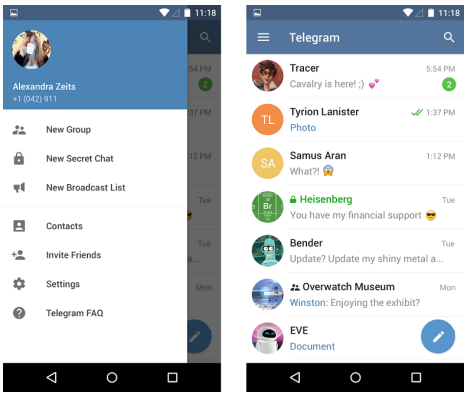

Рисунок 1.14 – Користувацький інтерфейс соціальної мережі Telegram

Весною 2016 року стало відомо, що в травні 2015 року Google розглядав можливість купівлі месенджера за більш, ніж 1 млрд доларів США, проте відмовився від цієї ідеї. Згодом з'явилася можливість редагування відправлених повідомлень, яка дозволяла внести зміни протягом двох діб з моменту відправлення при якому з'являвся спеціальний надпис біля повідомлення.

Взимку цього ж року було анонсовано проект Telegraph — платформа, безкоштовний видавничий інструмент, який дозволяв створювати публікації, огляди, вставляти фотографії і відеозаписи. У цього проекту були деякі ознаки блог-платформи, месенджера і соціальної мережі.

В наступному році колишній працівник ВК та Telegram Антон Розенберг розповів, що розробка месенджера ведеться в основному в Санкт-Петербурзі, в тій самій будівлі на Невському проспекті, де розташований офіс російської соцмережі ВКонтакті, сам Дуров заперечив ці дані, назвавши Антона «психічно хворим».

Влітку 2018 року було анонсовано Telegram Passport, який дозволяв зберігати персональні дані, такі як фотографії паспортів, квитанцій, комунальних рахунків, у захищеному наскрізним шифруванням хмарному сховищі Telegram. Користувач міг запросто завантажити свої документи, а потім користуватися ними, коли було потрібно та ділитися ними з сервісами, які вимагають авторизації.

Як запевнили розробники, соціальна мережа Telegram не мала доступу до цього веб-сервісу.

1.8 Висновок до розділу 1

У даному розділі було розглянуто основні поняття соціальних мереж. Також описано найпопулярніші соціальні мережі в Україні, їх призначення, структуру та історію створення. Деяким соціальним мережам було приділено більше уваги, а саме Facebook, Instagram, Twitter, LinkedIn та Telegram, тому що вони є найцікавішими у порівнянні з аналогами.

2. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СОЦІАЛЬНОЇ МЕРЕЖІ

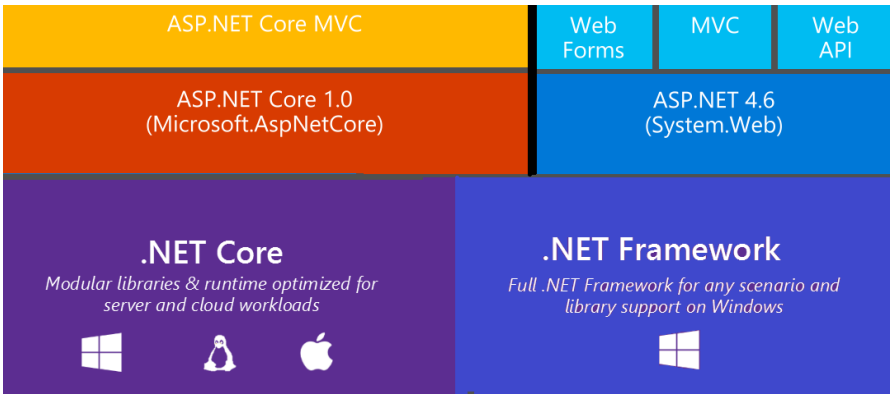

Рисунок 2.1 – Підходи технології ASP.NET

В даному розділі буде розглянуто сучасні технології, які можна застосувати для створення соціальної мережі на мові C#.

Спочатку буде розглянуто основні можливості технології ASP.NET, буде описано, що саме лежить в основі ASP.NET.

Далі будуть більш детально описані такі підходи, як ASP.NET MVC, ASP.NET Web API та ASP.NET Core. Найцікавішим є, звичайно, ASP.NET Core, адже цей фреймворк являє собою повний перепис, який об'єднує раніше окремі ASP.NET MVC та ASP.NET Web API у єдину програмувальну модель.

Може виникнути таке питання, навіщо взагалі використовувати C#, адже можна писати програми на Java чи на C++ та Swift, що є досить перспективними та цікавими мовами програмування. Дійсно, але мова C# має багату історію, цікаві конструкції, фреймворки, велику спільноту. Тому хотілося б написати основні та найбільш цікаві можливості даної мови:

1. Інкапсульовані сигнатури методів, звані делегатами, які підтримують типобезпечні повідомлення про події.

2. Властивості(properties), що виступають в ролі методів доступу для закритих змінних-членів.

3. Атрибути з декларативними метаданими про типи під час виконання.

4. Вбудовані коментарі XML-документації.

5. LINQ, що пропонує вбудовані можливості запитів в різних джерелах даних.

6. Підтримка асинхронних операцій.

7. Використання TPL – Task Parallel Library в .NET.

8. Використання широкого спектру класів .NET та фреймворків.

Як видно, дана мова має багато гарних можливостей, вона схожа на C++ та легша в засвоєнні.

2.1 Технологія ASP.NET. Опис та основні можливості

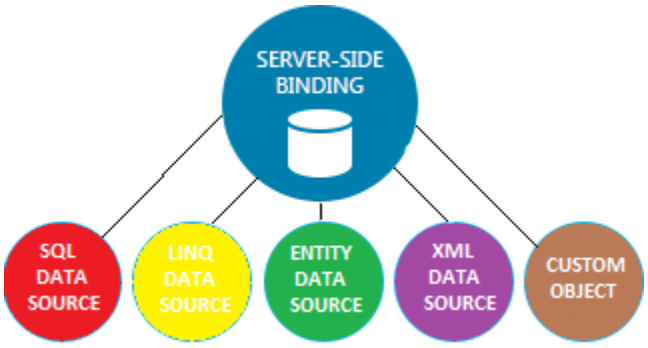

Рисунок 2.2 – Можливості технології ASP.NET

ASP.NET — технологія створення веб-застосунків і веб-сервісів, яка була створена компанією Microsoft. Ця технологія є основною частиною платформи Microsoft.NET і розвитком старішої технології Microsoft ASP. На цей час останньою версією цієї технології є ASP.NET Core 2.0.

ASP.NET дуже схожа на стару версію ASP зовні, що дозволяє розробникам відносно легко перейти на ASP.NET. У той же час внутрішній устрій ASP.NET істотно відрізняється від ASP, оскільки вона заснована на платформі .NET і, отже, використовує всі нові можливості, що надаються цією платформою.

Після випуску сервера Internet Information Services 4.0 в 1997 році, компанія Microsoft почала досліджувати можливість нової моделі веб-застосунків, яка

задовольнить скарги на ASP, особливо пов'язані з відділенням оформлення від змісту, і яка дозволить писати «чистий» код. Робота з розробки такої моделі була доручена Марку Андерсу, менеджеру команди IIS, і Скотту Гутрі, що прийшов на роботу в Microsoft в 1997. Андерс і Гутрі розробили початковий проект протягом двох місяців, і Гутрі написав код первісного прототипу під час різдвяних канікул 1997 року.

Початковий проект називався «XSP»; Гутрі пояснив в інтерв'ю 2007 року що, «завжди запитують, що означає буква X. У той час вона нічого не значила. XML починається з неї; XSLT починається з неї. Все кльове починається з X, тому ми його так і назвали.» Прототип XSP був написаний на Java, але скоро було вирішено побудувати нову платформу на основі Common Language Runtime (CLR), бо на платформу Java у компанії Microsoft закінчувалась ліцензія. Гутрі описав це рішення як «величезний ризик», тому що успіх нової розробки був пов'язаний з успіхом CLR, яка, як і XSP, перебувала на ранній стадії розробки.

Корпорація Майкрософт рекомендує використовувати в динамічному коді програми code-behind model, яка розміщує цей код у окремому файлі або в спеціально позначеному тегу. Файли коду, як правило, мають імена типу «MyPage.aspx.cs» або «MyPage.aspx.vb», а файл сторінки — MyPage.aspx (таке ж ім'я, як і у файла сторінки (ASPX), але з розширенням, що визначає сторінку мови). Ця практика використовується у Visual Studio та інших IDE. Також, у форматі веб-додатків, pagename.aspx.cs є частковим класом, який пов'язаний з файлом pagename.designer.cs. Файл дизайнера — це файл, який автоматично створюється з ASPX-сторінки, і дозволяє розробнику посилатись на компоненти сторінки ASPX зі сторінки CS без необхідності їх оголошувати вручну, як це було в попередніх версіях ASP.NET. Використовуючи цей стиль програмування, розробник пише код, що відповідає на різні події, такі як завантаження сторінки або натискання елемента керування, а не лише на процедурний перегляд документа.

Code-behind model ASP.NET відрізняється від класичного ASP, оскільки він заохочує розробників створювати додатки, відокремлюючи презентацію та зміст. Теоретично, це дозволить веб-дизайнерам, наприклад, зосередити більше уваги на розмітці дизайну, звертаючи менше уваги на порушення програмного коду, який його запускає. Це схоже на відокремлення контролера від представлення в рамках model–view–controller (MVC).

.NET використовує технологію рендерингу «відвіданих композитів» («visited composited»). Під час компіляції файл шаблону складається в код ініціалізації, який створює дерево керування (композит), що представляє вихідний шаблон. Літерний текст переходить в екземпляри класу Literal control, а елементи керування сервером в екземпляри специфічного класу керування. Код ініціалізації поєднується з кодом, написаним користувачем (зазвичай шляхом збірки декількох часткових класів), і переходить до специфічного класу сторінки.

Фактичні запити на сторінку обробляються у кілька кроків. По-перше, під час ініціалізації створюється екземпляр класу сторінки та виконується код ініціалізації. Це створює початкове дерево керування, яке маніпулюється методами сторінки в наступних кроках. Оскільки кожен вузол у дереві являє собою елемент керування, представлений як екземпляр класу, код може змінювати структуру дерева, а також маніпулювати властивостями / методами окремих вузлів. Нарешті, під час етапу візуалізації користувач (відвідувач) має відвідати кожен вузол у дереві, вимагаючи, щоб кожен вузол сам використовував методи відвідувача. Вихідний HTML надсилається клієнту.

Після того, як запит було оброблено, екземпляр класу сторінки відкидається, а разом з ним і все дерево керування. Це викликає труднощі у програмістів-початківців ASP.NET, які звертаються до членів екземпляру класу, які відкидаються з кожним циклом запиту / відповіді сторінки.

Хоча ASP.NET бере свою назву від старої технології Microsoft ASP, вона значно від неї відрізняється. Microsoft повністю перебудувала ASP.NET, ґрунтуючись на Common Language Runtime (CLR), який є основою

всіх застосунків Microsoft .NET. Розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, що входять у пакет .NET Framework (C#, Visual Basic.NET, і JScript.NET). ASP.NET має перевагу у швидкості в порівнянні зі скриптовими технологіями, тому що при першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію, і так далі.

2.2 ASP.NET Web API. Опис та основні можливості

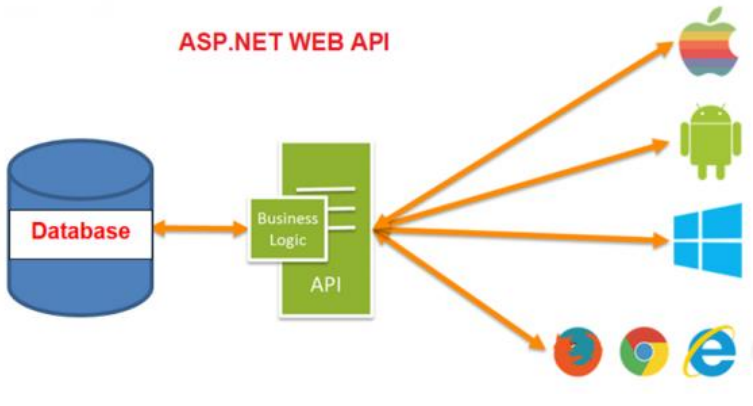

Рисунок 2.3 – Принцип роботи технології ASP.NET Web API

Засіб Web API засноване на додаванні в додаток ASP.NET MVC Framework контролера спеціального виду. Цей різновид контролерів, яка називається контролером API, володіє двома характеристиками:

1. Методи дій повертають об'єкти моделей, а не об'єкти типу ActionResult.
2. Методи дій вибираються на основі HTTP-методу, використовуваного в запиті.

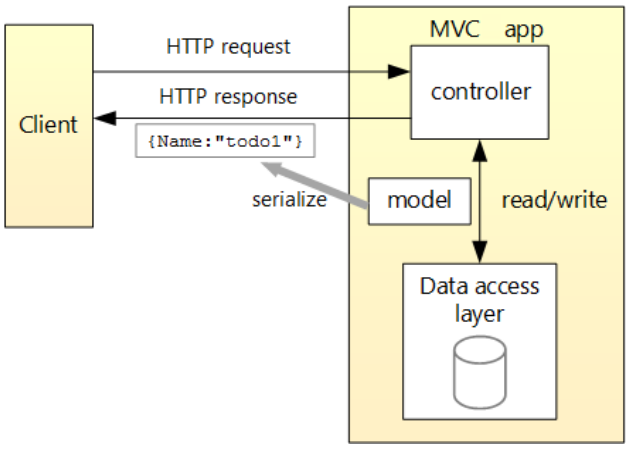

Рисунок 2.4 – Структура додатку, написаного за допомогою технології ASP.NET Web API

Об'єкти моделей, які повертаються методом дії контролера API, кодуються в форматі JSON і відправляються клієнту. Контролери API призначені для доставки веб-служб даних, тому вони не підтримують уявлення, компонування або будь-які інші засоби, які застосовувалися для генерації HTML-розмітки в прикладі створеного додатки.

Відсутність можливості у контролера API генерувати HTML-розмітку з уявлень є причиною, по якій в односторінкових додатках комбінуються стандартні прийоми ASP.NET MVC Framework з Web API. Інфраструктура ASP.NET MVC Framework виконує кроки, необхідні для доставки HTML-вмісту користувачеві (включаючи аутентифікацію, авторизацію, вибір і візуалізацію уявлення). Після того, як HTML-вміст доставлено в браузер, запити Ajax, що генеруються містяться всередині кодом JavaScript, будуть оброблятися контролером Web API.

В звичайних контролерах можна створювати методи дій, які повертають дані JSON для підтримки Ajax, але контролер API пропонує альтернативний підхід. Цей підхід передбачає відокремлення дій, що відносяться до даних, від дій, пов'язаних з поданням, і робить створення універсальних програм Web API швидким і простим.

2.3 ASP.NET MVC. Опис та основні можливості

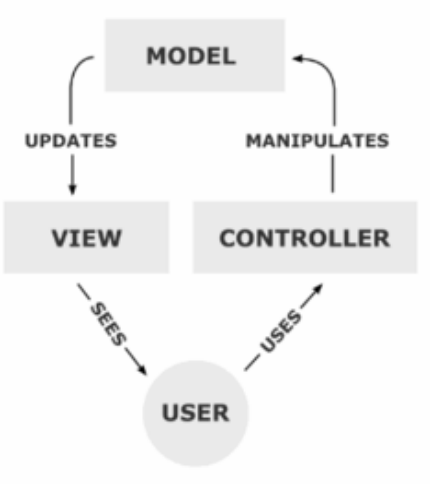

Рисунок 2.5 – Принцип роботи технології ASP.NET MVC

ASP.NET MVC - це структура веб-додатків, розроблена корпорацією Майкрософт, яка реалізує модель моделі-перегляду-контролера (MVC). Це програмне забезпечення з відкритим кодом, крім компонента ASP.NET Web Forms, який є власністю.

У більш пізніх версіях ASP.NET, ASP.NET MVC, ASP.NET Web API і ASP.NET Web Pages (платформа, що використовує тільки сторінки Razor) об'єднаються в уніфікований MVC 6.

На основі ASP.NET, ASP.NET MVC дозволяє розробникам програмного забезпечення створювати веб-додатки як композицію з трьох ролей: Model, View і Controller. Модель MVC визначає веб-додатки з 3 логічними шарами:

1. Модель (бізнес-рівень)
2. Вид (відображувальний шар)
3. Контролер (контроль входу)

Модель представляє стан конкретного аспекту програми. Контролер обробляє взаємодію та оновлює модель, щоб відобразити зміну стану програми, а потім передає інформацію до перегляду. Вид приймає необхідну інформацію від контролера і надає інтерфейс користувача для відображення цієї інформації.

У квітні 2009 року вихідний код ASP.NET MVC був випущений під ліцензією Microsoft Public License.

Каркас ASP.NET MVC - це легка, високотомізована структура презентацій, яка інтегрована з існуючими функціями ASP.NET. Деякі з цих інтегрованих функцій - це сторінки-шаблони та аутентифікація на основі членства. зборів. "

Каркас ASP.NET MVC об'єднує моделі, перегляди та контролери, використовуючи контракти на основі інтерфейсу, тим самим дозволяючи кожному компоненту тестуватися незалежно.

Двигуни перегляду, що використовуються в рамках ASP.NET MVC 3 і MVC 4, є Razor і Web Forms. Обидва двигуни перегляду є частиною системи MVC 3. За замовчуванням, механізм перегляду в рамках MVC використовує сторінки Razors .cshtml і .vbhtml або Web Forms .aspx для розробки макетів сторінок інтерфейсу користувача, на яких складаються дані. Однак можуть використовуватися різні двигуни перегляду. Крім того, замість типової моделі зворотного зв'язку ASP.NET Web Forms, будь-які взаємодії направляються до контролерів за допомогою механізму маршрутизації ASP.NET. Перегляди можуть відображатися на різних URL-адресах.

2.4 ASP.NET Core. Опис, основні можливості та переваги відносно попередніх версій технології ASP.NET

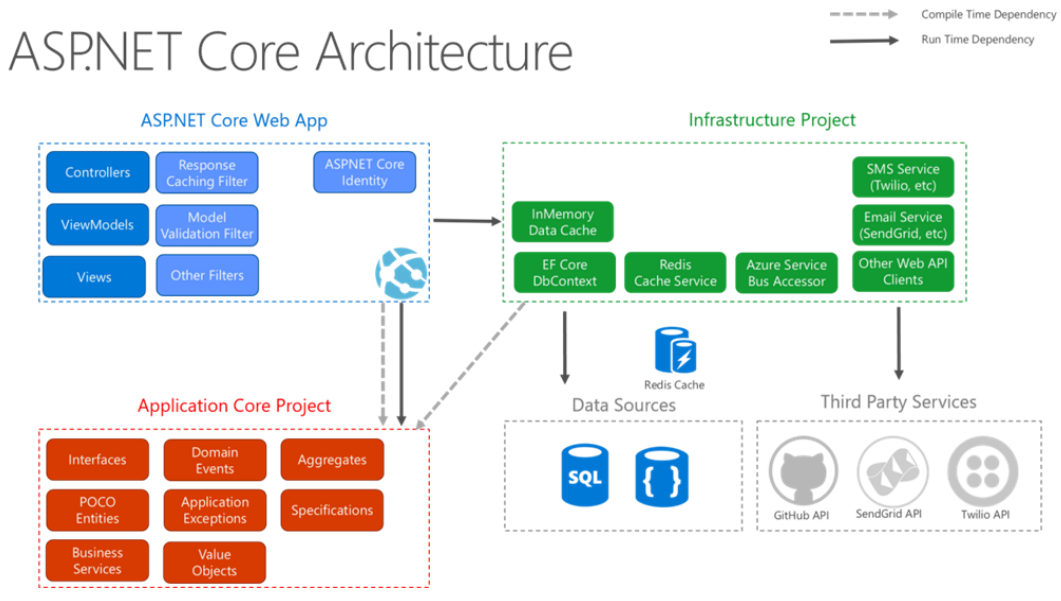

Рисунок 2.6 – Структура додатків, написаних за допомогою технології ASP.NET Core

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. А в травні 2018 року побачила версія ASP.NET Core 2.1, яка власне і охоплена в поточному керівництві.

ASP.NET Core тепер повністю є opensource-фреймворком. Всі вихідні файли фреймворку доступні на GitHub.

ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, яка може бути розгорнута на основних популярних операційних системах: Windows, Mac OS X, Linux. І таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі превалює, але тепер вже ми не обмежені тільки цією операційною системою. Тобто ми можемо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатки можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Хоча ASP.NET Core переважно націлене на використання .NET Core, але фреймворк також може працювати і з повною версією фреймворка .NET.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої

функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле.

Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій.

Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис html з кодом C#.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

В якості інструментарію розробки ми можемо використовувати останні випуски Visual Studio, починаючи з версії Visual Studio 2015. Крім того, ми можемо створювати додатки в середовищі Visual Studio Code, яка є крос-платформної і може працювати як на Windows, так і на Mac OS X і Linux.

Для обробки запитів тепер використовується новий конвеєр HTTP, який заснований на компонентах Katana і специфікації OWIN. А його модульність дозволяє легко додати свої власні компоненти.

2.6 Висновок до розділу 2

В даному розділі було розглянуто основні можливості ASP.NET для побудови додатків. Також було розглянуто версії ASP.NET, такі як ASP.NET Web API, ASP.NET MVC та ASP.NET Core. Окрім можливостей вищезазначених

фреймворків, було розглянуто історії створення даних технологій, їх розвиток та призначення.

Якщо підсумувати, то можна виділити наступні ключові відмінності ASP.NET Core від попередніх версій ASP.NET:

1. Новий легкий і модульний конвеєр HTTP-запитів.
2. Можливість розгортати додаток як на IIS, так і в рамках свого власного процесу.
3. Використання платформи .NET Core і її функціональності.
4. Поширення пакетів платформи через NuGet.
5. Інтегрована підтримка для створення та використання пакетів NuGet.
6. Єдиний стек веб-розробки, що поєднує Web UI і Web API.
7. Конфігурація для спрощеного використання в хмарі.
8. Вбудована підтримка для впровадження залежностей.
9. Можливість розширення.
10. Кросплатформеність: можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux.
11. Розвиток як open source, відкритість до змін.

Ці та інші особливості і можливості стали основою для нової моделі програмування.

3. МЕТОДИКА НАПИСАННЯ ПРОГРАМ ЗА ДОПОМОГОЮ ASP.NET CORE ТА ТЕХНОЛОГІЙ ДОСТУПУ ДО БАЗ ДАНИХ

ASP.NET Core – обширний фреймворк, в даний фреймворк входить дуже багато бібліотек, а також доступні і бібліотеки-розширення, створені програмістами для використання в своїх проектах, що є у вільному доступі. Тобто інфраструктура даного фреймворка постійно розширюється та вдосконалюється. Проте, при описі певної технології потрібно дивитися не лише на його інфраструктуру, а й на його зручність для розробника, поріг входження. Саме для цього і призначений даний розділ – оглянути ASP.NET Core з точки зору розробника, тобто, які основні концепції використовуються, та як вони реалізовані.

В даному розділі буде розглянуто основні принципи розробки з використанням технології ASP.NET Core, основна частина якого буде приділена паттернам та принципам програмування, порівняно та описано переваги та недоліки використання технологій доступу до баз даних, таких як Entity Framework Core, ADO.NET та Dapper.

3.1 Основні рекомендації для написання додатків на ASP.NET Core

ASP.NET Core – це новий фреймворк від Microsoft для розробки Веб додатків, який з'явився внаслідок редизайну раніше існуючого фреймворка ASP.NET MVC. Потрібно розуміти, що ASP.NET Core не обов'язково повинен базуватися на .NET Core. Можна створити ASP.NET Core додаток на основі .NET.

Кожен великий проект використовує паттерни для кращої структуризації коду та його підтримки в майбутньому. Паттерн – це “рекомендація” розробникам для написання якісного коду, тобто це не панацея і будь-який розробник в праві відмовитися від використання паттернів.

Розглянемо основні із них, що використовуються при написанні програм з використанням технології ASP.NET Core:

1. Розділення структури додатку на шари, використання так званої багатошарової архітектури (n-tier architecture).
2. Використання паттерна MVC(модель, представлення, контроллер), даний паттерн використовується ще й у веб-додатках для розшарування логіки, графічного інтерфейсу та доступу до даних;
3. Використання паттерна DI (вставка залежностей), класи в програмі не повинні залежати від конкретної реалізації, а від інтерфейсів, а самі залежності повинен вставляти контейнер;
4. Репозиторій(repository) – даний патерн використовується як метод доступу до об‘єктів бази даних.
5. Використання принципів SOLID.

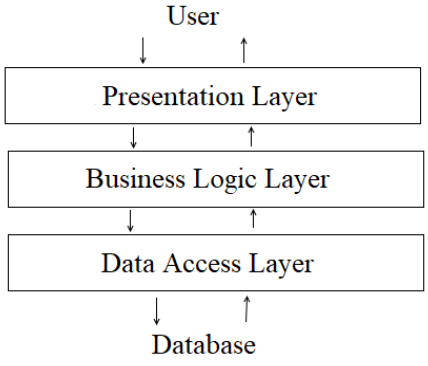

Рисунок 3.1 – Приклад багаторівневої архітектури

Багаторівнева архітектура – клієнт-серверна архітектура, в якій розділяються функції представлення, обробки і зберігання даних. Найбільш поширеною різновидом багаторівневої архітектури є трирівнева архітектура.

N-рівнева архітектура додатка надає модель, по якій розробники можуть створювати гнучкі і повторно-використовуваних програм. Поділяючи додаток на рівні абстракції, розробники набувають можливість внесення змін в якийсь певний шар, замість того, щоб переробляти все додаток цілком. Трирівнева

архітектура зазвичай складається з рівня уявлення, рівня бізнес логіки і рівня зберігання даних.

Хоча поняття шару і рівня найчастіше використовуються як взаємозамінні, багато сходяться на думці, що між ними все-таки є відмінність. Різниця полягає в тому, що шар - це механізм логічного структурування компонентів, з яких складається програмне рішення, в той час як рівень - це механізм фізичного структурування інфраструктури системи. Трьохрівневе рішення легко може бути розгорнуто на єдиному рівні, такому як персональна робоча станція.

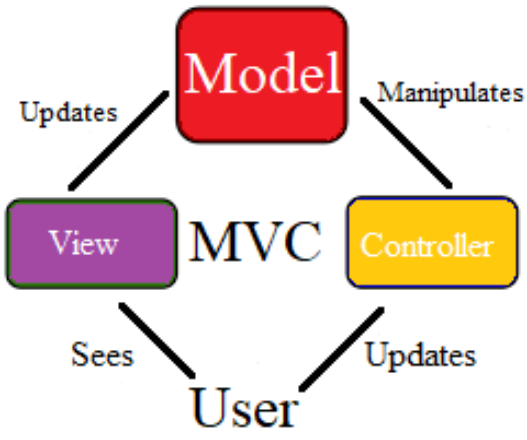

Рисунок 3.2 – Опис паттерна MVC

Цей паттерн передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

Інжекція залежностей - це метод, за допомогою якого один об'єкт (або статичний метод) постачає залежності іншого об'єкта. Залежність - це об'єкт, який можна використовувати (служба). Ін'єкція - це передача залежності на залежний об'єкт (клієнт), який буде використовувати його. Послуга входить до складу держави клієнта. Передача послуги клієнту, а не дозвіл клієнту побудувати або знайти послугу, є основною вимогою шаблону.

Метою ін'єкції залежностей є досягнення розділення проблем будівництва та використання об'єктів.

Ін'єкція залежності є однією з форм більш широкої техніки інверсії контролю. Як і в інших формах інверсії контролю, інжекція залежностей підтримує принцип інверсії залежностей. Клієнт делегує відповідальність за надання своїх залежностей зовнішньому коду (інжектору). Клієнту не дозволяється викликати інжекторний код, це ін'єкційний код, який конструює служби і викликає клієнта, щоб ввести їх. Це означає, що клієнтський код не повинен знати про код ін'єкції, про те, як побудувати послуги або навіть які фактичні послуги він використовує; клієнт повинен знати лише про внутрішні інтерфейси послуг, оскільки вони визначають, як клієнт може користуватися послугами. Це розділяє обов'язки використання та будівництва.

Існує три загальні засоби для клієнта для прийняття ін'єкції залежностей: інжекція засновника, інтерфейсу та конструктора. Інжектор сетера і конструктора відрізняється головним чином, коли вони можуть бути використані. Інжекція інтерфейсу відрізняється тим, що дається можливість контролювати власну ін'єкцію. Кожен з них вимагає, щоб окремий код будівництва (інжектор) брав на себе відповідальність за введення клієнта та його залежності один від одного.

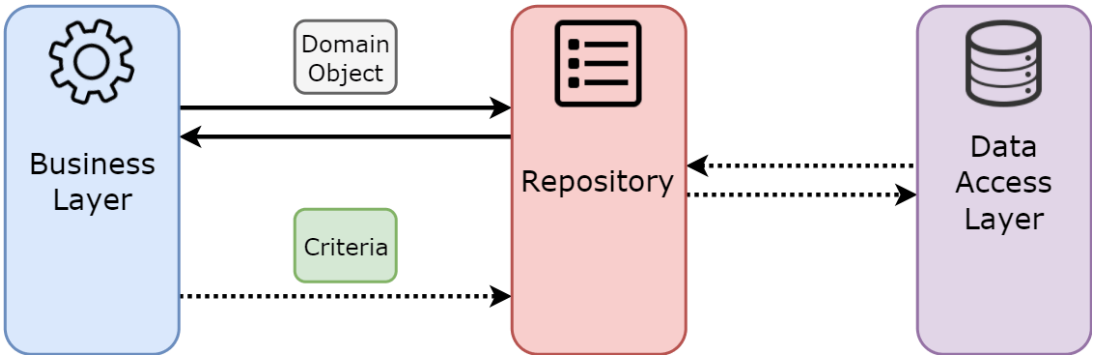

Рисунок 3.3 – Призначення паттерна Repository

Посередником між рівнями області визначення і розподілу даних (domain and data mapping layers), використовуючи інтерфейс, схожий з колекціями для доступу до об'єктів області визначення.

Система зі складною моделлю області визначення може бути спрощена за допомогою додаткового рівня, наприклад Data Mapper, який би ізолював об'єкти від коду доступу до БД. В таких системах може бути корисним додавання ще одного шару абстракції поверх шару розподілу даних (Data Mapper), в якому б був зібраний код створення запитів. Це стає ще більш важливим, коли в області визначення безліч класів або при складних, важких запитах. У таких випадках додавання цього рівня особливо допомагає скоротити дублювання коду запитів.

Патерн Repository є посередником між шаром області визначення і шаром розподілу даних, працюючи, як звичайна колекція об'єктів області визначення. Об'єкти-клієнти створюють опис запиту декларативно і направляють їх до об'єкта-сховища (Repository) для обробки. Об'єкти можуть бути додані або видалені з сховища, як ніби вони формують просту колекцію об'єктів. А код розподілу даних, прихований в об'єкті Repository, подбає про відповідних операціях в непомітно для розробника. У двох словах, патерн Repository инкапсулирует об'єкти, представлениие в сховище даних і операції, вироблені над ними, надаючи більш об'єктно-орієнтоване уявлення реальних даних. Repository також має на меті досягнення повного поділу і односторонньої залежності між рівнями області визначення і розподілу даних.

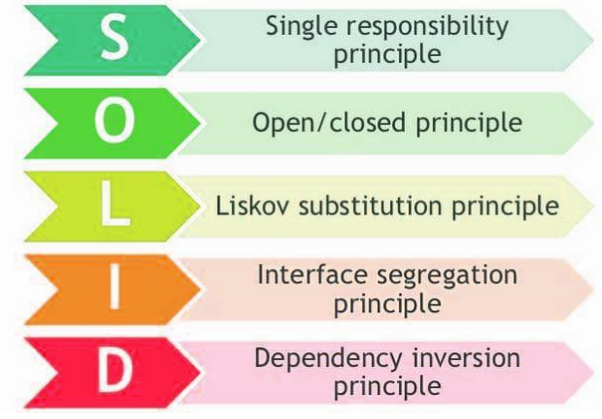

Рисунок 3.4 – Принципи SOLID

SOLID – це абревіатура складена з перших літер п’яти базових принципів об’єктно-орієнтованого програмування дизайну, запропонована Робертом Мартіном.

Принципи SOLID використовуються для дизайну та розробки таких програмних систем, які, з великою ймовірністю, зможуть тривалий час розвиватися, розширятися і підтримуватися.

Принцип єдиного обов'язка (Single Responsibility Principle) можна сформулювати так: у класу повинна бути тільки одна причина для зміни. Під обов'язком тут розуміється набір функцій, які виконують єдине завдання. Суть цього принципу полягає в тому, що клас повинен виконувати одну єдину задачу. Весь функціонал класу повинен бути цілісним, володіти високою зв'язністю (high cohesion). Конкретне застосування принципу залежить від контексту. В даному випадку важливо розуміти, як змінюється клас. Якщо клас виконує кілька різних функцій, і вони змінюються окремо, то це якраз той випадок, коли можна застосувати принцип єдиної обов'язки. Тобто іншими словами, у класу кілька причин для зміни.

Принцип відкритості/закритості (Open/Closed Principle) можна сформулювати так: об’єкти програми повинні бути відкриті для розширення, але закриті для зміни. Суть цього принципу полягає в тому, що система повинна бути

побудована таким чином, що всі її подальші зміни повинні бути реалізовані за допомогою додавання нового коду, а не зміни вже існуючого.

Принцип підстановки Лісков (Liskov Substitution Principle) являє собою деякий керівництво по створенню ієрархій успадкування. Повинна бути можливість замість базового типу підставити будь-який його підтип. Фактично принцип підстановки Лісков допомагає чіткіше сформулювати ієрархію класів, визначити функціонал для базових і похідних класів і уникнути можливих проблем при застосуванні поліморфізму.

Принцип поділу інтерфейсів (Interface Segregation Principle) відноситься до тих випадків, коли класи мають "жирний інтерфейс", тобто занадто роздутий інтерфейс, не всі методи і властивості якого використовуються і можуть бути затребувані. Таким чином, інтерфейс вийдуть занадто надмірний або "жирним". Принцип поділу інтерфейсів можна сформулювати так: клієнти не повинні вимушено залежати від методів, якими не користуються. При порушенні цього принципу клієнт, який використовує певний інтерфейс з усіма його методами, залежить від методів, якими не користується, і тому виявляється сприйнятливий до змін в цих методах. У підсумку ми приходимо до жорсткої залежності між різними частинами інтерфейсу, які можуть бути не пов'язані при його реалізації. В цьому випадку інтерфейс класу поділяється на окремі частини, які становлять окремі інтерфейси. Потім ці інтерфейси незалежно один від одного можуть застосовуватися і змінюватися. В результаті застосування принципу поділу інтерфейсів робить систему слабосвязанной, і тим самим її легше модифікувати і оновлювати.

Принцип інверсії залежностей (Dependency Inversion Principle) служить для створення слабосвязанних сутностей, які легко тестувати, модифікувати і оновлювати. Цей принцип можна сформулювати наступним чином: модулі верхнього рівня не повинні залежати від модулів нижнього рівня. І ті й інші повинні залежати від абстракцій. Абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

3.2 Технологія для доступу до баз даних Entity Framework Core

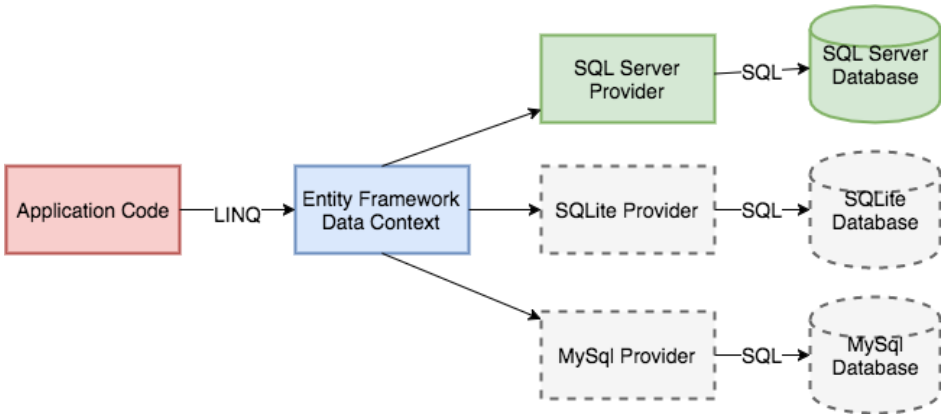

Рисунок 3.5 – Структура роботи технології EF Core

Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну і расширяемую технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

За замовчуванням на даний момент Microsoft надає ряд вбудованих провайдерів: для роботи з MS SQL Server, для SQLite, для PostgreSQL. Також є провайдери від сторонніх постачальників, наприклад, для MySQL.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо, наприклад, ми вирішимо змінити цільову СУБД, то основні зміни в проекті будуть стосуватися насамперед конфігурації і настройки підключення до відповідних провайдерам. А код, який безпосередньо працює з даними, отримує дані, додає їх в БД і т.д., залишиться колишнім.

Entity Framework Core багато успадкував від своїх попередників, зокрема, Entity Framework 6. У той же час треба розуміти, що EF Core - це не нова версія по відношенню до EF 6, а зовсім інша технологія, хоча в цілому принципи роботи у них будуть збігатися. Тому в рамках EF Core використовується своя система версій. Поточна версія - 2.0 була випущена в серпні 2017 року. І технологія продовжує розвиватися. Що вже є в EF Core, а що тільки планується додати, можна подивитися в роудмапе на гітхабе.

Як технологія доступу до даних Entity Framework Core може використовуватися на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні додатки, WPF, ASP.NET 4.6 / 4.7. Це і нові технології як UWP і ASP.NET Core. При цьому кроссплатформенная природа EF Core дозволяє задіяти її не тільки на ОС Windows, але і на Linux і Mac OS X.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людини, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік. Властивості необов'язково представляють прості дані типу int або string, але можуть також представляти і більш комплексні типи даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому суті можуть бути пов'язані асоціативної зв'язком один-ко-многим, один-ко-одному і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework Core, як технології ORM, є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо створювати різні запити на вибірку об'єктів, в тому числі пов'язаних різними асоціативними зв'язками. А Entity Framework при виконання запиту

транслює вираження LINQ в вирази, зрозумілі для конкретної СУБД (як правило, в вирази SQL).

3.3 Технологія для доступу до баз даних ADO.NET

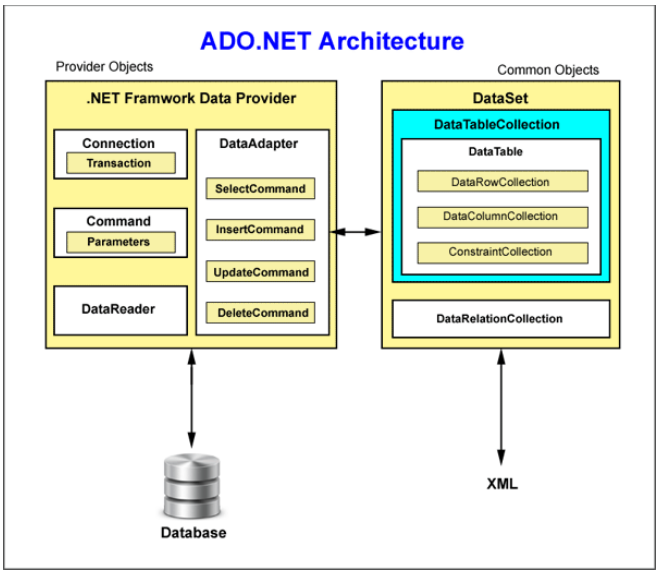

Рисунок 3.6 – Структура технології ADO.NET

Сьогодні велике значення має робота з даними. Для зберігання даних використовуються різні системи управління базами даних: MS SQL Server, Oracle, MySQL і так далі. І більшість великих додатків так чи інакше використовують для зберігання даних ці системи управління базами даних. Однак щоб здійснювати зв'язок між базою даних і додатком на C # необхідний посередник. І саме таким посередником є технологія ADO.NET.

ADO.NET надає собою технологію роботи з даними, яка заснована на платформі .NET Framework. Ця технологія являє нам набір класів, через які ми можемо відправляти запити до баз даних, встановлювати підключення, отримувати відповідь від бази даних і виробляти ряд інших операцій.

Причому важливо зазначити, що систем управління баз даних може бути безліч. У своїй сутності вони можуть відрізнятися. MS SQL Server, наприклад, для створення запитів використовує мову T-SQL, а MySQL і Oracle застосовують мову PL-SQL. Різні системи баз даних можуть мати різні типи даних. Також можуть

відрізнятися якісь інші моменти. Однак функціонал ADO.NET побудований таким чином, щоб надати розробникам уніфікований інтерфейс для роботи з самими різними СУБД.

Основу інтерфейсу взаємодії з базами даних в ADO.NET представляє обмежене коло об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. За допомогою об'єкта Connection відбувається установка підключення до джерела даних. Об'єкт Command дозволяє виконувати операції з даними з БД. Об'єкт DataReader зчитує отримані в результаті запиту дані. Об'єкт DataSet призначений для зберігання даних з БД і дозволяє працювати з ними незалежно від БД. І об'єкт DataAdapter є посередником між DataSet і джерелом даних. Головним чином, через ці об'єкти і буде йти робота з базою даних.

Однак щоб використовувати один і той же набір об'єктів для різних джерел даних, необхідний відповідний провайдер даних. Власне через провайдер даних в ADO.NET і здійснюється взаємодія з базою даних. Причому для кожного джерела даних в ADO.NET може бути свій провайдер, який власне і визначає конкретну реалізацію вищевказаних класів.

За замовчуванням в ADO.NET є наступні вбудовані провайдери:

1. Провайдер для MS SQL Server
2. Провайдер для OLE DB (Надає доступ до деяких старих версій MS SQL Server, а також до БД Access, DB2, MySQL і Oracle)
3. Провайдер для ODBC (провайдер для тих джерел даних, для яких немає своїх провайдерів)
4. Провайдер для Oracle
5. Провайдер EntityClient. Провайдер даних для технології ORM Entity Framework
6. Провайдер для сервера SQL Server Compact 4.0

Крім цих провайдерів, які є вбудованими, існує також безліч інших, призначених для різних баз даних, наприклад, для MySQL.

Функціонально класи ADO.NET можна розбити на два рівня: підключений і відключений. Кожен провайдер даних .NET реалізує свої версії об'єктів

Connection, Command, DataReader, DataAdapter і ряду інших, який складають підключений рівень. Тобто за допомогою них встановлюється підключення до БД і виконується з нею взаємодію. Як правило, реалізації цих об'єктів для кожного конкретного провайдера в своїй назві мають префікс, який вказує на провайдер.

Інші класи, такі як DataSet, DataTable, DataRow, DataColumn і ряд інших складають відключений рівень, так як після отримання даних в DataSet ми можемо працювати з цими даними незалежно від того, чи встановлено підключення чи ні. Тобто після отримання даних з БД додаток може бути відключено від джерела даних.

3.4 Технологія для доступу до баз даних Dapper

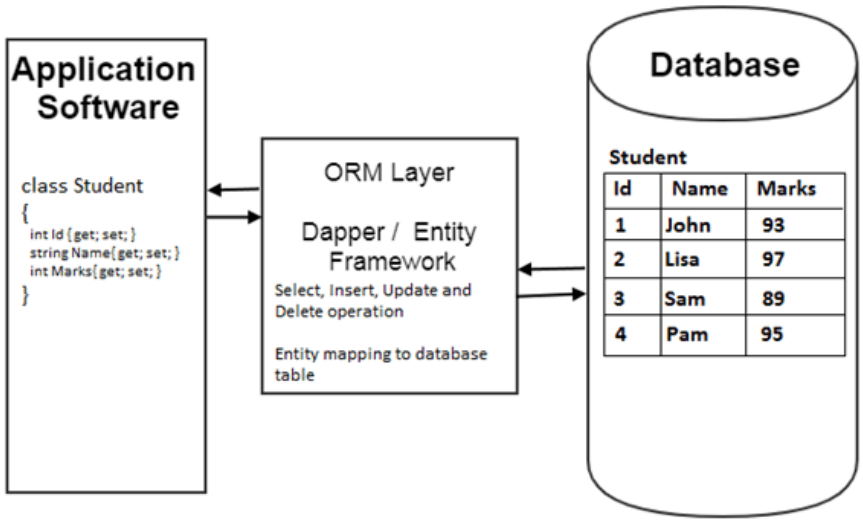

Рисунок 3.7 – Структура роботи технології Dapper

Dapper - це продукт об'єктно-реляційного відображення (ORM) для платформи Microsoft .NET: він забезпечує структуру для відображення об'єктно-орієнтованої моделі домену до традиційної реляційної бази даних. Її мета полягає в тому, щоб звільнити розробника від значної частини завдань програмування, пов'язаних з реляційними даних. Dapper є вільним, як програмне забезпечення з відкритим вихідним кодом, яке поширюється під двою ліцензією - ліцензією Apache 2.0 або ліцензією MIT.

Щоб працювати зі збереженими процедурами за допомогою Dapper, слід вказувати тип команди явно при виклику методів Query або Execute.

Dapper micro ORM надзвичайно легкий і простий у використанні. Він не генерує SQL, але полегшує відображення результатів запитів до ваших POCO (звичайні старі об'єкти CLR). Можна отримати набагато більшу швидкість виконання, ніж у Entity Framework - практично так само, як і ADO.NET.

3.5 Висновок до розділу 3

В даному розділі було розглянуто основні принципи розробки програмного забезпечення з використанням ASP.NET Core, проте, на відміну від минулого розділу, дані підходи було розглянуто не з технічної точки зору. Було показано, що використання ASP.NET Core є більш вигідним при побудові комплексних додатків.

Також було розглянуто основні рекомендації для структурування проекту та паттерни, які використовуються при розробці додатків. Було розглянуто основні паттерни та принципи, що використовуються в ASP.NET Core додатках, а також обґрунтовано їх використання в деяких ситуаціях. Описано принцип поділу проекту на шари, та показано, що додаток не обов’язково повинен мати усіх їх, адже в деяких ситуаціях вони просто не потрібні чи можуть бути об’єданані в один.

Ще однією розкритою темою даного розділу є використання технологій для доступу до баз даних, такі як EF Core, ADO.NET та Dapper. Можна використовувати будь-яку з цих технологій, проте у кожної з них є свої переваги та недоліки. Entity Framework Core зручніший у використанні, а ADO.NET та Dapper працює швидше.

4. РОЗРОБКА ТА АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Основні поняття

4.1.1 REST та Restful

REST (Representational State Transfer - «передача стану уявлення») - архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST є узгодженим набіром обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи. У певних випадках (інтернет-магазини, пошукові системи, інші системи, засновані на даних) це призводить до підвищення продуктивності і спрощення архітектури.

В мережі Інтернет виклик віддаленої процедури може являти собою звичайний HTTP-запит (зазвичай «GET» або «POST»; такий запит називають «REST запит»), а необхідні дані передаються в якості параметрів запиту. Для веб-служб, побудованих з урахуванням REST, застосовують термін «RESTful».

4.1.2 Nlog

Nlog – це гнучка і безкоштовна платформа для ведення журналів для різних платформ .NET, включаючи стандарт .NET. NLog дозволяє легко записувати до кількох цілей (база даних, файл, консоль) і зміна конфігурації реєстрації на льоту.

NLog підтримує структурований і традиційний журнал. Основна увага для NLog: висока продуктивність, проста у використанні, легко розширюється і гнучка для налаштування.

4.1.3 DAL

Доступ до даних (Data Access Layer - DAL) в програмному забезпеченні - це обчислювальна програма, яка забезпечує упроваджений доступ до даних, що

зберігаються в постійному резервуарі для какого-небудь типу, як реляційна база даних. Цей акроним в основному використовується в Microsoft ASP.NET.

Наприклад, DAL може повертати посилання на об'єкт (в терміналах об'єктивно-орієнтованого програмування) з його атрибутами, а не строк накладання даних. Це дозволяє створювати кліентські (або користувацькі) модулі з більш високим рівнем абстракції. Такая рода модель може бути реалізована шляхом створення класу з методами доступу до даних, які використовуються на відповідний набір процедур баз даних. Другая реалізація може потенційно отримувати або записувати записи в або з файлової системи. DAL приховує складність лежачого в основі зберігання даних від зовнішнього світу.

У такому вигляді, як "створити", "вилучити" або "оновити" в певній таблиці, клас і кілька процедур можуть бути створені в базах. Ці процедури можуть викликатися з методу всередині класу, який повертається об'єктом, що містить запитувані значення. Або ж створені, усунені і оновлені можуть бути виконані всередині простого функцій як registerUser або loginUser, що зберігається в доступному вигляді.

Крім того, можуть бути запропоновані бізнес-логіки з додатками. Наприклад, створення запиту до баз даних, щоб отримати всіх користувачів з декількох таблиць, додаток може зробити один вибір методу з DAL для даного виду.

4.1.4 BLL

Business Logic Layer або бізнес-рівень інкапсулює всю бізнес-логіку, всі необхідні обчислення, отримує об'єкти з рівня доступу до даних і передає їх на рівень представлення, або, навпаки, отримує дані з рівня уявлення і передає їх на рівень даних.

Рівень представлення не може безпосередньо отримувати дані з бази даних. В даному випадку BLL виступатиме в ролі посередника між двома рівнями. Але також треба враховувати, що безпосередньо він не може передавати в контролери

об'єкти Phone і Order, так як рівень представлення не повинен мати доступ до функціональності рівня DAL. Тому нам потрібні проміжні сутності.

4.1.5 MemoryCache

Кешування є збереження даних в спеціальному місці для більш швидкого доступу до них в майбутньому. Застосування кешування може значно підвищити продуктивність програми ASP.NET, істотно зменшуючи кількість звернень до джерел даних, наприклад, до баз даних. Особливо ефективно кешування в тих випадках, коли у нас є на веб-сторінці деякі елементи, дані яких рідко змінюються або змінюються через певний проміжок часу.

Кешування в ASP.NET Core побудовано навколо залежності Microsoft.Extensions.Caching.Memory і передбачає використання вбудованого інструменту кешування - об'єкта IMemoryCache.

У сфері обчислювальної обробки даних кеш - це високошвидкісний рівень зберігання, на якому необхідний набір даних, як правило, тимчасового характеру. Доступ до даних на цьому рівні здійснюється значно швидше, ніж до основного місця їх зберігання. За допомогою кешування стає можливим ефективне повторне використання раніше отриманих або обчислених даних.

Дані в кеші зазвичай зберігаються на пристрої з швидким доступом, такому як ОЗУ (оперативне запам'ятовуючий пристрій), і можуть використовуватися спільно з програмними компонентами. Основна функція кешу - прискорення процесу вилучення даних. Він позбавляє від необхідності звертатися до менш швидкісного базового рівня зберігання.

Невеликий обсяг пам'яті кешу компенсується високою швидкістю доступу. У кеші зазвичай зберігається тільки необхідний набір даних, причому тимчасово, на відміну від баз даних, де дані зазвичай зберігаються повністю і постійно.

4.2 Обгрунтування вибраних платформ, мов реалізації, фреймворків та технології для доступу до баз даних

1. REST архітектура більш за все підходить для поставленої задачі, тому що спроектована для передачі стану системи. Зважаючи на це, сервіс буде надавати REST API.
2. C# – це мова програмування, що була вибрана для написання соціальної мережі для навчального закладу. Вона дозволяє за мінімальний час реалізувати необхідний функціонал, зручний для побудови серверних додатків.
3. ASP.NET Core – це .NET фреймворк, призначений для створення веб-сервісів, він є новою епохою в розвитку технології ASP.NET, простий в освоєнні.
4. Microsoft SQL Server Management Studio – це багатофункціональна СУБД, призначена для управління та конфігурування компонентів Microsoft SQL Server.
5. Entity Framework Core – це технологія для доступу до баз даних, зручна у використанні. Основною перевагою EF Core є універсальний API для роботи з даними.
6. AutoMapper – це утиліта, яка дозволяє спроектувати одну модель на іншу для скорочення об’єму кода та спрощення системи.
7. Nlog – це дуже зручна платформа для запису до файлу інформації про роботу програми, легка у підключенні та використанні.
8. ASP.NET Core Identity – зручна система аутентифікації та авторизації. Дана система дозволяє користувачам створювати облікові записи, управляти ними та використовувати для входу на сайт облікові записи інших провайдерів.

4.3 Результати роботи

У результаті був отриманий програмний продукт – соціальна мережа для навчального закладу. Назва системи – “KPI Network”.

4.3.1 Соціальна мережа “KPI Network”

Соціальна мережа “KPI Network” заснована на технології REST, тобто він є Restful веб-сервісом.

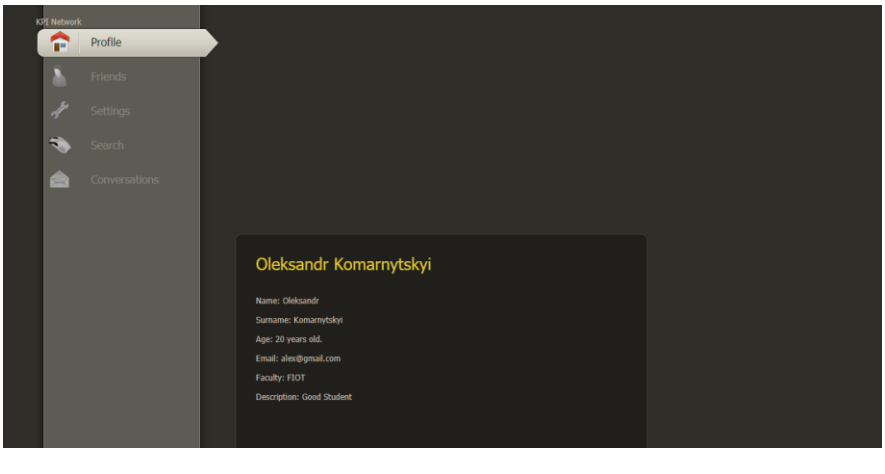

Рисунок 4.1 – Користувацький інтерфейс розробленої соціальної мережі

Він реалізує:

1. Пошук користувачів по:
   1.1 Імені.
   1.2 Прізвищу.
   1.3 Факультету.
   1.4 Віку.
2. Додання користувачів до списку “Друзі”.
3. Видалення користувачів зі списку “Друзі”.
4. Фільтрація користувачів за:
   4.1 Іменем.

4.2 Прізвищем.

4.3 Факультетом.

4.4 Віком.

5. Створення бесід між користувачами.
6. Додання користувачів в бесіди.
7. Видалення участників з бесід користувачами з спеціальними можливостями.
8. Покидання бесід участниками.
9. Написання повідомлень участниками бесід.
10. Видалення повідомлень участниками бесід.
11. Створення облікових записів.
12. Можливість редагування користувачем власного облікового запису.

Інформація про користувача надається у форматі відповіді JSON:

1. Id – це унікальний ідентифікатор користувача в системі
2. FirstName – ім’я користувача.
3. LastName – прізвище користувача.
4. Age – вік користувача.
5. Faculty – факультет користувача.
6. Friends – список друзів користувача.
7. Description – опис користувача.

Приклад відповіді в якості користувача у форматі JSON:

```
{
    id: 111,
    firstname: “Oleksandr”,
    lastname: “Komarnytskyi”,
    age: 20,
    faculty: “FIOT”,
    friends: [],
    description: “Good student”
```

}

4.4 Варіанти подальшого розвитку роботи

Подальший розвиток роботи складається як з розширення вже реалізованих можливостей, так і додавання нових:

1. Можливість додавання фотографій, відеофайлів та текстових документів.
2. Можливість ставити на аватарку фотографії.
3. Зберігання пошукових запитів користувача та створення рекомендацій на їх основі.
4. Створення груп.
5. Додавання нових параметрів для фільтрації та сортування користувачів, друзів та бесід.

4.5 Висновок до розділу 4

Оцінивши час, наданий на реалізацію дипломної роботи, була проведена декомпозиція завдань та складений план роботи. В результаті отримав програмний продукт, соціальну мережу для навчального закладу “KPI Network”, що готовий до випуску.

ВИСНОВКИ

У даній дипломній роботі було реалізовано соціальну мережу для навчального закладу з використанням мови C#, технологій ASP.NET Core та Entity Framework Core. Було спроектовано багатошарову архітектуру та використано основні можливості вищезазначених фреймворків.

У першому розділі було розглянуто основні поняття соціальної мережі та дано характеристику сучасним популярним соціальним мережам, таким як Facebook, Twitter, LinkedIn. Розглянуто історію створення, розвиток та структуру вищезазначених соціальних мереж. Зроблено висновок, що соціальні мережі були створені з абсолютно різним призначенням.

У другому розділі розглянуто ASP.NET як один з найпопулярніших фреймворків для створення веб-сервісів. Також було розглянуто версії ASP.NET, їх призначення, перееваги та недоліки. Було зроблено висновок, що найкраще за все використовувати технологію ASP.NET Core, адже вона є зручнішою за своїх попередників і навіть об’єднує в собі попередні версії технології ASP.NET.

У третьому розділі було розглянуто основні принципи побудови додатку за допомогою ASP.NET Core та різних технологій для доступу до баз даних, переваги та недоліки з точки зору розробки. Також показано основні принципи та патерни, що варто використовувати при написанні додатків, описано основні відмінності між технологіями для доступу до баз даних, такими як Entity Framework Core, ADO.NET та Dapper. Було зроблено висновок, що ADO.NET та Dapper більш продуктивніші, проте EF Core набагато легший в розумінні та зручніший у взаємодії з ASP.NET Core.

В останньому розділі було описано реалізацію соціальної мережі, деякі аспекти виконання, функції, які даний додаток може виконувати та різні доповнення, які згодом можна було б додати. Також було розглянуто можливий подальший розвиток соціальної мережі. Також було описано та обгрунтовано, які технології, мови та платформи використовувались в розробці даної соціальної мережі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Лямин А.В. Использование социальных сетей в образовании. – Санкт-Петербург, 2015. – 64с.
2. Які соціальні мережі популярні у світі. – Режим доступу: https://marketer.ua/ua/top-social-media-2018/. – Дата доступу: 15.05.2019.
3. Архитектура гигантов. Стек данных в Facebook, Netflix, Airbnb и Pinterest. – Режим доступу: https://apptractor.ru/info/articles/arhitektura-gigantov-stek-dannyih-v-facebook-netflix-airbnb-i-pinterest.html. – Дата доступу: 15.05.2019.
4. Использование Web API. – Режим доступу: https://professorweb.ru/my/ASP_NET/mvc/level8/8_2.php. Дата доступу: 16.05.2019.
5. Facebook. – Режим доступу: http://igroup.com.ua/seo-articles/facebook/. – Дата доступу: 16.05.2019.
6. Mugilan T. S. Ragupathi. ASP.NET Core: Cloud-ready, Enterprise Web Application Development. – Packt, June 2017. 1047c.
7. Введение в ASP.NET Core. – Режим доступу: https://metanit.com/sharp/aspnet5/1.1.php. – Дата доступу: 17.05.2019.
8. Путь ASP.NET Core. – Режим доступу: https://habr.com/ru/post/312226/. – Дата доступу: 17.05.2019.
9. Принципы SOLID. – Режим доступу: https://metanit.com/sharp/patterns/5.1.php. – Дата доступу: 17.05.2019.
10. Jon P. Smith. Entity Framework Core in Action. – Packt, July 2018. 520c.
11. Entity Framework Core documentation. – Режим доступу: https://docs.microsoft.com/en-us/ef/core/. – Дата доступу: 17.05.2019
12. Введение в Entity Framework Core. – Режим доступу: https://metanit.com/sharp/entityframeworkcore/1.1.php. – Дата доступу: 17.05.2019.

13. Entity Framework Core 2 Vs Dapper Performance Benchmark. – Режим доступу: https://medium.com/@engr.mmohsin/entity-framework-core-2-dapper-performance-benchmark-c29e8cce9e1b. – Дата доступу: 17.05.2019.
14. Введение в ADO.NET. – Режим доступу: https://metanit.com/sharp/adonet/1.1.php. – Дата доступу: 17.05.2019.
15. ADO.NET Overview. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview. – Дата доступу: 17.05.2019.
16. How to use the Dapper ORM in C#. – Режим доступу: https://www.infoworld.com/article/3025784/how-to-work-with-dapper-in-c.html. – Дата доступу: 17.05.2019.
17. Business Logic Layer. – Режим доступу: https://metanit.com/sharp/mvc5/23.8.php. – Дата доступу: 17.05.2019.

Додаток А

Лістинг розроблених програм

```
namespace SocialNetwork.API.Controllers
{
    [Route("api/[controller]")]
    public class ConversationsController : Controller
    {
        private readonly IConversationService _conversationService;
        private readonly IMapper _mapper;
        private readonly IConversationResponseCreator
_conversationResponseCreator;

        public ConversationsController(IConversationService conversationService,
IMapper mapper, IConversationResponseCreator
conversationResponseCreator)
        {
            _conversationService = conversationService;
            _mapper = mapper;
            _conversationResponseCreator = conversationResponseCreator;
        }

        [Authorize(Roles = "admin")]
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
```

```
            var conversationDtos = await _conversationService.GetAllAsync();
            var conversationModels =
_mapper.Map<IEnumerable<ConversationModel>>(conversationDtos);
            return Ok(conversationModels);
        }

        [Authorize(Roles = "user")]
        [HttpGet("{id}", Name = "GetConversation")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> Get(int id)
        {
            var conversationDto = await _conversationService.GetAsync(id);
            return
_conversationResponseCreator.ResponseForGet(conversationDto);
        }

        [Authorize(Roles = "user")]
        [HttpPost]
        public async Task<IActionResult> Add([FromBody]
ConversationAddOrUpdateModel conversationAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }
```

```
        var conversationDto =
_mapper.Map<ConversationDtoForCreate>(conversationAddOrUpdateModel);
        var statuscode = await _conversationService.AddAsync(conversationDto);
        return _conversationResponseCreator.ResponseForCreate(statuscode,
conversationDto);
    }

    [Authorize(Roles = "user")]
    [HttpPut]
    public async Task<IActionResult> Update(int id, [FromBody]
ConversationAddOrUpdateModel conversationAddOrUpdateModel)
    {
      if (!ModelState.IsValid)
      {
        return BadRequest();
      }

      var conversationDto =
        _mapper.Map<ConversationDto>(conversationAddOrUpdateModel);
      conversationDto.Id = id;
      var statuscode = await
_conversationService.UpdateAsync(conversationDto);
      return _conversationResponseCreator.ResponseForUpdate(statuscode);
    }

    [Authorize(Roles = "user")]
    [HttpPut("{id}/users/{userId}")]
    public async Task<IActionResult> UpdateUsers(int id, int userId)
```

```
        {
            var statuscode = await _conversationService.UpdateUsersAsync(id,
userId);
            return _conversationResponseCreator.ResponseForUpdate(statuscode);
        }

        [Authorize(Roles = "admin")]
        [HttpDelete]
        public async Task<IActionResult> Delete(int id)
        {
            var statuscode = await _conversationService.DeleteAsync(id);
            return _conversationResponseCreator.ResponseForDelete(statuscode);
        }
    }
}

namespace SocialNetwork.API.Controllers
{
    [Authorize(Roles = "user")]
    [Route("api/[controller]")]
    public class FriendRequestsController : Controller
    {
        private readonly IFriendRequestService _friendRequestService;
        private readonly IMapper _mapper;
        private readonly IFriendRequestResponseCreator
_friendRequestResponseCreator;
```

```
        public FriendRequestsController(IFriendRequestService
    friendRequestService, IMapper mapper, IFriendRequestResponseCreator
    friendRequestResponseCreator)
        {
            _friendRequestService = friendRequestService;
            _mapper = mapper;
            _friendRequestResponseCreator = friendRequestResponseCreator;
        }

        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            var friendRequestDtos = await _friendRequestService.GetAllAsync();
            return
    Ok(_mapper.Map<IEnumerable<FriendRequestModel>>(friendRequestDtos));
        }

        [HttpGet("sender/{senderId}")]
        public async Task<IActionResult> GetBySender(int senderId)
        {
            var friendRequestDtos = await
    _friendRequestService.GetSenderRequestsAsync(senderId);
            return
    Ok(_mapper.Map<IEnumerable<FriendRequestModel>>(friendRequestDtos));
        }

        [HttpGet("receiver/{receiverId}")]
        public async Task<IActionResult> GetByReceiver(int receiverId)
```

```
    {
        var friendRequestDtos = await
_friendRequestService.GetReceiverRequestsAsync(receiverId);
        return
Ok(_mapper.Map<IEnumerable<FriendRequestModel>>(friendRequestDtos));
    }

    [HttpPost]
    public async Task<IActionResult> Add([FromBody] FriendRequestModel
friendRequestModel)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest();
        }

        var friendRequestDto =
_mapper.Map<FriendRequestDto>(friendRequestModel);
        var statuscode = await _friendRequestService.Create(friendRequestDto);
        return
_friendRequestResponseCreator.ResponseForAddFriends(statuscode);
    }

    [HttpPut("accept")]
    public async Task<IActionResult> Accept([FromBody] FriendRequestModel
friendRequestModel)
    {
        if (!ModelState.IsValid)
```

```
        {
            return BadRequest();
        }

        var friendRequestDto =
            _mapper.Map<FriendRequestDto>(friendRequestModel);
        var statuscode = await _friendRequestService.Accept(friendRequestDto);
        return
_friendRequestResponseCreator.ResponseForAcceptOrDeclineOrCreate(statusc
ode);
    }

    [HttpPut("declined")]
    public async Task<IActionResult> Decline([FromBody] FriendRequestModel
friendRequestModel)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest();
        }

        var friendRequestDto =
            _mapper.Map<FriendRequestDto>(friendRequestModel);
        var statuscode = await _friendRequestService.Decline(friendRequestDto);
        return
_friendRequestResponseCreator.ResponseForAcceptOrDeclineOrCreate(statusc
ode);
    }
```

```
    }
}

namespace SocialNetwork.API.Controllers
{
    [Authorize(Roles = "user")]
    [Route("api/[controller]")]
    public class MessagesController : Controller
    {
        private readonly IMessageService _messageService;
        private readonly IMapper _mapper;
        private readonly IMessageResponseCreator _messageResponseCreator;

        public MessagesController(IMessageService messageService, IMapper
mapper, IMessageResponseCreator messageResponseCreator)
        {
            _messageService = messageService;
            _mapper = mapper;
            _messageResponseCreator = messageResponseCreator;
        }
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            var messageDtos = await _messageService.GetAllAsync();
            return
Ok(_mapper.Map<IEnumerable<MessageModel>>(messageDtos));
        }
```

```
[HttpGet("{id}", Name = "GetMessage")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public async Task<IActionResult> Get(int id)
{
   var messageDto = await _messageService.GetAsync(id);
   return _messageResponseCreator.ResponseForGet(messageDto);
}

[HttpGet]
public async Task<IActionResult> GetByConversation(int conversationId)
{
   var messageDtos = await
_messageService.GetByConversationAsync(conversationId);
   return
Ok(_mapper.Map<IEnumerable<MessageModel>>(messageDtos));
}

[HttpPost]
public async Task<IActionResult> Add([FromBody] MessageAddModel
messageAddModel)
{
   if (!ModelState.IsValid)
   {
      return BadRequest();
   }
```

```
        var messageDto = _mapper.Map<MessageDtoForCreate>(messageAddModel);
        var statuscode = await _messageService.AddAsync(messageDto);
        return _messageResponseCreator.ResponseForCreate(statuscode, messageDto);
    }

    [HttpPut]
    public async Task<IActionResult> Update(int id, [FromBody] MessageUpdateModel messageModel)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest();
        }

        var messageDto =
            _mapper.Map<MessageDtoForUpdate>(messageModel);
        messageDto.Id = id;
        var statuscode = await _messageService.UpdateAsync(messageDto);
        return _messageResponseCreator.ResponseForUpdate(statuscode);
    }

    [HttpDelete]
    public async Task<IActionResult> Delete(int id)
    {
        var statuscode = await _messageService.DeleteAsync(id);
        return _messageResponseCreator.ResponseForDelete(statuscode);
```

```
        }
    }
}

namespace SocialNetwork.API.Controllers
{
    [Route("api/[controller]")]
    public class UsersController : Controller
    {
        private readonly IUserService _userService;
        private readonly IMapper _mapper;
        private readonly IUserResponseCreator _userResponseCreator;

        public UsersController(IUserService userService, IMapper mapper,
IUserResponseCreator userResponseCreator)
        {
            _userService = userService;
            _mapper = mapper;
            _userResponseCreator = userResponseCreator;
        }
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            var userDtos = await _userService.GetAllAsync();
            var userModels = _mapper.Map<IEnumerable<UserModel>>(userDtos);
            return Ok(userModels);
        }
```

```
        [HttpGet("{id}", Name = "GetUsers")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public async Task<IActionResult> Get(int id)
        {
            var userDto = await _userService.GetAsync(id);
            return _userResponseCreator.ResponseForGet(userDto);
        }

        [Authorize(Roles = "user")]
        [HttpGet]
        public async Task<IActionResult> GetFriends(int userId)
        {
            var userDtos = await _userService.GetFriendsAsync(userId);
            var userModels = _mapper.Map<IEnumerable<UserModel>>(userDtos);
            return Ok(userModels);
        }

        [Authorize(Roles = "user")]
        [HttpPost]
        public async Task<IActionResult> Add([FromBody] UserAddOrUpdateModel
    userAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }
```

```
            var userDto =
        _mapper.Map<UserDtoForCreate>(userAddOrUpdateModel);
            var statuscode = await _userService.AddAsync(userDto);
            return _userResponseCreator.ResponseForCreate(statuscode, userDto);
        }

        [Authorize(Roles = "user")]
        [HttpPut]
        public async Task<IActionResult> Update(int id, [FromBody]
        UserAddOrUpdateModel userAddOrUpdateModel)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var userDto =
                _mapper.Map<UserDto>(userAddOrUpdateModel);
            userDto.Id = id;
            var statuscode = await _userService.UpdateAsync(userDto);
            return _userResponseCreator.ResponseForUpdate(statuscode);

        }

        [Authorize(Roles = "admin")]
        [HttpDelete]
        public async Task<IActionResult> Delete(int id)
        {
```

```
            var statuscode = await _userService.DeleteAsync(id);
            return _userResponseCreator.ResponseForDelete(statuscode);
        }

        [Authorize(Roles = "user")]
        [HttpDelete("deletefriends")]
        public async Task<IActionResult> DeleteFriends(int userId, int friendId)
        {
            var statuscode = await _userService.DeleteFriends(userId, friendId);
            return _userResponseCreator.ResponseForDeleteFriends(statuscode);
        }
    }
}
namespace SocialNetwork.BLL.Conversations
{
    public class ConversationService : IConversationService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public ConversationService(IUnitOfWork unitOfWork, IMapper mapper)
        {
            _unitOfWork = unitOfWork;
            _mapper = mapper;
        }

        public async Task<IEnumerable<ConversationDto>> GetAllAsync()
        {
```

```
        var conversations = await
_unitOfWork.Conversations.GetAll().ToListAsync();
        return _mapper.Map<IEnumerable<ConversationDto>>(conversations);
    }

    public async Task<ConversationDto> GetAsync(int id)
    {
        var conversation = await _unitOfWork.Conversations.GetAsync(id);
        return _mapper.Map<ConversationDto>(conversation);
    }

    public async Task<int> AddAsync(ConversationDtoForCreate
conversationDtoForCreate)
    {
        if (string.IsNullOrEmpty(conversationDtoForCreate.ConversationName))
        {
            return -1;
        }
        var conversation =
_mapper.Map<Conversation>(conversationDtoForCreate);
        await _unitOfWork.Conversations.AddAsync(conversation);
        await _unitOfWork.SaveChangesAsync();
        return conversation.Id;
    }

    public async Task<int> UpdateAsync(ConversationDto conversationDto)
    {
        if (string.IsNullOrEmpty(conversationDto.ConversationName))
```

```
            {
                return -1;
            }
            if (!await
_unitOfWork.Conversations.ContainsEntityWithId(conversationDto.Id))
            {
                return -2;
            }
            var conversation = await
_unitOfWork.Conversations.GetAsync(conversationDto.Id);
            _mapper.Map(conversationDto, conversation);
            _unitOfWork.Conversations.Update(conversation);
            await _unitOfWork.SaveChangesAsync();
            return conversation.Id;

        }

        public async Task<int> DeleteAsync(int id)
        {
            if (!await _unitOfWork.Conversations.ContainsEntityWithId(id))
            {
                return -1;
            }
            _unitOfWork.Conversations.Delete(id);

            await _unitOfWork.SaveChangesAsync();
            return 1;
        }
```

```
        public async Task<int> UpdateUsersAsync(int id, int userId)
        {
            if (!await _unitOfWork.Conversations.ContainsEntityWithId(id))
            {
                return -3;
            }
            if (!await _unitOfWork.Users.ContainsEntityWithId(id))
            {
                return -4;
            }
            var conversation = await _unitOfWork.Conversations.GetAsync(id);
            conversation.UserConversations.Add(new  UserConversation { UserId =
userId });
            _unitOfWork.Conversations.Update(conversation);
            await _unitOfWork.SaveChangesAsync();
            return 1;
        }
    }
}
namespace SocialNetwork.BLL.FriendRequests
{
    public class FriendRequestService : IFriendRequestService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public FriendRequestService(IUnitOfWork unitOfWork, IMapper mapper)
```

```
        {
            _unitOfWork = unitOfWork;
            _mapper = mapper;
        }

        public async Task<IEnumerable<FriendRequestDto>> GetAllAsync()
        {
            var friendRequests = await
_unitOfWork.FriendRequests.GetAll().ToListAsync();
            return
_mapper.Map<IEnumerable<FriendRequestDto>>(friendRequests);
        }
        public async Task<IEnumerable<FriendRequestDto>>
GetSenderRequestsAsync(int senderId)
        {
            var friendRequests = await
_unitOfWork.FriendRequests.GetSenderRequestsAsync(senderId);
            return
_mapper.Map<IEnumerable<FriendRequestDto>>(friendRequests);
        }

        public async Task<IEnumerable<FriendRequestDto>>
GetReceiverRequestsAsync(int receiverId)
        {
            var friendRequest = await
_unitOfWork.FriendRequests.GetSenderRequestsAsync(receiverId);
            return
_mapper.Map<IEnumerable<FriendRequestDto>>(friendRequest);
```

```
        }

        public async Task<int> Create(FriendRequestDto friendRequestDto)
        {
            if (await
_unitOfWork.FriendRequests.ContainsEntityWithId(friendRequestDto.SenderId
, friendRequestDto.ReceiverId))
            {
                return -1;
            }
            if (!await
_unitOfWork.Users.ContainsEntityWithId(friendRequestDto.SenderId))
            {
                return -2;
            }
            if (!await
_unitOfWork.Users.ContainsEntityWithId(friendRequestDto.ReceiverId))
            {
                return -3;
            }

            var friendRequest = _mapper.Map<FriendRequest>(friendRequestDto);
            friendRequest.FriendRequestStatus = FriendRequestStatus.Awaiting;
            await _unitOfWork.FriendRequests.AddAsync(friendRequest);
            await _unitOfWork.SaveChangesAsync();
            return 1;
        }
```

```
        public async Task<int> Accept(FriendRequestDto friendRequestDto)
        {
            if (!await
_unitOfWork.FriendRequests.ContainsEntityWithId(friendRequestDto.SenderId
, friendRequestDto.ReceiverId))
            {
                return -1;
            }
            if (!await
_unitOfWork.Users.ContainsEntityWithId(friendRequestDto.SenderId))
            {
                return -2;
            }
            if (!await
_unitOfWork.Users.ContainsEntityWithId(friendRequestDto.ReceiverId))
            {
                return -3;
            }

            var friendRequest =
                await
_unitOfWork.FriendRequests.GetAsync(friendRequestDto.SenderId,
friendRequestDto.ReceiverId);

            friendRequest.FriendRequestStatus = FriendRequestStatus.Accepted;
            await AddFriends(friendRequestDto.SenderId,
friendRequestDto.ReceiverId);
            _unitOfWork.FriendRequests.Update(friendRequest);
```

```
            await _unitOfWork.SaveChangesAsync();
            return 1;
        }

        public async Task<int> Decline(FriendRequestDto friendRequestDto)
        {
            if (!await
_unitOfWork.FriendRequests.ContainsEntityWithId(friendRequestDto.SenderId
, friendRequestDto.ReceiverId))
            {
                return -1;
            }
            If (!await
_unitOfWork.Users.ContainsEntityWithId(friendRequestDto.SenderId))
            {
                return -2;
            }
            if (!await
_unitOfWork.Users.ContainsEntityWithId(friendRequestDto.ReceiverId))
            {
                return -3;
            }
            var friendRequest =
                await
_unitOfWork.FriendRequests.GetAsync(friendRequestDto.SenderId,
friendRequestDto.ReceiverId);
            friendRequest.FriendRequestStatus = FriendRequestStatus.Declined;
            _unitOfWork.FriendRequests.Update(friendRequest);
```

```
            await _unitOfWork.SaveChangesAsync();
            return 1;
        }
        public async Task<int> AddFriends(int userId, int friendId)
        {
            if (!await _unitOfWork.Users.ContainsEntityWithId(userId))
            {
                return -1;
            }
            if (!await _unitOfWork.Users.ContainsEntityWithId(friendId))
            {
                return -2;
            }
            if (await _unitOfWork.UserFriends.CheckIfFriends(userId, friendId))
            {
                return -3;
            }
            if (userId == friendId)
            {
                return -4;
            }
            await _unitOfWork.UserFriends.AddAsync(new UserFriend { UserId =
userId, FriendId = friendId });
            await _unitOfWork.UserFriends.AddAsync(new UserFriend {UserId =
friendId, FriendId = userId});
            await _unitOfWork.SaveChangesAsync();
            return 1;
        }
```

```
            }
        }
namespace SocialNetwork.BLL.Messages
{
    public class MessageService : IMessageService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public MessageService(IUnitOfWork unitOfWork, IMapper mapper)
        {
            _unitOfWork = unitOfWork;
            _mapper = mapper;
        }

        public async Task<IEnumerable<MessageDto>> GetAllAsync()
        {
            var messages = await _unitOfWork.Messages.GetAll().ToListAsync();
            return _mapper.Map<IEnumerable<MessageDto>>(messages);
        }

        public async Task<MessageDto> GetAsync(int id)
        {
            var message = await _unitOfWork.Messages.GetAsync(id);
            return _mapper.Map<MessageDto>(message);
        }
```

```
    public async Task<IEnumerable<MessageDto>> GetByConversationAsync(int
conversationId)
    {
        var messages = await
_unitOfWork.Messages.GetByConversation(conversationId).ToListAsync();
        return _mapper.Map<IEnumerable<MessageDto>>(messages);
    }

    public async Task<int> AddAsync(MessageDtoForCreate messageDtoForCreate)
    {
        if (!await
_unitOfWork.Users.ContainsEntityWithId(messageDtoForCreate.UserId))
        {
            return -1;
        }
        if (!await
_unitOfWork.Conversations.ContainsEntityWithId(messageDtoForCreate.Conversatio
nId))
        {
            return -2;
        }
        if (string.IsNullOrEmpty(messageDtoForCreate.Text))
        {
            return -3;
        }
        var message = _mapper.Map<Message>(messageDtoForCreate);
        await _unitOfWork.Messages.AddAsync(message);
        await _unitOfWork.SaveChangesAsync();
```

```
        return message.Id;
    }

    public async Task<int> UpdateAsync(MessageDtoForUpdate
messageDtoForUpdate)
    {
        if (string.IsNullOrEmpty(messageDtoForUpdate.Text))
        {
            return -1;
        }
        if (!await
_unitOfWork.Messages.ContainsEntityWithId(messageDtoForUpdate.Id))
        {
            return -2;
        }
        var message = await
_unitOfWork.Messages.GetAsync(messageDtoForUpdate.Id);
        _mapper.Map(messageDtoForUpdate, message);
        _unitOfWork.Messages.Update(message);
        await _unitOfWork.SaveChangesAsync();
        return message.Id;

    }

    public async Task<int> DeleteAsync(int id)
    {
        if (!await _unitOfWork.Messages.ContainsEntityWithId(id))
        {
```

```
            return -1;
        }
        _unitOfWork.Messages.Delete(id);
        await _unitOfWork.SaveChangesAsync();
        return 1;
      }
   }
}
namespace SocialNetwork.BLL.Users
{
   public class UserService : IUserService
   {
      private readonly IUnitOfWork _unitOfWork;
      private readonly IMapper _mapper;

      public UserService(IUnitOfWork unitOfWork, IMapper mapper)
      {
         _unitOfWork = unitOfWork;
         _mapper = mapper;
      }

      public async Task<IEnumerable<UserDto>> GetAllAsync()
      {
         var users = await _unitOfWork.Users.GetAll().ToListAsync();
         return _mapper.Map<IEnumerable<UserDto>>(users);
      }

      public async Task<UserDto> GetAsync(int id)
```

```
{
    var user = await _unitOfWork.Users.GetAsync(id);
    return _mapper.Map<UserDto>(user);
}

public async Task<IEnumerable<UserDto>> GetFriendsAsync(int userId)
{
    var friends = await _unitOfWork.UserFriends.GetByUser(userId).ToListAsync();
    return _mapper.Map<IEnumerable<UserDto>>(friends);
}

public async Task<int> AddAsync(UserDtoForCreate userDtoForCreate)
{
    if (string.IsNullOrEmpty(userDtoForCreate.FirstName))
    {
        return -1;
    }
    if (string.IsNullOrEmpty(userDtoForCreate.LastName))
    {
        return -2;
    }
    var user = _mapper.Map<User>(userDtoForCreate);
    await _unitOfWork.Users.AddAsync(user);
    await _unitOfWork.SaveChangesAsync();
    return user.Id;
}

public async Task<int> UpdateAsync(UserDto userDto)
```

```
{
    if (string.IsNullOrEmpty(userDto.FirstName))
    {
        return -1;
    }
    if (string.IsNullOrEmpty(userDto.LastName))
    {
        return -2;
    }
    if (!await _unitOfWork.Users.ContainsEntityWithId(userDto.Id))
    {
        return -3;
    }
    var user = await _unitOfWork.Users.GetAsync(userDto.Id);
    _mapper.Map(userDto, user);
    _unitOfWork.Users.Update(user);
    await _unitOfWork.SaveChangesAsync();
    return user.Id;
}

public async Task<int> DeleteAsync(int id)
{
    if (!await _unitOfWork.Users.ContainsEntityWithId(id))
    {
        return -1;
    }
    _unitOfWork.Users.Delete(id);
    await _unitOfWork.SaveChangesAsync();
```

```
            return 1;
        }

        public async Task<int> DeleteFriends(int userId, int friendId)
        {
            if (!await _unitOfWork.Users.ContainsEntityWithId(userId))
            {
                return -1;
            }
            if (!await _unitOfWork.Users.ContainsEntityWithId(friendId))
            {
                return -2;
            }
            if (!await _unitOfWork.UserFriends.CheckIfFriends(userId, friendId))
            {
                return -3;
            }
            if (userId == friendId)
            {
                return -4;
            }
            _unitOfWork.UserFriends.Delete(userId, friendId);
            _unitOfWork.UserFriends.Delete(friendId, userId);
            await _unitOfWork.SaveChangesAsync();
            return 1;
        }
    }
```