**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

«До захисту допущено»

Завідувач кафедри

|  |  |  |
| --- | --- | --- |
|  |  | Плескач В.Л. |
| (підпис) |  | (ініціали, прізвище) |

«12» квітня 2019 р.

**Дипломна робота**

на здобуття ступеня бакалавра

з напряму: 6.040302 «Інформатика»

на тему: «**Розроблення веб застосунку Месенджер**»

Пояснювальна записка

Виконав: студент 4 курсу, групи І-42

Гозак Ярослав Дмитрович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник: доцент, к.т.н. Бойко Юлія Петрівна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність інформатика за напрямом 6.040302 «Інформатика»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Плескач.В.Л

|  |  |  |
| --- | --- | --- |
| (підпис) |  | (ініціали, прізвище) |

«20» травня 2019р.

**ЗАВДАННЯ**

**на дипломний проект студенту**

**Гозаку Ярославу Дмитровичу**

(прізвище, ім’я, по батькові)

1. Тема проекту: «Розроблення веб застосунку Месенджер»

Керівник проекту: Бойко Юлія Петрівна, к.т.н., доцент

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені протоколом кафедри №3 від «12» жовтня 2018 р.

2. Термін подання студентом проекту: 23 травня 2019 року

3. Вихідні дані до проекту: монографії, підручники, навчальні посібники, статті та тези конфренцій вітчизняних і зарубіжних авторів, Інтернет-ресурси з питань мобільної розробки

4. Зміст пояснювальної записки:

1) Аналітична частина:

місце та роль веб застосунків у житті суспільства, зокрема дослідити процес розроблення веб застосунків на прикладі застосунку для ефективного обміну інформацією у режимі реального часу, обґрунтувати доцільність обраної теми дослідження, виявити невирішені проблеми, порівняти досягнення вітчизняних і зарубіжних фахівців із вказаної проблеми дослідження.

2) Практична частина:

визначити особливості архітектурного рішення веб застосунку для ефективного обміну інформацією у режимі реального часу, надати рішення та види забезпеченя вказаного застосунку, розробити веб застосунок, пропозиції та конкретні рекомендації щодо впровадження об’єкту дослідження та його супроводження.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслень, плакатів, презентацій тощо)

1. Аналітична частина: 1 лист – Use-case діаграма
2. Практична частина: 12 листів додатку програмного коду; 7 листів зображень веб застосунку та його опису інструкції користувача

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| Завдання видав | Завдання прийняв |
| 1 і 2 розділи аналітична частина | Бойко Ю.П., доцент |  |  |
| 3 розділ практична частина | Бойко Ю.П., доцент |  |  |

6. Дата видачі завдання 25 жовтня 2018 р.

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання дипломного проекту | Термін виконання етапів проекту | Примітка |
| 1 | *Вибір теми та наукового керівника кваліфікаційної роботи* |  |  |
| 2 | *Дата видачі завдання дипломної роботи* |  |  |
| 3 | *Настановча групова співбесіда з дипломної роботи* |  |  |
| 4 | *Затвердження плану дипломної роботи* |  |  |
| 5 | *Вивчення рекомендованої літератури* |  |  |
| 6 | *Аналіз методологій розроблення прикладного програмного забезпечення Інтернет-магазину* |  |  |
| 7 | *Постановка та формалізація задачі* |  |  |
| 9 | *Алгоритмізація задачі* |  |  |
| 10 | *Обґрунтування використовуваних програмно-технічних засобів* |  |  |
| 11 | *Розроблення прикладного веб додатку чату* |  |  |
| 12 | *Попередній захист* |  |  |
| 13 | *Налагодження програми* |  |  |
| 14 | *Виконання графічної частини* |  |  |
| 15 | *Оформлення пояснювальної записки дипломного проекту* |  |  |
| 16 | *Подання дипломного проекту на попередній захист* |  |  |
| 17 | *Подання дипломної роботи на основний захист* |  |  |
| 18 | *Подання дипломного проекту рецензенту* |  |  |
| 19 | *Захист дипломної роботи* |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  |  |  |  |
|  |  | (підпис) |  | (ініціали, прізвище) |
| Керівник проекту |  |  |  |  |
|  |  | (підпис) |  | (ініціали, прізвище) |

## Анотація (реферат)

Метою дипломної роботи є підвищення ефективності процесу розроблення веб застосунків із використанням ASP.NET Core на основі використання веб застосунку, реалізованого засобами ASP.NET WEB API.

Для досягнення мети вирішено такі завдання:

* дослідити процес розробки веб застосунків;
* проаналізувати особливості веб розробки на основі ASP.NET;
* проаналізувати архітектурні рішення веб розробки;
* реалізувати застосунок на основі ASP.NET.

### Об’єкт дослідження.

Об’єктом дослідження є веб застосунок на основі ASP.NET та Angular.

### ****Предмет дослідження****

**Предметом дослідження є нововведення та новітні технології, що були продемонстровані у зазначених технологіях/фреймворках.**

### ****Методи дослідження****

**Методом дослідження є системний аналіз і синтез знань щодо особливостей розробки веб застосунків на основі ASP.NET та Angular.**

**Наукова новизна результатів дослідження полягає в комплексному проведенні аналізу процесу розробки веб застосунків та можливості просувати розробку на основі ASP.NET для задоволення потреб користувачів.**

### ****Практичне значення та значущість роботи****

**Розроблений програмний продукт дозволяє ефективно обмінюватися інформацією у режимі реального часу та популяризує використання ASP.NET SignalR для розробки веб застосунків.**

***Ключові слова:* веб застосунок, ASP.NET, SignalR, C#, WEB API, фреймворк.**

**ЗМІСТ**

[Анотація (реферат) 7](#_Toc9176001)

[Об’єкт дослідження. 7](#_Toc9176002)

[Предмет дослідження 7](#_Toc9176003)

[Методи дослідження 7](#_Toc9176004)

[Практичне значення та значущість роботи 7](#_Toc9176005)

[ВСТУП 14](#_Toc9176006)

[РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ВЕБ - ЗАСТОСУНКІВ 16](#_Toc9176007)

[1.1 Місце та роль веб – застосунків в обміні інформацією 16](#_Toc9176008)

[1.3 Методи розробки веб – застосунків 20](#_Toc9176009)

[1.3.1 Класифікація методологій 21](#_Toc9176010)

[1.3.2 Типи методологій 22](#_Toc9176011)

[1.4 Висновки 26](#_Toc9176012)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ЗАСТОСУНКУ ДЛЯ ШВИДКОГО ОБМІНУ ІНФОРМАЦІЄЮ 27](#_Toc9176013)

[2.1 Аналіз ринку ПЗ для обміну повідомленнями 28](#_Toc9176014)

[2.2 Структура застосунку 30](#_Toc9176015)

[2.3 Архітектура клієнтської частини додатку 32](#_Toc9176016)

[2.4 Архітектура користувацької частини додатку — технології розрбки 33](#_Toc9176017)

[2.4.1 Фреймворк Angular 2+ 33](#_Toc9176018)

[2.4.2 Фреймворк React 35](#_Toc9176019)

[2.4.3 Фреймворк VueJS 36](#_Toc9176020)

[2.5 Архітектура серверної частини додатку — технології розробки 37](#_Toc9176021)

[2.5.1 Технологія ASP.NET 38](#_Toc9176022)

[2.5.2 Технологія ASP.NET Core 39](#_Toc9176023)

[2.5.3 SignalR 41](#_Toc9176024)

[2.5.4 База даних 43](#_Toc9176025)

[2.6 Безпека даних 44](#_Toc9176026)

[2.6.1 Шифрування бази даних 45](#_Toc9176027)

[2.6.2 Хешування з SHA-256 46](#_Toc9176028)

[2.6.3 Шифрування з AES 47](#_Toc9176029)

[2.7 Висновки 49](#_Toc9176030)

[РОЗДІЛ 3 РЕАЛІЗАЦІЯ І ВПРОВАДЖЕННЯ ВЕБ ЗАСТОСУНКУ 52](#_Toc9176031)

[НА ОСНОВІ ASP.NET WEB API 52](#_Toc9176032)

[3.1 Загальний опис застосунку 52](#_Toc9176033)

[3.2 Проектування бази даних за стосунку 54](#_Toc9176034)

[3.3 Архітектурне рішення застосунку 57](#_Toc9176035)

[3.3.1 Use-case діаграма 59](#_Toc9176036)

[3.4 Інструкція користувача 62](#_Toc9176037)

[3.5 Висновки 69](#_Toc9176038)

[ВИСНОВКИ 70](#_Toc9176039)

[ПЕРЕЛІК ПОСИЛАНЬ 72](#_Toc9176040)

**ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ**

Баг – помилка у програмі чи системі.

Веб застосунок - клієнт-серверний застосунок, в якому клієнт взаємодіє з сервером за допомогою браузера, а за сервер відповідає веб-сервер

В’ю – елемент користувацького інтерфейсу, сторінка, яку бачить користувач при роботі зі застосунком.

Гібридний застосунок (прогресивний веб-застосунок) – веб-застосунок, який є гібридом звичайної веб-сторінки та мобільного застосунку.

Десктоп – настільний комп’ютер.

Інтерфейс – набір інструментів для взаємодії користувача та комп’ютерної системи; візуальна частина, з якою взаємодіє користувач.

Компіляція – це процес, за якого комп’ютер перетворює мову високого рівня у машинний код, зрозумілий комп’ютеру. Програма, яка виконує це перетворення, називається компілятором.

Кросплатформна розробка – це розробка застосунків, які можуть запускатися на різних операційних системах.

Мобільний застосунок – програмне забезпечення, призначене для роботи на смартфонах, планшетах і інших мобільних пристроях.

ОС – операційна система.

Платформозалежна розробка – це розробка мобільних застосунків окремо під кожну платформу.

Програмний застосунок — користувацька [комп'ютерна програма](https://uk.wikipedia.org/wiki/Комп'ютерна_програма), що дає змогу вирішувати конкретні прикладні задачі користувача

Пуш-повідомлення – повідомлення, яке раптово випливає поверх усіх вікон.

Сплеш-екран – у мобільній розробці екран, який першим показується під час відкриття застосунку.

Фреймворк – готовий до використання комплекс програмних рішень, що охоплює дизайн, логіку та базову функціональність системи чи підсистеми.

Хакатон – змагання між командами розробників програмного забезпечення, які згуртовано працюють над розв’язанням якоїсь проблеми чи створення нового застосунку чи сервісу.

AngularJS – JavaScript-фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових застосунків.

Apache Cordova – фреймворк для створення мобільних застосунків, що продовжує розвиток платформи PhoneGap після передачі проекту компанією Adobe до фонду Apache.

API (прикладний програмний інтерфейс) – набір готових класів, процедур, функцій, структур і констант, що надаються застосунком (бібліотекою, сервісом) чи операційною системою для використання у зовнішніх програмних продуктах.

AR (Augmented Reality) – доповнена реальність.

ARM – це архітектура 32-бітових RISC-процесорів, розроблених компанією ARM Limited для мобільних пристроїв.

CLI (Command Line Interface) – інтерфейс командного рядка, консоль.

Cocoa – об’єктно-орієнтований прикладний програмний інтерфейс для ОС macOS.

Cocoa Touch – фреймворк для створення застосунків для пристроїв під iOS.

DI (Dependency injection) – шаблон проектування програмного забезпечення (ПЗ), що передбачає надання зовнішньої залежності програмному компоненту. Працює у парі з IoC.

ECMAScript – стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262.

FreeBSD – сучасна ОС на базі UNIX для серверів, десктопів і вбудованих комп’ютерних платформ.

GNU – вільна Unix-подібна операційна система.

GNUstep – вільна реалізація Cocoa.

HTML (HyperText Markup Language) – стандартна мова розмітки веб-сторінок у мережі Інтернет.

HTTP – протокол передачі даних, що використовується у інфокомунікаційних мережах.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

IoC Container (інверсія управління) – абстрактний принцип, набір рекомендацій для написання слабозв’язаного коду. Його суть полягає в тому, що кожний компонент системи повинен бути якомога більше ізольованим від інших, не покладаючись у своїй роботі на деталі конкретної реалізації інших компонентів.

iOS – мобільна операційна система для смартфонів, електронних планшетів, програвачів і інших пристроїв, що розроблювана компанією Apple.

JavaScript (JS) – мова програмування високого рівня.

JSX – це об’єктно-орієнтована мова програмування, призначена для сучасних веб-браузерах.

Linux – це сімейство Unix-подібних операційних систем на базі ядра Linux, яка охоплює набір утиліт і програм проекту GNU та інші компоненти.

LLVM (Low Level Virtual Machine) – універсальна система аналізу, трансформації і оптимізації програм, що реалізує віртуальну машину з RISC-подібними інструкціями.

NeXTSTEP – об’єктно-орієнтована, багатозадачна ОС, розроблена компанією NeXT Computer для власних комп’ютерів.

PWA (Progressive Web App) – прогресивний веб застосунок, веб застосунок, що може виконуватися як нативний мобільний застосунок.

Redux – JavaScript бібліотека з відкритим вихідним кодом для керування станом застосунку.

RISC (reduced instruction set computer) – архітектура процесора, в якій швидкість роботи збільшується за рахунок спрощення інструкцій, щоб їх декодування було простішим, а час виконання меншим.

UNIX – операційна система, яку розробляли співробітники підрозділу Bell Labs корпорації AT&T.

Xamarin – фреймворк для розробки мобільних застосунків (iOS/macOs/watchOS, Android, Windows Phone) з використанням мови C#.

Xcode – IDE, розроблений компанією Apple.

XNU – ядро операційної системи, розроблений компанією Apple. Є акронімом для X is Not Unix (X не є UNIX).

# ВСТУП

**Актуальність дослідження.**

**У сьогоднішньому світі рівень соціального контакту має дуже велике значення. Сьогодні дуже важливо щоб людина могла бути осткпною у будь-який час та у будб-якому місці. Велику роль в такій соціалізації людини отримав Інтернет. Поява інтернету зробила можливим передачу інформації у будь-якому вигляді із точки А в точку Б, що знаходяться дуже далеко одна від іншої. Проте Інтернет це лише набір системного обладнання і протоколів, за якими відбувається передача даних. Щоб звичайні люди могли обмінюватися інформацією потрібен зручний інтерфейс, який дозволяє відправляти та приймати дані у легкий спосіб. Так з’явилася електронна пошта.**

**Проте в наші дні пошта вже не є тим зручним способом передачі інформації будь-кому і будь-куди. Пошта має деякі обмеження, які потрібно вирішити. Одже наступною ітерацією стають месенджери і чати. Месенджери мають такі переваги як:**

**більш зручний інтерфейс користувача**

**можливість вкладання відео без посилання і, зазвичай, месенджери мають вбудовані програвачі для відео (та аудіо)**

**Адаптивна нотифікація користувача**

**Мета дослідження.**

Метою дипломної роботи є дослідження сучасних технологій веб програмування та **розробка веб застосунку Месенджер засобами бекенд фреймворку C# ASP.NET.**

**Завдання дослідження:**

* дослідити існуючі веб-застосунки;
* проаналізувати архітектурні рішення;
* розробити застосунок на основі ASP.NET.

**Об’єкт дослідження.**

Об’єктом дослідження є веб застосунок на основі ASP.NET.

**Предмет дослідження.**

**Предметом дослідження є нововведення та новітні технології, що були продемонстровані у зазначених технологіях/фреймворках.**

**Методи дослідження.**

**Методом дослідження є системний аналіз і синтез знань щодо особливостей розробки веб застосунків на основі ASP.NET.**

**Практичне значення одержаних результатів.**

**Виконана робота є програмною реалізацією веб застосунку на основі ASP.NET та Angular..**

**Структура роботи.**

**Робота складається зі вступу, трьох розділів, поділених на підрозділи, висновків, додатків і переліку посилань.**

# РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ВЕБ - ЗАСТОСУНКІВ

Розробка веб застосунків – це процес, при якому додатки розробляються для пристроїв, що мають доступ до інтернету, таких, як ПК (персональні комп’ютери), смартфони, планшети. Ці додатки можуть бути доступні лише через мережу інтернет, що вимагає стійкого або більш-менш стабільного з’єднання у клієнта. Такі додатки мають свою перевагу коли сконфігуровані у форматі тонкого клієнта. Тонкий клієнт означає, що більшість ресурсозатратних операцій виконується на сервері і лише відповідь надається клієнту, тож клієнту в свою чергу потрібно лише обробити відповідь, що часто зводиться до простого ренедерингу сторінки. Кожна з платформ для веб застосунків має інтегроване середовище розробки, яке надає інструменти, що дозволяють розробнику програмувати, тестувати і впроваджувати програми на цільову платформу.

## Місце та роль веб – застосунків в обміні інформацією

Отже, веб-застосунок —це розподілений [застосунок](https://uk.wikipedia.org/wiki/Застосунок), в якому клієнтом виступає [браузер](https://uk.wikipedia.org/wiki/Браузер), а сервером — [веб-сервер](https://uk.wikipedia.org/wiki/Веб-сервер). Він є різновидом програмних застосунків із такими властивостями: браузер виступає тонким клієнтом у цій клієнт-серверній архітектурі, а зберігання усіх даних і координація процесів відбувається на стороні сервера. Дані зберігаються на окремому сервері – сервері бази даних. Хоча база даних може знаходитися на одному сервері із самим програмним застосунком.

Істотною перевагою побудови веб-застосунків для підтримки стандартних функцій браузера є те, що функції повинні виконуватися незалежно від операційної системи клієнта. Замість того, щоб писати різні версії для [Microsoft Windows](https://uk.wikipedia.org/wiki/Microsoft_Windows), Mac OS, GNU/Linux й інших операційних систем, застосунок створюється один раз для довільно обраної платформи і на ній розгортається. Проте різна реалізація [HTML](https://uk.wikipedia.org/wiki/HTML), [CSS](https://uk.wikipedia.org/wiki/CSS), DOM й інших специфікацій в браузерах може викликати проблеми при розробці веб-застосунків і подальшої підтримки. Крім того, можливість користувача налаштовувати багато параметрів браузера (наприклад, розмір шрифту, кольори, відключення підтримки сценаріїв) може перешкоджати коректній роботі застосунку.

Веб-застосунок являє собою веб-сайт, на якому розміщені сторінки з частково або повністю несформованим вмістом. Остаточне вміст формується тільки після того, як відвідувач сайту запросить сторінку з веб-сервера. У зв'язку з тим що остаточне вміст сторінки залежить від запиту, створеного на основі дій користувача, така сторінка називається динамічної. Спектр використання веб-додатків дуже широкий.

Веб-додатки дозволяють відвідувачам швидко і легко знаходити необхідну інформацію на веб-сайтах з великим об'ємом інформації. Даний вид веб-додатків дозволяє здійснювати пошук у вмісті, упорядковувати вміст і переміщатися по ньому зручним для відвідувачів способом. Прикладами таких додатків можуть служити внутрішні мережі компаній - Microsoft MSDN (www.msdn.microsoft.com) і Amazon (www.amazon.com).

Веб-додатки дозволяють збирати, зберігати і аналізувати дані, отримані від відвідувачів сайту. Довгий час використовувався метод, при якому дані, введені в HTML-форми, відсилалися для обробки CGI-додатків або спеціально призначеним працівникам у вигляді повідомлень електронної пошти. Веб-застосунок дозволяє зберігати дані безпосередньо в базі даних, а також отримувати дані і формувати звіти на основі отриманих даних для аналізу. Як приклад можна привести інтерактивні сторінки банків, сторінки для контролю товарних запасів, соціологічні дослідження та опитування, а також форми для зворотного зв'язку з користувачами.

Веб-застосунок може використовуватися для оновлення веб-сайтів з періодично мінливим вмістом. Веб-застосунок звільняє веб-дизайнера від рутинної роботи постійного оновлення HTML-сторінок сайту. Постачальники вмісту, наприклад редактори новин, відповідають за наявність свіжого матеріалу, а веб-застосунок стежить за автоматичним оновленням сайту. Як приклад можна привести веб-версію служби новин CNN (www.cnn.com).

Отже, веб-застосунком є програмний застосунок, що потребує доступ до мережі і виконується у браузері. Недоліками веб-застосунку перед десктопними застосунками є:

* Залежність від мережі. Виключення становлять деякі веб-додатки, які можуть функціонувати деякий час автономно.
* Повільна у порівнянні до десктопних додатків анімація і відклик графіки, часто пов’язані з передачею даних по мережі.

Перевагами, безкмовно, є:

* Платформонезалежність – оскільки додаток виконується у браузері, то розробникам не варто перейматися платформою/ОС користувача. За це відповідальний браузер
* Не потребує встановлення на комп’ютер
* Різноманітний інтерфейс, можливість легкого встановлення додаткових шрифтів, елементів керування

Отже, веб-застосунки набирають все більшої популярності, оскільки з кожним роком рівень застосунків підіймаєтьяс, а отже і місце, що вони займають, користувачам потрібно більше місця для завантаження нових застосунків. Отже звільнення користувача від потреби завантажувати весь додаток робить веб-застосунки дуже корисними у наш час.

* 1. **Класифікація веб – застосунків**

Основною класифікаційною характеристикою веб-застосунків є архітектура. Однак, архітектура веб-застосунків теж піддана змінам, тож з часом різні архітектури почали об'єднуватися та почали виділятись нові типи архітектури. Так довгий час стандартним шаблоном архітектури веб-застосунку був MVC шаблон. Ключовою особливістю цього шаблону є розділення програмии на 3 основних компоненти: Model – модель, модель стану застосунку, власне – це дані застосунку із визначеними для них структурами представлення; View – представлення, власне – це графічний інтерфейс, який бачить користувач; Controller – контроллер, вся бізнес логіка програми.

Проте дуже швидко стало зрозумілим, що цей шаблон не підходить для великих додатків. Адже із ростом застосунку зростає кількість залежностей між частинами цього застосунку, що веде до зростання складності підтримувати застосунок, додавати нові компоненти і, часом, тестувати його.

Тож, з появою досить великих застосунків народився новий шаблон архітектури як мікросервісна архітектура. Мікросервісна архітектура описує додаток як сукупність незалежних функціонально завершених компонентів – мікросервісів. Ця архітектура вносить легку розширюваність застосунку та відносно легку підримку існуючтого функціоналу. Проте тепер, оскільки логічні компоненти програми відділені одна від іншої, постає питання їх спілкування та обміну інформацією між собою. Найпростішим вірішенням цієї проблеми стало створення API для кожного мікросервісу і комунікація між сервісами за допомогою протоколу HTTP. Більш елегантним та функціональним рішенням є використання Message Queues, далі черги повідомлень. Черга повідомлень це незалежний компонент застосунку, через який спілкуються між собою мікросервіси. Черги повідомлень дозволяють легко надсилати повідомлення одразу кільком учасникам, отримувати декілька відповідей на один запит та деякі інші речі, які не може надати HTTP.

Є ще один вид архітектури застосунку, проте він більше стосується саме користувацького інтерфейсу. Це SPA застосунки. SPA (Single Page Application) – це застосунок, що складається з однієї, проте дуже динамічної сторінки. Ця архітектура стала можливою із появленням і отриманням популярності іншої технології – AJAX (Asynchronous Javascript And Xml). Ця технологія дозволяє сторінці робити запити на сервер у фоновому режимі без перезагрузки сторінки. Таким чином застосунок підгружає нові дані і додає їх на сторінку без повної перегрузки. Це дозволяє створити одну базову сторінку, контент якої рендериться у режимі реального часу і всі нові дані поступають у фоновому режимі. Ця архітектура дуже легко і натурально може бути поєднана з мікросервісною архітектурою.

## Методи розробки веб – застосунків

Основним аспектом серед методів розробки програмних застосунків, в тому числі веб-застосунків є методологія олзробки програмного застосунку. Методології розробки почали з’являтись у часи написання перших програм.

Методологіярозробкипрограмногозабезпечення - сукупність методів, застосовуваних на різних стадіях життєвогоциклурозробкипрограмного [забезпечення](https://uk.wikipedia.org/wiki/Життєвий_цикл_програмного_забезпечення), що мають спільний філософський підхід та, відповідно до цього підходу, дозволяють забезпечити найкращу ефективність [процесів розробки](https://uk.wikipedia.org/wiki/Процес_розробки_програмного_забезпечення).

Кожна методологія характеризується:

* Філософським підходом або основними принципами. Ці принципи, від яких залежить ефективність всієї методології, зазвичай можна коротко сформулювати і легко пояснити;
* Узгодженою множиною моделей та методів, які реалізують дану методологію;
* Концепціями (поняттями), що дозволяють більш точно визначити методи.

Можна простежити три шляхи виникнення методологій. По-перше, вони можуть бути вираженням практичного досвіду. По-друге, методології можуть походити від однієї з чотирьох моделей алгоритму: абстрактна машина Тюрінга (імперативне програмування), рекурсивні функції Гільберта і Аккермана (структурне програмування), лямбда-числення Черча (функціональне програмування), нормальні алгоритми Маркова (логічне програмування). По-третє, методології можна пояснити через відображення однієї з трьох структур мови моделювання на структуру мови програмування. Складовими частинами можуть бути структура даних, структура управління і логіка. Кожне з дев'яти відображень визначає або методологію, або досить серйозний метод програмування. Наприклад, відображення логіка-логіка лежить в основі логічного програмування.

### Класифікація методологій

**Класифікація по ядрам**

При зверненні до методології, яка має ядро (англ. core), відповідне способу опису алгоритму, та додаткові особливості, можна виділити такі п'ять основних ядер методологій:

* Методологія імперативного програмування
* Методологія ООП
* Методологія функціонального програмування
* Методологія логічного програмування
* Методологія програмування в обмеженнях

Можна помітити, що ці методології знаходяться на шкалі від навігаційних (покрокове управління виконанням) до специфічних (визначення вимог до результату).

**Класифікація за топологічною специфікою**

Специфіка (топологічна специфіка) — спосіб вибору методів для уточнення ядра методології. Критерієм якості тієї чи іншої топології можуть бути загальні витрати на розробку ПЗ. У свою чергу, витрати на розробку залежать серед іншого від ключових мовних абстракцій: абстракції даних, управління і модульності. Наприклад, в імперативній методології можна дотримуватися методів структурного програмування, що дає більш вигідну топологію з погляду мовних засобів. Результатом є методологія структурного програмування.

**Класифікація за специфікою реалізації**

У відповідності до архітектури апаратного забезпечення, реалізація може бути централізованою або паралельною. Наприклад, методологія (імперативного) паралельного програмування, методологія логічного паралельного програмування.

Крім того, методологія може бути гібридної. Наприклад, найбільш часта суміш функціонального та логічного програмування.

Проводяться дослідження щодо уніфікації методологій програмування.

### Типи методологій

Водоспадна (каскадна, послідовна) модель

Водоспадна модель життєвого циклу (англ. Waterfall model) була запропонована в 1970 р Уинстоном Ройсом. Вона передбачає послідовне виконання всіх етапів проекту в строго фіксованому порядку. Перехід на наступний етап означає повне завершення робіт на попередньому етапі. Вимоги, визначені на стадії формування вимог, строго документуються у вигляді технічного завдання і фіксуються на весь час розробки проекту. Кожна стадія завершується випуском повного комплекту документації, достатньої для того, щоб розробка могла бути продовжена іншою командою розробників.

Етапи проекту відповідно до каскадної моделі:

1. Формування вимог
2. Проектування
3. Реалізація
4. Тестування
5. Впровадження
6. Експлуатація та супровід

Переваги:

* Повна і узгоджена документація на кожному етапі;
* Легко визначити терміни і витрати на проект.

Недоліки:

У Водоспадної моделі перехід від однієї фази проекту до іншого передбачає повну коректність результату (виходу) попередньої фази. Однак неточність будь-якої вимоги або некоректна його інтерпретація в результаті призводить до того, що доводиться «відкочуватися» до ранньої фази проекту і необхідна переробка не просто вибиває проектну команду з графіка, але призводить часто до якісного зростання витрат і, не виключено, до припинення проекту в тій формі, в якій він спочатку замислювався. На думку сучасних фахівців, основна помилка авторів водоспадної моделі полягає в припущеннях, що проект проходить через весь процес один раз, спроектована архітектура гарна і проста у використанні, проект здійснення розумний, а помилки в реалізації легко усуваються в міру тестування. Ця модель виходить з того, що всі помилки будуть зосереджені в реалізації, а тому їх усунення відбувається рівномірно під час тестування компонентів і системи. Таким чином, водоспадна модель для великих проектів мало реалістична і може бути ефективно використана тільки для створення невеликих систем.

**Ітераційна модель**

Альтернативою послідовній моделі є так звана модель ітеративної і інкрементальної розробки (англ. Iterative and incremental development, IID), що отримала також від Т. Гілбі в 70-і рр. назву еволюційної моделі. Також цю модель називають ітеративною моделлю і інкрементальною моделлю.

Модель IID передбачає розбиття життєвого циклу проекту на послідовність ітерацій, кожна з яких нагадує «міні-проект», включаючи всі процеси розробки в застосуванні до створення менших фрагментів функціональності, в порівнянні з проектом в цілому. Мета кожної ітерації - отримання працюючої версії програмної системи, що включає функціональність, отриману інтегрованим змістом усіх попередніх і поточної ітерації. Результат фінальної ітерації містить всю необхідну функціональність продукту. Таким чином, із завершенням кожної ітерації продукт бере зріст - інкремент - до його можливостей, які, отже, розвиваються еволюційно. Ітеративність, Інкрементальність і Еволюційність в даному випадку є вираз одного і того ж змісту різними словами зі злегка різних точок зору.

За висловом Т. Гілбі, «Еволюція - прийом, призначений для створення видимості стабільності. Шанси успішного створення складної системи будуть максимальними, якщо вона реалізується в серії невеликих кроків і якщо кожен крок містить в собі чітко визначений успіх, а також можливість «відкату» до попереднього успішному етапу в разі невдачі. Перед тим, як пустити в справу всі ресурси, призначені для створення системи, розробник має можливість отримувати з реального світу сигнали зворотного зв'язку і виправляти можливі помилки в проекті ».

Підхід IID має і свої негативні сторони, які, по суті, - зворотна сторона достоїнств. По-перше, цілісне розуміння можливостей і обмежень проекту дуже довгий час відсутні. По-друге, при ітераціях доводиться відкидати частину зробленої раніше роботи. По-третє, сумлінність фахівців при виконанні робіт все ж знижується, що психологічно зрозуміло, адже над ними постійно тяжіє відчуття, що «все одно все можна буде переробити і поліпшити пізніше».

Різні варіанти ітераційного підходу реалізовані в більшості сучасних методологій розробки (RUP, MSF, XP).

**Спіральна модель**

Спіральна модель (англ. Spiral model) була розроблена в середині 1980-х років Баррі Боем. Вона заснована на класичному циклі Демінга PDCA (plan-do-check-act). При використанні цієї моделі програмний застосунок створюється в кілька ітерацій (витків спіралі) методом прототипування.

Кожна ітерація відповідає створенню фрагмента або версії ПЗ, на ній уточнюються цілі і характеристики проекту, оцінюється якість отриманих результатів та плануються роботи наступної ітерації.

На кожній ітерації оцінюються:

1. Ризик перевищення термінів і вартості проекту;
2. Необхідність виконання ще однієї ітерації;
3. Ступінь повноти і точності розуміння вимог до системи;
4. Доцільність припинення проекту.

Важливо розуміти, що спіральна модель є не альтернативою еволюційної моделі (моделі IID), а спеціально проробленим варіантом. На жаль, нерідко спіральну модель або помилково використовують як синонім еволюційної моделі взагалі, або (не менше помилково) згадують як абсолютно самостійну модель поряд з IID.

Відмінною особливістю спіральної моделі є спеціальна увага, що приділяється ризикам, що впливає на організацію життєвого циклу, і контрольним точкам. Боем формулює 10 найбільш поширених (за пріоритетами) ризиків:

1. Дефіцит фахівців.
2. Нереалістичні терміни і бюджет.
3. Реалізація невідповідної функціональності.
4. Розробка неправильного користувальницького інтерфейсу.
5. Перфекціонізм, непотрібна оптимізація і відточування деталей.
6. Безперервний потік змін.
7. Брак інформації про зовнішні компоненти, що визначають оточення системи або залучених в інтеграцію.

Недоліки в роботах, виконуваних зовнішніми (по відношенню до проекту) ресурсами.

Недостатня продуктивність одержуваної системи.

Розрив в кваліфікації фахівців різних областей.

У сьогоднішній спіральної моделі визначено наступний загальний набір контрольних точок:

1. Concept of Operations (COO) - концепція (використання) системи;
2. Life Cycle Objectives (LCO) - цілі і зміст життєвого циклу;
3. Life Cycle Architecture (LCA) - архітектура життєвого циклу; тут же можна говорити про готовність концептуальної архітектури цільової програмної системи;
4. Initial Operational Capability (IOC) - перша версія створюваного продукту, придатна для дослідної експлуатації;
5. Final Operational Capability (FOC) - готовий продукт, розгорнутий (встановлений і налаштований) для реальної експлуатації.

## Висновки

Отже, мессенджери стали практично невід’ємною частиною життя людини у суспільстві. Мессенджерами користуються у різних сферах життя людини: особисте життя, робочий час. Відповідно, мессенджери можна розділити на корпоративні (нприклад Slack, Skype) та особисті (Telegram, WhatsApp).

Також було виявлено велику кількість методологій розробки програмних, в тому числі, веб застосунків. Найбільш популярними методологіями є каскадна (ватерфол) модель, канбан та скрам. Вибір конкретної методології залежить від багатьох чинників, в тому числі: архітектури застосунку, стилю управління компанією, кількості команд розробників.

# РОЗДІЛ 2 [ПРОЕКТУВАННЯ АРХІТЕКТУРИ З](#__RefHeading___Toc6231_3828987355)АСТОСУНКУ ДЛЯ [ШВИДКОГО ОБМІНУ ІНФОРМАЦІЄЮ](#__RefHeading___Toc6231_3828987355)

Архітектура програмного забезпечення (англ. Software architecture) – сукупність найважливіших рішень про організацію програмної системи. Архітектура включає:

1. вибір структурних елементів і їх інтерфейсів, за допомогою яких складена система, а також їх поведінки в рамках співпраці структурних елементів;
2. з'єднання обраних елементів структури і поведінки у більші системи;
3. архітектурний стиль, який направляє всі елементи, їх інтерфейси, їх співпрацю та їх з'єднання.

Документування архітектури програмного забезпечення спрощує процес комунікації між розробниками, дозволяє зафіксувати прийняті проектні рішення і надати інформацію про них експлуатаційного персоналу системи, повторно використовувати компоненти і шаблони проекту в інших.

Архітектура програмного забезпечення є метафорою, аналогічною архітектурі будівлі.

Архітектура ПЗ полягає в тому, щоб зробити фундаментальний структурний вибір, який буде дорого змінити після виконання. Вибір архітектури є надзвичайно важливим завданням, яке впливає на кінцевий результат. Працюючи над архітектурою необхідно враховувати всі завдання та можливі розширення проекту, щоб якомога якісніше виконати завдання.

Документування архітектури ПЗ полегшує спілкування між зацікавленими сторонами, фіксує ранні рішення щодо дизайну високо рівні і дозволяє повторно використовувати конструктивні компоненти між проектами.

## 2.1 Аналіз ринку ПЗ для обміну повідомленнями

На даний момент на ринку є велика кількість різного ПЗ для обміну повідомленнями. Розглянемо найбільш популярні з них:

**Telegram** — кросплатформовий месенджер, що дозволяє обмінюватися повідомленнями і медіафайлами багатьох форматів. Використовуються пропріетарна серверна частина c закритим кодом, що працює на потужностях декількох компаній США і Німеччини, що фінансуються Павлом Дуровим в обсязі близько 13 млн доларів США щорічно, і кілька клієнтів з відкритим вихідним кодом, в тому числі під ліцензією GNU GPL.

Для месенджера був створений протокол MTProto, що передбачає використання декількох протоколів шифрування. При авторизації і аутентифікації використовуються алгоритми RSA-2048, DH-2048 для шифрування, при передачі повідомлень протоколу в мережу, вони шифруються AES з ключем, відомим клієнту і серверу. Також застосовуються криптографічні хеш-алгоритми SHA-1 і MD5.

Особливості: Telegram забезпечує шифрування повідомлень, анонімність користувачів та додаткові сервіси із збереження персоанальних і платіжних даних у зашифрованому вигляді. Є можливість відео- та аудіодзвінків.

**Viber** – додаток для смартфонів, що дозволяє здійснювати відео і голосові VoIP дзвінки через мережу Інтернет, підключаючись до неї за допомогою технології Wi-Fi або передаючи інтернет-трафік через мобільні мережі сотового зв'язку. Здійснювати дзвінки між користувачами з встановленим Viber безкоштовно (оплачується тільки інтернет-трафік за тарифом оператора зв'язку). Також в Viber є можливість передачі текстових повідомлень, зображень, відео- та аудіоповідомлень, документів і файлів.

До останнього часу Viber не зберігав даних повідомлень на серверах, а лише перенаправляв повідомлення одержувачам. Це пов“язано із спробою Viber зробити обмін даними більш безпечним для користувачів за рахунок відсутності цих даних на серверах.

**Facebook Messenger** – додаток для обміну миттєвими повідомленнями і відео, створений Facebook. Застосунок інтегрований з системою обміну повідомленнями на основному сайті Facebook (Facebook Chat) і побудований на базі відкритого протоколу MQTT. За даними на квітень 2017 року місячна аудиторія месенджера становила 1 млрд. чоловік. Основною особливістю Messenger є його інтеграція із Facebook, у той час як всі інші месенджери являються індивідуальними застосунками і мають свій власний веб-сайт, що розроблений для них, а не навпаки.

**WhatsApp** – популярна безкоштовна система миттєвого обміну текстовими повідомленнями для мобільних і інших платформ з підтримкою голосового і відеозв'язку. Дозволяє пересилати текстові повідомлення, зображення, відео та аудіо через Інтернет.

Клієнт працює на платформах Android, iOS, Windows Phone, Nokia Symbian, Nokia S40, KaiOS, а також ОС Windows і у вигляді веб-додатку.

WhatsApp використовує модифікований протокол Extensible Messaging and Presence Protocol (XMPP, раніше відомий як Jabber). При установці створюється обліковий запис на сервері s.whatsapp.net, який використовує номер телефону в якості імені користувача (Jabber ID: [номер телефону] @ s.whatsapp.net). Версія під Android автоматично використовує в якості пароля MD5-хеш від зміненого ідентифікатора IMEI, а версія під iOS використовує MD5-хеш від MAC-адреси.

Алгоритм генерації пароля і відсутність шифрування в ранніх версіях WhatsApp неодноразово критикувалися.

У травні 2011 року було повідомлено про дірку в безпеці, яка залишала аккаунт користувача WhatsApp відкритим для перехоплення сесії і пакетного аналізу. Передача даних в WhatsApp не шифрується і дані відправляються і приймаються у відкритому тексті – це значить, що повідомлення можуть легко бути прочитані, якщо був простежено шлях проходження пакетів. У вересні 2011 року WhatsApp випустили нову версію для iPhone, яка закрила діру в безпеці, яка дозволяла відправляти підроблені повідомлення і читати повідомлення будь-якого користувача WhatsApp.

Проте проблеми з шифруванням повідомлень і даних користувачів ще довгий час були присутні.

Великі проблеми з конфіденційністю і безпекою стали предметом спільного канадсько-голландського урядового розслідування. Найбільше занепокоєння полягала в тому, що після установки додаток WhatsApp копіює всі номери мобільних телефонів з адресної книги користувачів на сервера WhatsApp, щоб повідомити користувачеві, хто з його контактів доступний через WhatsApp. При цьому на сервер відсилаються контактні дані будь-яких людей, навіть тих, хто не використовує WhatsApp. Перед відправкою телефонні номери перетворюються за допомогою хеш-функції без використання «солі» (зворотне перетворення даного хешу в телефонний номер займає, за оцінками фахівців, не більше 3 хвилин навіть на побутових комп'ютерах), імена з адресної книги не надсилаються

Отже, на даний момент у світі присутня велика кількість аналогів із різними недоліками та перевагами. Як видно, Телеграм став першим месенджером, що повністю, або майже повністю розв“язав питання конфіденційності, анонімності та безпеки передачі даних.

## 2.2 Структура застосунку

Стандартною для веб-застосунку є структура, що має чітко визначений модуль користувацького інтерфейсу, що написаний на мові JavaScript або JavaScript- чи TypeScript-фреймворку, має API для спілкування користувацького інтерфейсу із програмною логікою застосунку. Окрім API веб-застосунки для обміну повідомленнями зазвичай мають окремі сервіси для безпосереднього обміну повідомленнями користувачами, сервіс аутентифікації, сервіс профілю користувача, тощо.

Користувацька частина, що призначена для відображення у браузері має бути написана на мові JavaScript та її інтерфейс має бути розмічений HTML. Ці вимоги виставляються до кожного веб-застосунку і обумовлені потребою в загальному інтерфейсі для усіх браузерів і веб-застосунків. Таким інтерфейсом являється сукупність технологій JavaScript, HTML та CSS. Використання TypeScript дозволено з тої причини, що TypeScript може бути скомпільований у JavaScript.

Оскільки більшість дій у застосунку для обміну повідомленнями користувач може робити лише будучи авторизованим, то завжди гарним підходом є наявність API, який перед виконанням будь-якої команди перевіряє чи авторизовує користувача. У правильно сконфігурованому та зпроектованому застосунку лише API Gateway спілкується з сервісом авторизації з невеликою кількістю виключень. Проте сервіс аутентифікації також може бути використаний, якщо веб-застосунок надає послуги авторизації користувачів для інших застосунків.

Основною функцією застосунку для обміну повідомленнями є обмін повідомленнями. Оскільки функціонал може бути досить обштрний із включенням багатьох нововведень, додаткових функцій (як використання смайликів), то зазвичай обмін повідомленнями виконується через спеціалізований сервіс, який може розширюватися.

Інші функції веб-застосунку, як збереження інформації про користувача, надання користувачу доступу до його профілю, пошук інших користувачів та ін. Можуть виконуватися спеціально призначеними для цього сервісами застосунку.

Вся інформація застосунку зберігається у його базах даних, що можуть бути розміщені на інших серверах. Застосунок може мати одну або декілька баз даних. Уе залежить від архітектури застосунку та його розмірів. У мікросервісній архітектурі кожен окремий мікросервіс може мати свою власну базу даних. Таким чином досягається дійсна незалежність мікросервісів і більш слабка їх зв’язність. Недоліком цього є дублювання даних у базах даних різних мікросервісів.

## 2.3 Архітектура [клієнтської частини додатку](#__RefHeading___Toc7416_1282661385)

Клієнтська частина додатку, іншими словами — фронтенд — це та частина додатку, що безпосередньо завантажується на пристрій користувача і представляє такі функції:

Інтерфейс Користувача — UI (User Interface)

Зв’язок із серверною частиною додатку

UI — та єдина компонента додатку, з якою взаємодіє користувач. Дуже часто користувач навіть не уявляє, що за користувацьким інтерфейсом стоїть дуже великий масив логіки, який дозволяє користувачу натиснути кнопку “Надіслати” і його, наприклад, повідомлення відправиться іншому користувачеві.

Раніше всі елементи, що бачив користувач були розмічені за допомогою HTML та декоровані за допомогою CSS. За анімацію частково відповідав JavaScrip та частково CSS. Також всі маніпуляції з даними та оновлення DOM теж виконував JavaScript.

Проте з часом стали з’являтися нові фреймворки, ціллю яких було спрощення написання розмітки і коду. Основна проблема, я ку призвані вирішити фреймворки — це складні маніпуляції з DOM елементами а також надання користувачеві нового якісного User Experience.

## 2.4 [Архітектура користувацької частини додатку — технології розрбк](#__RefHeading___Toc5078_346575917)и

Інтерфейс користувача є однією з найбільш важливих частин проекту. Тому на це відводиться досить багато часу. Оскільки Інтерфейс Користувача досить тісно пов’язний з логікою інтерфейсу, то про ці реці завжди гоаорять сумісно. Також це є основною причиною розвитку фронтенд фреймфорків таких як Angular, ReactJS, VueJS та інших. Часто фрейморки абстрагують застосунок у вигляді компонент об’єднуючи розмітку, стилі і пов’язану з ними логіку у єдиній логічній компоненті.

Компонентно-орієнтоване програмування (англ. Component-oriented programming, COP) - парадигма програмування, істотно спирається на поняття компонента - незалежного модуля вихідного коду програми, призначеного для повторного використання і розгортання і реалізується у вигляді безлічі мовних конструкцій (наприклад, «класів» в об'єктно-орієнтованих мовах програмування), об'єднаних за спільною ознакою і організованих відповідно до визначених правилами і обмеженнями.

### 2.4.1 Фреймворк Angular 2+

Фреймфорк Angular опирається на такий підхід, що дозволяє скоротити написаний код та використовувати його повторно.

Angular (найчастіше називається "Angular 2+" або "Angular v2 і вище") - це основа для веб-додатків з відкритим вихідним кодом на основі TypeScript, очолювана Angular Team у Google і громадою та корпораціями. Angular - це повна переписка з тієї ж команди, яка побудувала AngularJS.

Спочатку переписання AngularJS було названо командою "Angular 2", але це призвело до плутанини серед розробників. Щоб пояснити, команда оголосила, що окремі терміни повинні використовуватися для кожного фреймворку з "AngularJS", що відноситься до версій 1.X і "Angular" без "JS", що відноситься до версій 2 і вище.

Angular був перезаписом AngularJS.

Angular не має поняття "Scope" або контролерів, замість цього використовує ієрархію компонентів як свою первинну архітектурну характеристику.

Angular має інший синтаксис виразу, фокусуючись на "[]" для прив'язки властивості, і "()" для прив'язки подій

Модульність - багато основних функціональних можливостей перейшли до модулів

Компанія Angular рекомендує використовувати мову TypeScript на Microsoft, що вводить наступні функції:

* Об'єктно-орієнтоване програмування на основі класів
* Статичність типів змінних
* Дженерики — узагальнені класи чи методи, що можуть приямати чи віддавати не конкретні типи даних, а сімейство типів
* TypeScript є надмножиною ECMAScript 6 (ES6) і сумісний з ECMAScript 5 (тобто JavaScript). Angular також включає ES6:
* Лямбда-вирази
* Ітератори
* Генератори в стилі Python
* Відображення
* Динамічне завантаження
* Асинхронна компіляція шаблонів
* Ітеративні зворотні виклики, надані RxJS.
  + RxJS обмежує видимість і налагодження стану, але їх можна вирішити за допомогою реактивних додатків, таких як ngReact або ngrx.
* Підтримка
  + Підтримка Angular Universal, технологія, яка запускає вашу програму Angular на сервері
* Має свій власний набір сучасних компонентів інтерфейсу, які працюють через Інтернет, мобільні та ПК, називаються Angular Material

### 2.4.2 Фреймворк React

React (іноді React.js або ReactJS) - JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів.

React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій.

React може використовуватися для розробки односторінкових і мобільних додатків. Його мета - надати високу швидкість, простоту і масштабованість. Як бібліотеки для розробки призначених для користувача інтерфейсів React часто використовується з іншими бібліотеками, такими як Redux.

React був створений Джорданом Валку, розробником програмного забезпечення з Facebook. На нього вплинув XHP - компонентний HTML фреймворк для PHP. У перший раз React був використаний в стрічці новин Facebook в 2011 році і пізніше в стрічці Instagram в 2012 році. Вихідний код React був відкритий в травні 2013 року на конференції «JSConf US».

React Native був анонсований на конференції Facebook «React.js Conf» в лютому 2015 року, а вихідний код був відкритий в березні 2015 року. Він дозволяє розробляти нативні Android, iOS і UWP додатки з використанням React.

18 квітня 2017 року Facebook анонсував React Fiber, переписану і оптимізовану версію React. React Fiber стане основою розробки всіх майбутніх функцій і поліпшень.

Особливості:

* Односпрямована передача даних - Властивості передаються від батьківських компонентів дочірнім. Компоненти отримують властивості як безліч незмінних (англ. Immutable) значень, тому компонент не може безпосередньо змінювати властивості, але може викликати зміни через callback функції. Такий механізм називають «властивості вниз, події наверх».
* Віртуальний DOM - React використовує віртуальний DOM (англ. Virtual DOM). React створює кеш структуру в пам'яті, що дозволяє обчислювати різницю між попереднім і поточним станами інтерфейсу для оптимального поновлення DOM браузера. Таким чином програміст може працювати зі сторінкою, вважаючи, що вона оновлюється вся, але бібліотека самостійно вирішує, які компоненти сторінки необхідно оновити.
* JSX - JavaScript XML (JSX) - це розширення синтаксису JavaScript, яке дозволяє використовувати схожий на HTML синтаксис для опису структури інтерфейсу. Як правило, компоненти написані з використанням JSX, але також є можливість використання звичайного JavaScript. JSX нагадує іншу мову, створену в компанії Фейсбук для розширення PHP, XHP.

### 2.4.3 Фреймворк VueJS

Vue.js (також Vue; / vjuː /) - JavaScript-фреймворк з відкритим вихідним кодом для створення користувацьких інтерфейсів. Легко інтегрується в проекти з використанням інших JavaScript-бібліотек. Може функціонувати як веб-фреймворк для розробки односторінкових додатків в реактивному стилі.

У 2013 році співробітник Google Еван Ю (Evan You), працюючи над одним з проектів, прийшов до висновку, що не існує готових рішень для швидкого прототипування складних користувацьких інтерфейсів веб-додатків: React тоді був на ранній стадії розробки, основними інструментами були такі складні фреймворки, як AngularJS або орієнтований на MVC-архітектуру Backbone.js, не відрізнялися простотою і орієнтовані на розробку великих програм. Для подолання цієї прогалини Еван Ю почав розробку Vue.js, яка, зберігаючи простоту, виявилася придатна не тільки для прототипування, але і для повноцінної розробки.

У жовтні 2015 року було випущено версію 1.0 бібліотеки, у вересні 2016 року - версія 2.0.

## 2.5 [Архітектура серверної частини додатку — технології розробки](#__RefHeading___Toc5084_346575917)

Серверна частина додатку, власне, є самим додатком — тут сконцентрована вся бізнес логіка додатку, організоване збереження даних, організована взаємодія з іншими провайдерами функціоналу для додатку.

Компанія Microsoft випустила свій фреймворк для веб додатків оснований на пдатформі .NET. Ця веб-орієнтований фреймворк називається ASP.NET (Active Server Page ).

В наш час веб додатки перестали бути звичайними сайтами чи блогами. Тепер існують справжні додатки, дуже могутні додатки, які використовують потужності багатьох комп’ютерів щоб функціонувати. Серед них є додатки, які вимагають оновлення клієнту у реальному часі, що означає, що клієнту не потрібно оновлювати сторінку, чи робити інший запит на сервер, щоб отримати нову інформацію. Такі додатки потребують технологій, що дозволяють серверу надавати дані без запиту на них клієнту. Наприклад, чати використовують технологію ВебСокетів для миттєвої доставки нового повідомлення користувачу, коли таке з’являється. Або системи моніторингу здоров’я додатку. Так трапляється, що великі хмарні, розподілені додатки часом перестають частково функціонувати. Для швидкої реакції на такі ситуації потрібна технологія миттєвого сповіщення оператора про подію. Навіть, гугл мапа використовує технологію ВебСокетів для оновлення маршрутів тощо.

Microsoft SignalR є реалізацією ВебСокетів у фреймворку ASP.NET.

Для збереження всієї інформації веб додатки використовують бази даних. В залежності від розміру додатку, розподіленості системи, архітектури, тощо, база даних може бути одна, а може буди багато різних баз даних.

### 2.5.1 Технологія ASP.NET

.NET - це платформа розробників, яка складається з інструментів, мов програмування та бібліотек для створення багатьох різних типів програм.

Базова платформа надає компоненти, які застосовуються до різних типів програм. Додаткові фреймворки, такі як ASP.NET, розширюють .NET з компонентами для створення конкретних типів програм.

Ось деякі речі, включені до платформи .NET:

* Мова програмування C # і його компілятори
* Базові бібліотеки для роботи з рядками, датами, файлами / IO та ін
* Редактори та інструменти для Linux, Windows, MacOS і Docker

ASP.NET розширює платформу .NET засобами та бібліотеками спеціально для створення веб-додатків.

Ось деякі речі, які ASP.NET додає до платформи .NET:

* Базова основа для обробки веб-запитів
* Синтаксис шаблонів веб-сторінок, відомий як Razor, для створення динамічних веб-сторінок за допомогою C #
* Бібліотеки для загальних веб-шаблонів, таких як Model View Controller (MVC)
* Система аутентифікації, яка включає бібліотеки, базу даних і сторінки шаблонів для обробки вхідних даних, включаючи багатофакторну аутентифікацію та зовнішню аутентифікацію з Google, Twitter і багато іншого.

Розширення редактора забезпечують підсвічування синтаксису, завершення коду та інші функції, спеціально для розробки веб-сторінок

### 2.5.2 Технологія 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 якісну зміну.

Розробка платформи 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.

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

ASP.NET Core може працювати поверх крос-платформного 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-додатку можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Хоча ASP.NET Core переважно націлений 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 версіях 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 С #.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або 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-платформною 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.

### 2.5.3 SignalR

Останнім часом в веб-середовищі все частіше створюються веб-додатки, що використовують технології комунікацій в реальному часі: це і прості чати, і більш складні розраховані на багато користувачів відеоконференції. Такі додатки можуть використовувати різні прийоми роботи: технологію Web Socket, опитування long polling і т.д. Для спрощення роботи з комунікаціями реального часу була створена спеціальна бібліотека під назвою SignalR.

При цьому SignalR не є бібліотекою, призначеною тільки для використання в ASP.NET MVC. Її також можна використовувати і в веб-формах, а також в консольних додатках, десктопних додатках і в WPF.

SignalR надає простий API для створення функціоналу, який дозволяє викликати функції JavaScript на стороні клієнта з серверного коду, написаного за допомогою мов платформи .NET. SignalR значно спрощує роботу з комунікаціями реального часу. Бібліотека обробляє всі підключення і автоматично розсилає повідомлення всім підключеним клієнтам, або яким-небудь специфічним клієнтам.

Фактично бібліотека SignalR складається з API серверної сторони, який застосовується в коді на C #, і з клієнтських бібліотек JavaScript.

SignalR надає розробникам дві моделі: постійні підключення (Persistent Connection) і хаби (Hubs).

Постійні підключення (Persistent Connection API) представляють розробникам прямий доступ до низкорівневого протоколу комунікації. Підключення в цій моделі представляють кінцеву точку, до якої підключаються клієнти, на зразок моделі підключень в WCF.

Хаби ж надають протокол взаємодії більш високого рівня. Вони представляють верхній шар над Persistent Connection API і дозволяють клієнту і серверу безпосередньо викликати методи один одного.

SignalR підтримується найбільш поширеними серверними платформами на базі ОС Windows:

* Windows Server 2012
* Windows Server 2008 r2
* Windows 8
* Windows 7
* Windows Azure

Для роботи на стороні сервера також необхідна версія фреймворку .net від 4.0 і вище.

* Підтримувані клієнтські платформи (браузери) теж різноманітні:
* Microsoft Internet Explorer 8, 9, 10, 11. Версії Modern, Desktop і Mobile
* Mozilla Firefox: як на ОС Windows, так і на Mac ОС
* Google Chrome: як на ОС Windows, так і на Mac ОС
* Safari: як на ОС Windows, так і на Mac ОС
* Opera: ОС Windows
* Android-браузер

Також слід враховувати, що браузер повинен підтримувати бібліотеку jQuery 1.6.4 або більш пізні версії.

**Транспорт передачі даних**

Для обміну даними між клієнтом і сервером SignalR використовує той спосіб передачі або той транспорт, який найбільш підходить до даної ситуації. Однак розробники можуть мати визначити пріоритет способу передачі. SignalR надає наступні типи технологій для взаємодії сервера і клієнта:

* WebSockets
* Server-sent events
* Forever Frames
* Long polling

### 2.5.4 База даних

База даних - сукупність даних, організованих відповідно до концептуальної структурою, яка описує характеристики цих даних і взаємини між ними, яка підтримує одну або більше областей застосування.

Потрібно зазначити, що бази даних бувають різних типів в залежності від даних, які потрібно зберігати. Для структурованих даних найкраще підходять реляційні бази даних, такі як MS SQL, MySQL, PostgreSQL. Для неструктурованих даних краще підходять документоорієнтовані або нереляційні бази даних, такі як MongoDB.

Реляційна база даних є цифровою базою даних, що базується на реляційній моделі даних, запропонованій Е.Ф.Коддом в 1970 році. Практично всі системи реляційних баз даних використовують SQL (Structured Query Language) для запитів і підтримки бази даних.

Реляційна модель організовує дані в одній або декількох таблицях (або "відносинах") стовпців і рядків з унікальним ключем, що ідентифікує кожен рядок. Рядки також називаються записами або кортежами. Стовпці також називаються атрибутами. Як правило, кожна таблиця / відношення представляє один "тип об'єкта" (наприклад, клієнт або продукт). Рядки являють собою екземпляри цього типу об'єкта (наприклад, "Лі" або "Стілець") і стовпці, що представляють значення, приписані цьому екземпляру (наприклад, адресу або ціну).

Відносини (Relationships) є одним з ключових аспектів реляційних баз даних. Відносини є логічним зв'язком між різними таблицями, встановленими на основі взаємодії між цими таблицями. Це значить,що будь-які дві таблиці можна пов’язати одна з одною через йх колонки.

Співвідношення визначається як набір кортежів, які мають однакові атрибути. Кортеж зазвичай представляє об'єкт і інформацію про цей об'єкт. Об'єкти зазвичай є фізичними об'єктами або поняттями. Відношення зазвичай описується як таблиця, яка організована у рядки та стовпці. Всі дані, на які посилається атрибут, знаходяться в одному і тому ж домені і відповідають таким самим обмеженням.

Реляційна модель вказує, що кортежі відносин не мають специфічного порядку і що кортежі, в свою чергу, не накладають ніякого порядку на атрибути. Програми звертаються до даних шляхом надання запитів, які використовують такі операції, як SELECT для ідентифікації кортежів, PROJECT для ідентифікації атрибутів і JOIN для об'єднання відносин. Відносини можуть бути змінені за допомогою операторів INSERT, DELETE та UPDATE. Нові кортежі можуть надавати явні значення або виводитися з запиту. Аналогічно, запити ідентифікують кортежі для оновлення або видалення.

Файли за визначенням унікальні. Якщо кортеж містить кандидат або первинний ключ, то, очевидно, він є унікальним; однак первинний ключ не повинен бути визначений для рядка або запису, який буде кортежем. Визначення кортежу вимагає, щоб він був унікальним, але не вимагав визначення первинного ключа. Оскільки кортеж унікальний, його атрибути за визначенням складають супер-ключ.

## 2.6 Безпека даних

Безпека веб-додатків знаходиться в першій десятці трендів і загроз інформаційній безпеці вже понад 10 років. Дійсно, сучасні бізнес-процеси і повсякденне життя - все більше і більше залежать від використання веб-додатків, в найрізноманітніших аспектах: від складних інфраструктурних систем до IoT пристроїв. Проте спеціалізованих засобів захисту веб-додатків досить мало, здебільшого це завдання покладають на розробників. Це і використання різних фреймворків, засобів санації, очищення даних, нормалізації і багато чого іншого. Проте, навіть з використанням цих засобів безпечніше веб не став, більш того, усі уразливості "класичного інтернету" практично в незмінному вигляді мігрували в мобільну розробку.

З чого складалася захист веб-додатків раніше? З грамотного налаштування веб-сервера, чищення сайту від непотрібних файлів і ділянок коду і мінімального контролю. Дійсно, канали були слабкі, DDoS ні поширений, вразливостей було мало, додатки були простими, дії користувачів передбачувані декількома сценаріями.

Згодом ускладнювалися веб-додатки, серверна інфраструктура і взаємодія, код ставав все більш об'ємним і громіздким - це набагато збільшило т.зв. "Поверхню атаки". Веб став дуже популярним, в ньому з'явилися можливості фінансового зростання - що природно привернуло в цю сферу бажаючих незаконно скористатися чужою працею.

Експоненціально стало рости кількість і різновид атак на веб-додатки, які умовно можна розділити на дві категорії (виходячи з концепції інформаційної безпеки):

* загрози спрямовані на порушення конфіденційності інформації;
* загрози спрямовані на порушення доступності інформації.

Тож захист даних і додатку повинен бути присутній на всіх рівнях, починаючи з аутентифікації користувачів до шифрування даних у базі даних.

### 2.6.1 Шифрування бази даних

Шифрування бази даних - використання технології шифрування для перетворення інформації, що зберігається в базі даних (БД), в шифротекст, що робить її прочитання неможливим для осіб, що не володіють ключами шифрування

Прозоре шифрування бази даних (англ. Transparent Database Encryption, TDE) - технологія, що застосовується, наприклад, в продуктах Microsoft і Oracle для шифрування і дешифрування введення-виведення файлів БД. Дані шифруються перед записом на диск і дешифруються під час читання в пам'ять, що вирішує проблему захисту «неактивних» даних, але не забезпечує безпеку інформації при передачі по каналах зв'язку або під час використання. Перевагою TDE є те, що шифрування і дешифрування виконуються прозоро для додатків, тобто їх модифікація не потрібно.

У базах даних від Microsoft TDE застосовується для файлів БД і журналу транзакцій на рівні сторінок. Сторінки шифруються за допомогою спеціального симетричного ключа шифрування бази даних (англ. Database Encryption Key), захищеного сертифікатом, який зберігається в БД master і шифрується її головним ключем, або асиметричним ключем, захищеним модулем розширеного керування ключами (англ. Extensible Key Manager, EKM) . Застосування TDE не збільшує розмір зашифрованої БД, а вплив на продуктивність незначний.

Проте для зберігання чутливої інформації (такої як паролі, ключі доступу) цього не достатньо, оскільки дані можуть бути отримані через SQL-ін’єкцію у розшифрованому вигляді. Для забезпечення безпеки таких даних використовується додаткове хешування (для паролів) чи шифрування.

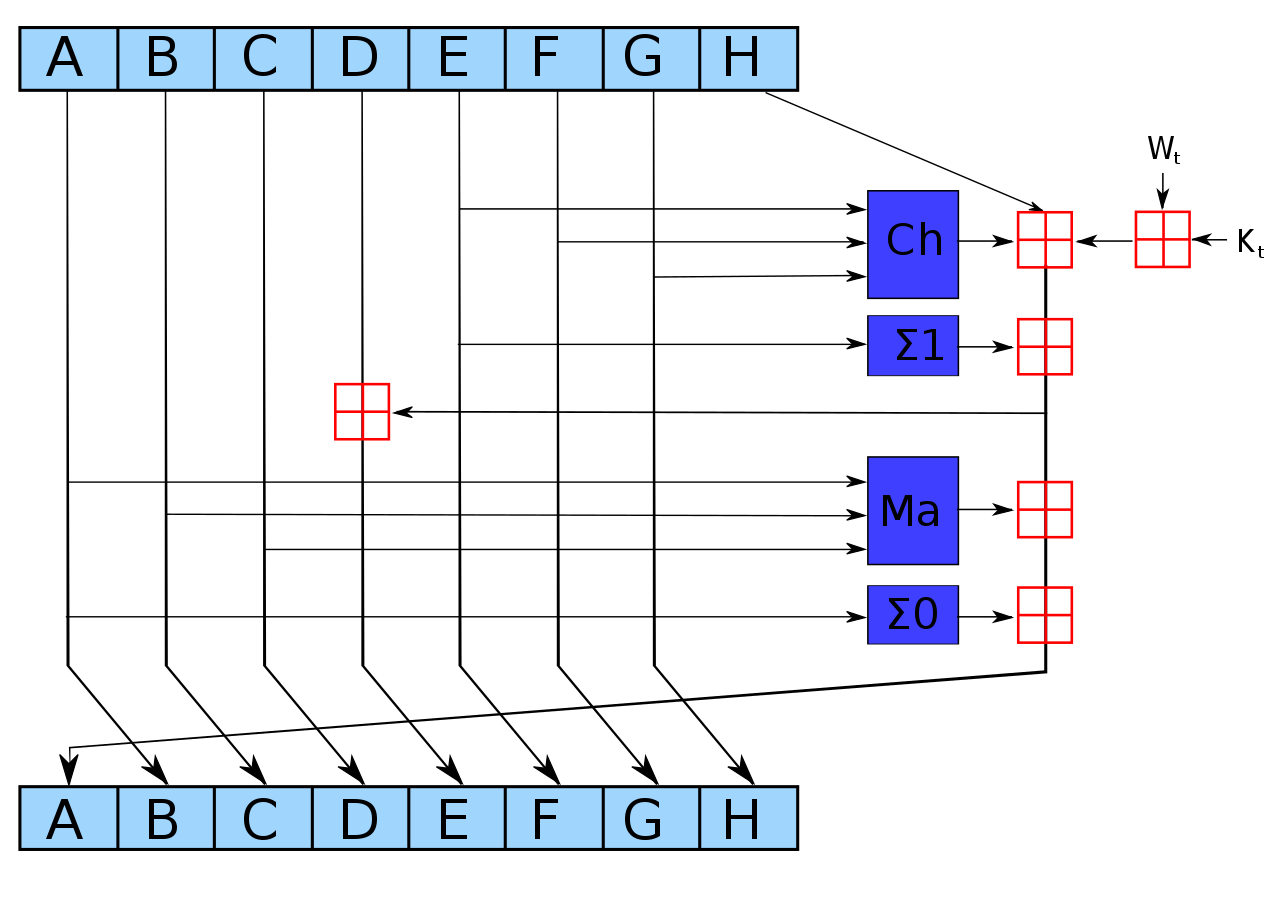
### 2.6.2 Хешування з SHA-256

SHA-256 являє собою односпрямовану функцію для створення цифрових відбитків фіксованої довжини (256 біт, 32 байт) з вхідних даних розміром до 2,31 ексабайт (2⁶⁴ біт) і є окремим випадком алгоритму з сімейства криптографічних алгоритмів SHA-2 (Secure Hash Algorithm Version 2) опублікованим АНБ США в 2002 році.

Хеш-функції сімейства SHA-2 побудовані на основі структури Меркле - Дамгарда.

Оригінал тексту після доповнення розбивається на блоки, кожен блок - на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64 ітераціями. На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням хеш-функції. Так як ініціалізація внутрішнього стану проводиться результатом обробки попереднього блоку, то немає можливості обробляти блоки паралельно. Графічне представлення однієї ітерації обробки блоку даних:

На поточний момент відомі методи для конструювання колізій до 31 ітерації. З огляду на алгоритмічні схожості SHA-2 з SHA-1 і наявності в останньої потенційних вразливостей прийнято рішення, що SHA-3 буде базуватися на зовсім іншому алгоритмі. 2 жовтня 2012 року NIST затвердив в якості SHA-3 алгоритм Keccak.

  
рисунок 1: Алгоритм SHA-2

### 

### 2.6.3 Шифрування з AES

Advanced Encryption Standard (AES), також відомий як Rijndael (вимовляється [rɛindaːl] (Рендал)) - симетричний алгоритм блочного шифрування (розмір блоку 128 біт, ключ 128/192/256 біт), прийнятий в якості стандарту шифрування урядом США за результатами конкурсу AES. Цей алгоритм добре проаналізований і зараз широко використовується, як це було з його попередником DES. Національний інститут стандартів і технологій США (англ. National Institute of Standards and Technology, NIST) опублікував специфікацію AES 26 листопада 2001 після п'ятирічного періоду, в ході якого були створені і оцінені 15 кандидатур. 26 травня 2002 року AES був оголошений стандартом шифрування. Станом на 2009 рік AES є одним з найпоширеніших алгоритмів симетричного шифрування [2] [3]. Підтримка AES (і тільки його) введена фірмою Intel в сімейство процесорів x86 починаючи з Intel Core i7-980X Extreme Edition, а потім на процесорах Sandy Bridge.

AES є стандартом, заснованим на алгоритмі Rijndael. Для AES довжина input (блоку вхідних даних) і State (стану) постійна і дорівнює 128 біт, а довжина шіфроключа K становить 128, 192, або 256 біт. При цьому вихідний алгоритм Rijndael допускає довжину ключа і розмір блоку від 128 до 256 біт з кроком в 32 біта. Для позначення обраних довжин input, State і Cipher Key в 32-бітових словах використовується нотація Nb = 4 для input і State, Nk = 4, 6, 8 для Cipher Key відповідно для різних довжин ключів.

Шифрування:

1. На початку шифровування input копіюється в масив State за правилом *state[r, c] = input[r + 4c]*, для *0 <= r < 4* та *0 <= c < Nb*. Після цього до State застосовується процедура AddRoundKey(), і потім State проходить через процедуру трансформації (раунд) 10, 12, або 14 раз (в залежності від довжини ключа), при цьому треба врахувати, що останній раунд дещо відрізняється від попередніх. У підсумку, після завершення останнього раунду трансформації, State копіюється в output за правилом *output[r + 4c] = state[r, c]*, для *0 <= r < 4*  і *0 <= c < Nb*.
2. Процедура SubBytes () обробляє кожен байт стану, незалежно виробляючи нелінійну заміну байтів, використовуючи таблицю замін (S-box). Така операція забезпечує нелінійність алгоритму шифрування. Побудова S-box складається з двох кроків. По-перше, проводиться взяття зворотного числа в полі Галуа *GF(2)*. По-друге, до кожного байту b, з яких складається S-box, застосовується операція побітового додавання та взяття за модулем. Таким чином забезпечується захист від атак, заснованих на простих алгебраїчних властивостях.
3. ShiftRows працює з рядками State. При цій трансформації рядки стану циклічно зсуваються на r байт по горизонталі залежно від номера рядка. Для нульовою рядки *r = 0*, для першого рядка *r = 1* Б і т. Д. Таким чином, кожна колонка вихідного стану після застосування процедури ShiftRows складається з байтів з кожної колонки початкового стану. Для алгоритму Rijndael патерн зсуву рядків для 128 і 192-бітних рядків однаковий. Однак для блоку розміром 256 біт відрізняється від попередніх тим, що 2-е, 3-й і 4-е рядки зміщуються на 1, 3 і 4 байта відповідно.
4. У процедурі MixColumns чотири байти кожної колонки State змішуються, використовуючи для цього оборотну лінійну трансформацію. MixColumns обробляє стани по колонках, трактуючи кожну з них як поліном третього ступеня. Над цими поліномами проводиться множення в *GF(2)* по модулю *x + 1*  на фіксований многочлен {\ displaystyle *c (x) = 3x + x + x + 2.* Разом з ShiftRows MixColumns вносить дифузію в шифр.
5. У процедурі AddRoundKey RoundKey кожного раунду об'єднується з State. Для кожного раунду RoundKey виходить з CipherKey c допомогою процедури KeyExpansion; кожен RoundKey такого ж розміру, що і State. Процедура виробляє побітовий XOR кожного байта State з кожним байтом RoundKey

## 2.7 Висновки

У нинішній час на ринку присутня велика кількість програмних застосунків для обміну повідомленнями. Ці застосунки відрізняються призначенням: для корпоративних цілей або для особистих цілей; за функціоналом: від месенджера для лише обміну повідомленнями до соціальної мережі з розширеними можливостями публікації новин, коментарів і посилань. Всі месенджери є захищеними у тій чи іншій мірі, проте багато з них мають недоліки у безпеці, які вирішуються, або ігноруються. З точки зору безпеки Телеграм є найбезпечнішим месенджером, оскільки надає шифрування даних на всіх рівнях застосунку, окрім цього надає можливість збереження даних лише на девайсах користувачів таким чином не зберігаючи даних у базах даних.

В залежності від розміру веб застосунку його архітектура може бути монолітною чи мікросервісною. Приймаючи до уваги, що більшість мессенджерів є застосунками із багатою серверною частиною має сенс проектувати мессенджер, що буде задовольняти мікросервісну архітектуру.

Важливою ознакою мессенджера, що і являється відмінною характеристикою від імейлу є миттєвість надходження повідомлень і пуш-повідомлення користувача. Отже, важливою технологією, що дозволила мессенджерам стати такими, якими мі їх знаємо є технологія веб-сокетів. На теперішній час усі сучасні браузери підтримують веб-сокети, що робить месенджери ще більш популярними.

Платформа .NET Framework, а точніше її відгалуження ASP.NET є дуже гарним вибором для рішення такого роду задач. ASP.NET (Core) надає зручні механізми для розробки серверної та клієнтської частин застосунку в одному редакторі одній людині. Це можливе задяки тому, що платформа ASP.NET надає можливість написання клієнтської частини застосунку використовуючи сторінки Razor, які є поєднанням HTML розмітки із C# кодом, що використовується на сервері. Одже, це робить ASP.NET дуже підходящою платформою для створення веб застосунків і, особливо, високо навантажених застосунків. Технологія SignalR дозволяє досить легко сконфігурувати канал для веб сокетів і надалі створювати хаби, які будуть обробляти запити і надсилати пуш-повідомлення до клієнтського додатку (веб сторінки). Основним недоліком платформи ASP.NET і, взагалі, платформи .NET є вимога до спеціалізованого сервера (сервер має мати операційну систему Windows та на ньому має бути встановлений IIS сервер або Kestrel), на якому буде запущений серверний додаток.

## 

# РОЗДІЛ 3 РЕАЛІЗАЦІЯ І ВПРОВАДЖЕННЯ ВЕБ ЗАСТОСУНКУ

# НА ОСНОВІ ASP.NET WEB API

## 3.1 Загальний опис застосунку

Застосунок “Месенджер” є прикладом написання чат-застосунку, тобто такого, який дозволяє користувачам обмінюватися повідомленнями у режимі реального часу, застосунок має більш менш зручний інтерфейс користувача і прийнятну швидкодію. Застосунок буде складатися з двох основних сторінок: головна сторінка та сторінка реєстрації/авторизації. На головній сторінці авторизований користувач може обрати чат, або створити новий, може написати повідомлення до обраного чату і прочитати повідомлення і може бачити, хто з користувачів зараз онлайн у вибраному чаті. Неавторизований користувач направляється на сторінку авторизації, де він може ввести свої ідентифікаційні дані або перейти на допоміжну сторінку реєстрації і там створити свій обліковий запис. Після реєстрації користувач автоматично авторизується, після авторизації користувач перенеправляється на головну сторінку.

Стосовно інтерфейсу користувача, він простий. Головна сторінка розділена на три колонки і складається з таких головних компонентів: секція із списком чатів, секція чату із повідомленнями, секція із списком користувачів, що знаходяться у чаті. Секція із списком знаходиться зліва, у верху секції є кнопка для створення нового чату, внизу секції є підсекція із даними користувача: його фотографією та ім’ям. Також у підсекції із профілем користувача справа є кнопка виходу. Середня секція є секцією чату із повідомленнями і є найбільшою з трьох, адже є головним компонентом застосунку. У шапці секції повідомлень є назва чату та кнопка видалення чату. Не можна видалити стандартний чат. Внизу секції чату є поле для вводу повідомлення із можливістю завантажувати картинки та додавати базові емоджі. Третя секція являється додатковою, в ній можна побачити саисок користувачів онлайн. Внизу секції є поле для пошуку користувача за його ім’ям.

Сторінка авторизації має два поля: поле для логіну та поле для пароля. І кнопка для входу. Якщо логін не знайдено чи пароль не співпадає, то користувач інформується про це шляхом отримання відповідного повідомлення. Для реєстрації нового користувача в системі потрібно натиснути відповідне посилання, що знаходиться на сторінці авторизації під кнопкою підтвердження. Сторінка реєстрації нового користувача складається з чотирьох полей. Всі поля досить стандартні: поле логіну — має бути унікальним у системі, є ключем/ідентифікатором користувача, поле для ім’я, що буде зображене на сайті і яке будуть бачити інші користувачі, поля для пароля і підтвердження пароля. Якщо паролі не задовольняють умовам паролей, чи паролі не співпадають, або логін користувача вже зайнятий то користувач отримає відповідне повідомлення.

## 3.2 Проектування бази даних за стосунку

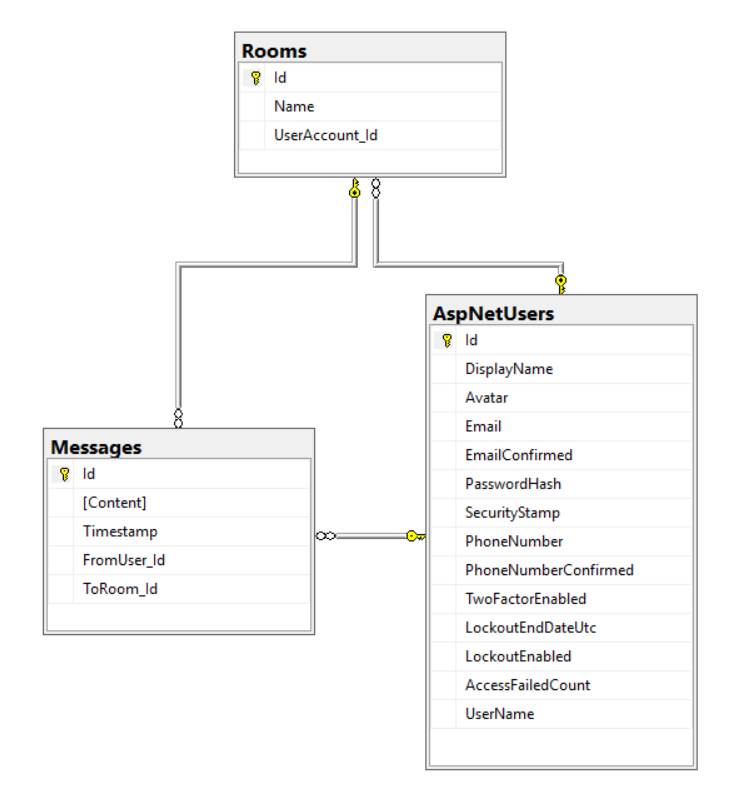


Рисунок 1. Модель повідомлення у БД

Для початку створимо модель повідомлення, оскільки повідомлення є основним компонентом застосунку. Модель створимо із основних полей, що потрібні для обміну повідомленнями. Ці поля такі:ID повідомлення – позначка, що унікально ідентифікує повідомлення; Content – поле, в якому, власне, зберігається зміст самого повідомлення; Timestamp – часова позначка повідомлення (час відправлення), для можливості сортування повідомлень по часу відправлення та для відображення користувачу. FromUser\_Id – унікальний ідентифікатор користувача – відправника повідомлення. ToRoom\_Id – унікальний ідентифікатор кімнати, до якої надсилається повідомлення, всі користувачі, що знаходяться у кімнаті, побачать це повідомлення.

Модель користувача використаємо базову. Базова модель користувача створюється при підключенні Identity Framework. Одразу після підключення та увімкнення міграцій першою міграцією створиться модель користувача та додаткові моделі, що використовуються при авторизації.

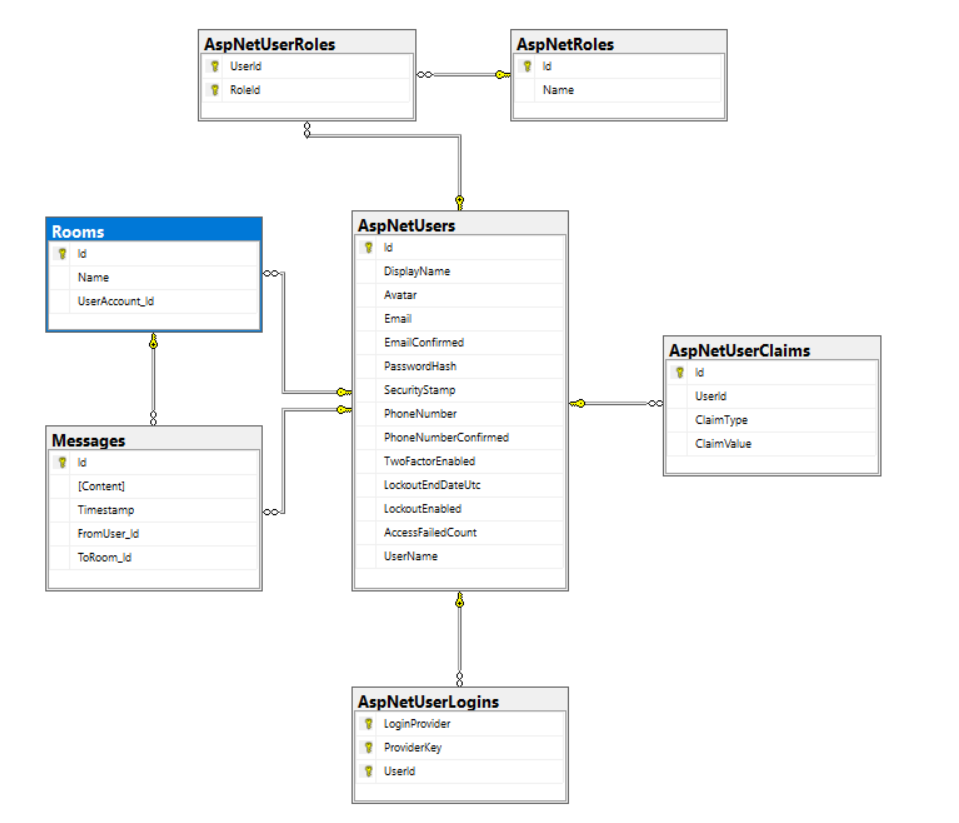


Рисунок 2. Схема бази даних

Спершу спроектуємо модель користувача для збереження його персональних даних. DisplayName – ім’я користувача, що буде відображене на сайті, Avatar – посилання на картинку з аватаром користувача, Email – електронна адреса користувача (використовується для авторизації користувача а також у випадках відновлення паролю та інших, якщо потрібно зв'язатися з користувачем), UserName – системне ім'я користувача, що той обирає власноруч, також слугує для ідентифікації користувача, може використовуватись замість email під час авторизації, EmailConfirmed – поле, що зберігає дані чи електронна адреса підтверджена, чи ні.

Під час реєстрації нового користувача вводиться нова електронна адреса, проте система не може гарантувати, що той, хто ввів адресу дійсно має до неї доступ. Для підтвердження власності електронної адреси на неї надсилається повідомлення підтвердження із посиланням на підтвердження адреси. Якщо користувач перейшов по посиланню, то адреса підтверджена і в полі EmailConfirmed ставиться помітка, що адреса пітверджена. В нашому випадку це поле є бітовим (аналог до типу Boolean у C#) і може приймати два значення: 0 – не підтверджена, 1 – підтверджена.

Для збереження пароля використовується його хеш, тобто зберігається не пароль, а хеш. Цьому є принаймні одна вагома причина: якщо дані з бази даних розкриються, то ті, хто отримав дані не отримають паролей від аккаунтів користувачів – лише хеш. Одже для збереження паролю використовуємо поле PasswordHash, що є типу VARCHAR(max).

Поле PhoneNumber зберігає номер телефону користувача, якщо є, а поле PhoneNumberConfirmed зберігає інформацію про підтвердження номеру телефону. Підтвердження номеру телефону відбувається таким же чином, як і підтвердження електронної адреси, лише повідомлення підтвердження надсилається через смс.

TwoFactorEnabled – поле, що визначає чи активована двофакторна аутентифікація для користувача. Двофакторна аутентифікація (також відома як 2FA) - це тип або підмножина багатофакторної аутентифікації. Це метод підтвердження ідентичності користувачів за допомогою комбінації двох різних факторів: 1) те, що вони знають; 2) щось у них є, або 3) те, що вони є. Хорошим прикладом двофакторної аутентифікації є отримання грошей з банкомату; тільки правильне поєднання банківської картки (що користувач має) і PIN-код (що користувач знає) дозволяє здійснити транзакцію. Двоетапна перевірка або двоступенева автентифікація - це метод підтвердження заявленої ідентичності користувача, використовуючи те, що він знає (пароль), і другий чинник: що він має або чим є. Прикладом другого етапу є користувач, який повторює щось, що було надіслано йому через третій механізм. Або другий крок може бути шестизначним числом, створеним програмою, яка є спільною для користувача та системи аутентифікації

Поля LockoutEndDateUtc, LockoutEnabled, AccessFailedCount – є допоміжними полями, що допомагають сконфігурувати авторизацію, кількість помилково введених паролей до того, як користувач буде заблокований, час блокування користувача, чи така функція включена для даного користувача.

Доступ до деяких функцій за стосунку, наприклад: панель адміністратора - має бути обмеженим. Для обмеження доступу (авторизації) використовуються ролі. Таблиця AspNetRoles зберігає всі ролі у форматі Id – ідентифікатор ролі, Name – назва ролі. Користувач може мати декілька ролей, а до одної ролі можуть належати багато користувачів. Одже для зв’язку користувачів з ролями використовується проміжна таблиця (Junction Table), що називається AspNetUserRoles. Ця таблиця лише зв’язує користувачів і ролі, одже має поля UserId – вказує на ідентифікатор користувача, RoleId – вказує на ідентифікатор ролі. Одже один запис у таблиці відображає приналежність одного користувача до однієї ролі.

## 3.3 Архітектурне рішення застосунку

Наш месенджер буде простим застосунком із ключовою можливістю обміну повідомленнями. Додатково планується зробити базову реєстрацію та авторизацію у першій ітерації розробки. Із огляду на те, що за стосунок не планується сильно розширювати у майбутньому нам буде достатня MVC модель. Також мі відмовимось від використання спеціальних фронтенд фреймворків на користь простоти та швидкості розробки, більше того ніякого дуже навантаженого UI не передбачається, що робить використання фронтенд фреймворків необгрунтовано ускладненим. Таким чином наш Месенджер буде написаний на ASP.NET MVC фреймворку і будемо використовувати вбудовані Razor сторінки.

Стандартне MVC рішення передбачає три шари за стосунку:

* Шар даних (дані у базі даних MS SQL і фреймворк для роботи з даними у C# Entity Framework – дуже популярний фреймворк, завдяки його простоті використання у коді)
* Шар бізнес-логіки (виконаний на C#)
* Шар представлення (виконаний на Razor pages)

Одже, розглянемо use case діаграму для того, щоб зрозуміти основні компоненти, що нам потрібні.

### 3.3.1 Use-case діаграма

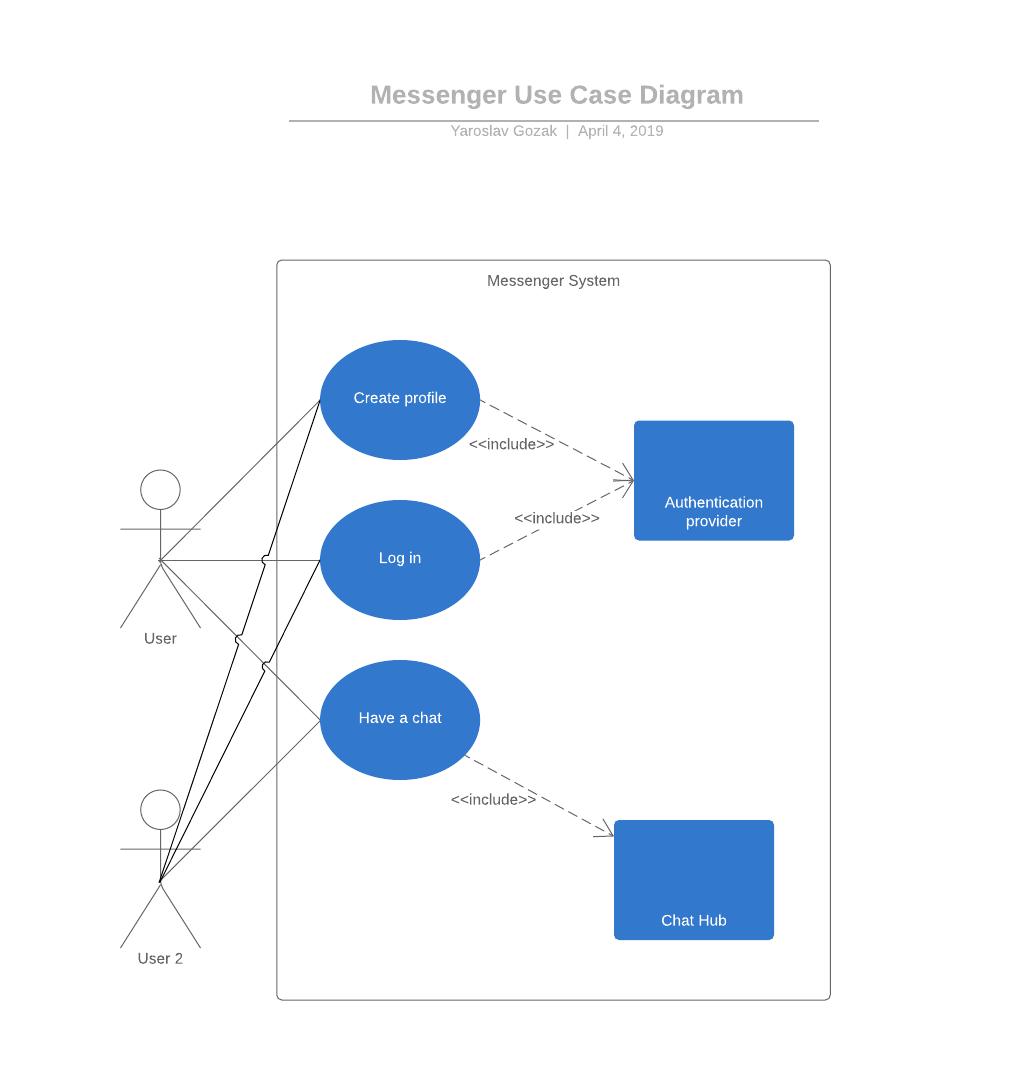


Рисунок 3. Use-Case діаграма

При переході на сторінку Месенджеру система визначає чи авторизований користувач за допомогою атрибута [Authorize] та сервісу авторизації, що на малюнку зображений як Authentication Provider. Якщо користувач не авторизований, то він буде перенаправлений на сторінку Login, де зможе ввести свої дані. Коли користувач введе дані і натисне кнопку «Увійти», то сторінка надішле запит до Authentication Provider, який перевірить коректність введених даних. Якщо спроба успішна, то користувач буде направлений на сторінку Home (на малюнку Have a chat). Якщо спроба неуспішна, то користувач повернеться на сторінку авторизації, де додатково отримає повідомлення, що описує помилку.

Якщо користувач ще не має профілю, то зможе перейти на сторінку реєстрації. Увівши необхідні дані та натиснувши кнопку «Зареєструватись» користувач зініціює запит сторінки на сервер через AccountController до Authentication Provider (що представлений AppSigninManager). Якщо всі дані валідні, то користувач буде авторизований і направлений на сторінку HomePage (на малюнку Have a chat). Якщо під час реєстрації користувача виникне помилка, то користувач буде заново направлений на сторінку реєстрації де додатково побачить повідомлення, що описує помилку.

На головній сторінці користувач може створити новий чат або приєднатися до одного з вже існуючих. Иакож користувач може видаляти чати. Увійшовши до обраного чату, користувач може написати повідомлення.

При переході на головну сторінку між JavaScript застосунком та сервером встановлюється з’єднання через веб сокети. Надалі всі дії із чатами (створення, видалення, з’єднання) відбувається через веб сокети, всі запити надходять до хабу ChatHub і надалі з нього же надсилаються сповіщення JavaScript застосунку у браузері клієнта.

Технічно при заході на сторінку Мессенджеру система має верифікувати користувача чи він авторизований, чи ні. У разі якщо користувач не авторизований, то він має бути направлений на сторінку авторизації, яка має також надати можливість створити новий профіль користувача при потребі.

Одже, одним з перших компонентів буде сервіс авторизації і аутентифікації. Оскільки авторизація і аутентифікація є задачами, із якими стикається майже кожен веб застосунок і оскільки шляхи вирішення авторизації і аутентифікації є досить стандартними, то платформа ASP.NET Core представляє свій сервіс авторизації і аутентифікації користувачів **Identity.** Identity надає досить широкі можливості роботи з користувачами, розглянемо деякі з них:

Підтримка авторизації і аутентифікації користувачів через кукі, в тому числі claims. Claims – це дані про користувача у кукі, які зберігають в собі такі дані як хто видав ці дані, яка роль користувача, чи, навіть, яке ім’я користувача. Насправді claims є найбільш вільною у використанні системою ідентифікації користувачів. Взагалі, користувача можна аутентифікувати за ім’ям, що має бути унікальним, токеном чи за клеймами. Проте, авторизація може бути проведена також за допомогою ролей, які в свою чергу є заздалегідь створеними системою клеймами.

Для чого це потрібно? Наприклад, ми хочемо створити адміністратора, який зможе видаляти користувачів. Звісно, адміністратором не може бути кожен користувач, проте нам потрібно декілька адміністраторів, які зможуть це робити. І, звісно, нам потрібні самі користувачі. Тож для доступу до контролеру видалення користувачів повінні мати лише адмвнвстратори. Для цього чудово підходять ролі. Користувачі, що є адміністраторами наділяються роллю Адміністратор і контролер видалення користувачів конфігурується бути доступним лише користувачам із роллю Адміністратор. Це чудове і елегантне рішення схожих задач. Із Identity доступ до контролера чи окремого його метода (екшену) може бути сконфігурований через C# атрибути. На контролер, що має бути обмежений у доступі навішується атрибут [Authorize()] і туди передається список ролей чи/та імен, яким дозволений доступ до контролера.

Також Identity надає зручний спосіб керування користувачами, а саме: авторизація та створення нового аккаунту. Створення та авторизація відбувається через методи класу AppSigninManager, а управління аккаунтом відбувається через сервіс AppUserManager, які є вбудованими. Зазвичай для отримання можливості використання цих сервісів потрібно успадкувати ці класи і використовувати, власне, новостворені класи.

Вся інформація про користувачів, ролі та інші допоміжні дані зберігаються у базі даних, яка створюється автоматично за допомогою Entity Framework та механізму міграцій. Для того, щоб створити перший раз базу даних для Identity потрібно виконати команди Allow-Migrations та New-Migration і потім Commit-Changes у консольному менеджері. Цей вбудований функціонал Entity Framework та Identity дозволяють не створювати всю базу даних вручну.

Одже, наш застосунок має три основні сторінки: головна, на якій можна обрати чат і написати повідомлення, сторінка реєстрації і сторінка авторизації. Частина View нашого шаблону MVC матиме 3 сторінки відповідно. HomePage, LoginPage та RegistrationPage. Але кортролери буде лише два, оскільки сторінки авторизації і реєстрації мають дуже схожу та у більшості спільну логіку. Контролери HomeController для HomePage та AccountController для сторінок LoginPage та RegisterPage. Тож AccountController опрацьовує дві сторінки. До речі, нормальним є відношення сторінок до контролерів як 5 до 1.

Усі операції із маніпуляції даних будуть обробляти вище зазначені сервіси: AppUserManager та AppSigninManager.

## 3.4 Інструкція користувача

Застосунок не є великим, весь його функціонал зосереджений на головній сторінці. Одже, коли користувач вперше спробує перейти на головну сторінку застосунку, то будучи неавторизованим користувач буде направлений на сторінку авторизації

На сторінці авторизації (Рисунок 4) є три стандартні поля для введення даних користувача:

* + - 1. Ім'я користувача (Username)
      2. Пароль (Password)
      3. Маркер чи запам'ятати цього користувача на даному девайсі (Remember me)

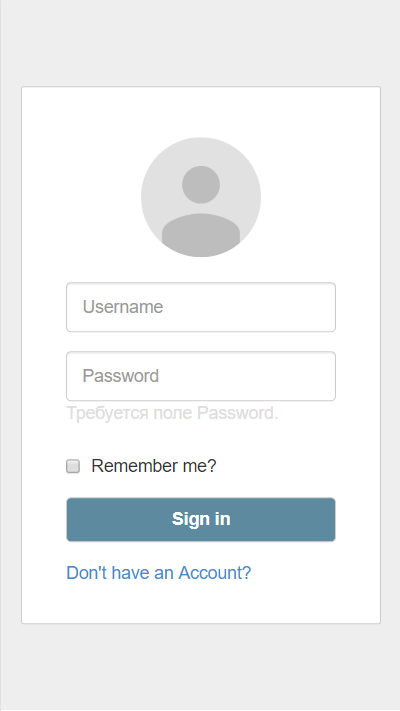


Рисунок 4. Сторінка авторизації. Мобільна версія

Дані валідуються, щоб ім'я користувача не повторювалось у системі – було унікальним і щоб пароль задовольняв політиці паролей застосунку. Як зазначалося вище, політика паролей конфігурується черех Identity Framework.

Після вводу даних користувач натискає на кновку Sign in і, якщо дані введені правильно, користувач успішно авторизований і направляється на основну сторінку. Якщо користувач не має власного профілю у застосунку, то може перейти за посиланням «Don’t have an Account?» і перейти на сторінку авторизації.

На сторінці авторизації присутня більша кількість полей для даних, оскільки застосунок має отримати мінімальну достатню кулькість інформації про користувача для продовження роботи з ним. Розглянемо поля для даних:

* 1. Ім’я користувача (Username) – унікальне ім’я користувача, за яким буде відбуватися авторизація та ідентифікація користувача у системі
  2. Пароль (Password)
  3. Повторити пароль (Confirm Password)
  4. Зображуване ім’я (Display Name) – ім’я, яке буде зображене у за стосунку

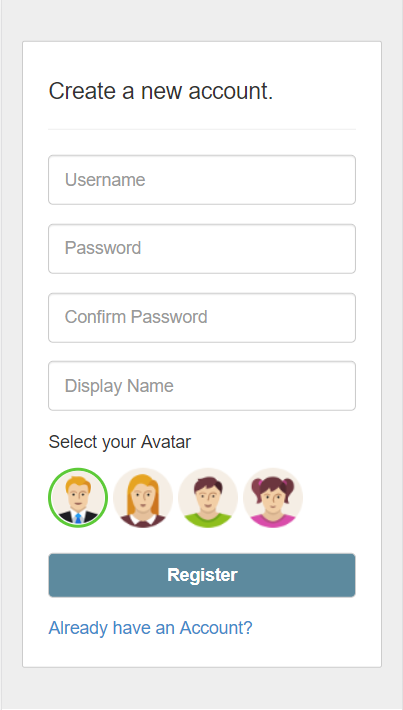


Рисунок 5. Сторінка реєстрації. Мобільна версія

Одже, ввівши дані у всі поля, користувач натискає кнопку «Зареєструватися». Якщо всі поля заповнені правильно, то користувач буде направлений на головну сторінку хастосунку.

На головній сторінці в першу чергу буде відображено список груп (рис. 6), до яких користувач має доступ, та буде підсвічено назву групи, повідомлення з якої користувач зараз бачить. Натискання на кнопку меню у правому верхньому кутку сховає список груп, а замість нього користувачу буде показано діалогове вікно групи. До секції діалогу групи можна також дістатися проскроливши вниз, адже секція діалогу знаходиться прямо під секцією меню.

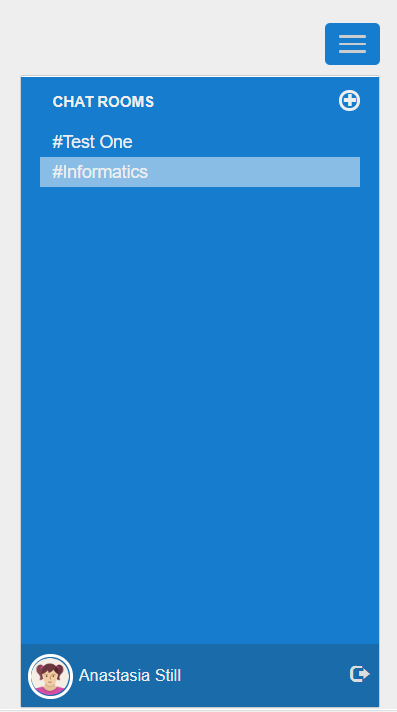


Рисунок 6. Секція меню. Мобільна версія

Діалогова секція (рис. 7) спроектована бути якнайпростішою. Коли секція меня схована, то у верхньому правому кутку є іконка меню, натиснувши на яку, секція меню знову буде відкрита і користувач може змінити діалог або вийти.

Сама секція діалогу має хедер (шапку секції), який містить назву групи, яку переглядає користувач. Справа від назви групи є кнопка видалення групи. Групу може видалити лише її засновник. Далі йде основна секція із повідомленнями групи. Ця секція найважливіша і містить найбільшу кількість інформації, одже і займає набільше місця.

У футері розміщено текстове поле, в яке користувач може ввести текст для відправлення у групу. У правій частині текстового поля розміщено допоміжні іконки: іконка смайлику відкриває список емоджі, які користувач може відправити; іконка паперового літачка дозволяє відправити повідомлення – всі користувачі десктоп звикли відправляти повідомлення натисканням клавіші «Enter», проте на телефоні такої клавіші немає; іконка картинки дозволяє додати зображення до повідомлення користувача – проте не всі типи картинок підтримуються.

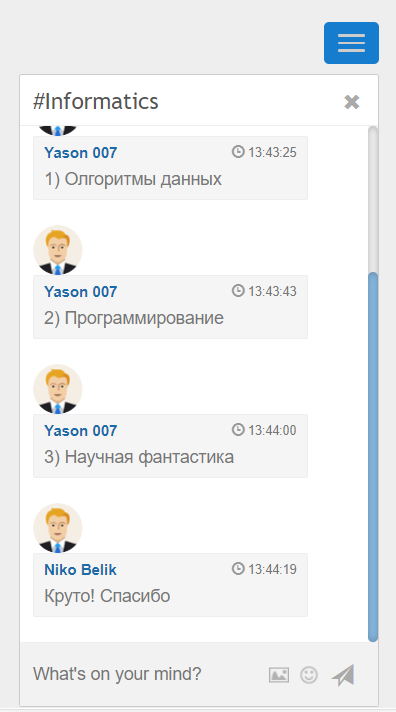
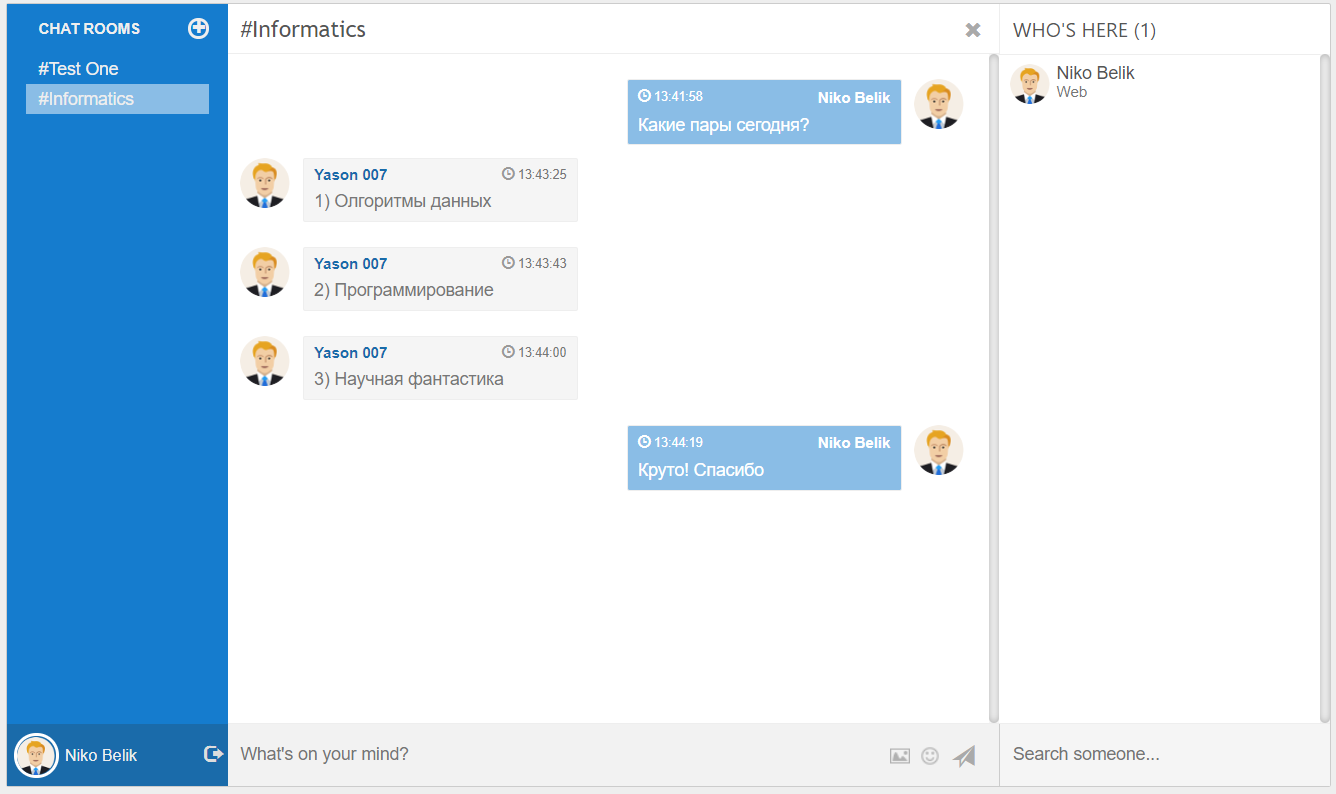


Рисунок 7. Секція діалогу групи. Мобільна версія

Версія головної сторінки для десктоп виглядає трішки інакше. Завдяки великому розміру екрану є більше місця щоб розмістити секції головної сторінки застосунку. Таким чином секція меню просто розташована зліва екрану, а секція діалогу групи розташована посередині екрану і займає трохи більше половини всього місця. Справа розташована додаткова секція, що відображає список користувачів онлайн, а точніше користувачів, що зараз переглядають даний діалог. У нижній частині секції користувачів онлайн знаходиться текстове поле, яке дозволяє шукати користувачів за їх ім’ям.



## 3.5 Висновки

На основі ASP.NET MVC було реалізовано прототип застосунку для обміну повідомленняму у режимі реального часу. Для досягнення цих цілей було використано технологію .NET для веб-сокетів SignalR. Наявність більшості основоного функціоналу вже у готовому вигляді, в тому числі: реалізація технології веб-сокетів, фреймворк авторизації та аутентифікації користувачів – значно прискорює розробку основоного функціоналу, базового каркасу застосунку. Задяки наявності автоматизованої системи генерації таблиць у базі даних для збереження інформації про користувача, а також автоматичного мапінгу у моделі користувача можна не витрачати час на розробку функціоналу, що є поширеним серед різних застосунків, тобо можна використовувати уніфікований шаблон, що підходить більшості проектів.

# ВИСНОВКИ

Було досліджено існуючі веб-застосунки для обміну повідомленнями, а саме: Telegram – один з найпопулярніших застосунків серед користувачів усього світу; Viber – популярний за стосунок, в Україні переважно використовується старшим поколінням; Messenger – за стосунок від фейсбук; WhatsApp – популярний за стосунок у Європі та Штатах, має проблеми із захищеністю.

Telegram також являється еталоном безпеки, адже і створювався як застосунок, що гарантує приватність спілкування. Маючи наскрізне шифрування і розподілене збереженя повідомлент у зашифрованому вигляді та ключів до шифрів Telegram вважається найбільш захищеним застосунком для спілкування.

Viber створювався із ціллю надати безпеку спілкування шляхом відсутності збережених повідомлень у своїх базах даних. Таким чином Viber являється лише брокером повідомлень між користувачами. Проте збереження повідомлень лише на стороні клієнта – на девайсах користувачів приводить до відсутності можливості відновлення історії повідомлень на новому девайсі навіть із прив’язкою до існуючого акаунту. Із цієї точки зору Viber є незручним у використанні, адже багато користувачів мають декілька гаджетів і дужк бажано, щоб при підключенні нового гаджета до акаунту застосунок вів себе так, наче був підключений завжди. Проте відсутність історії на серверах Viber не дозволяє цього зробити.

Messenger та WhatsApp надають користувачам досить непоганий досвід використання, проте також мають свої недоліки, в тому числі у сфері безпеки та приватності.

Було проаналізовано наявні архітектурні рішення. З огляду на існуючі архітектури застосунки типу Месенджер, тобто застосунки для обміну повідомленнями можуть проектуватися як монолітні застосунки, а також як мікросервісні. Обидві архітектури можуть бути доречними, проте це вже залежить від розмірів застосунку та планів на його подальше розширення. Для простого застосунку, який лише надає базову можливість обмінюватися повідомленнями у режимі реального часу досить монолітної архітектури, де всі компоненти та шари (шпр доступу до даних, шар бізнес-логіки, презентаційний шар) знаходяться у одному проекті, у випадку із .NET, в одному рішенні (Solution). Це дозволяє легко маніпулювати будь-якими компонентами системи, а також заощаджує час на обробку запиту сервером, так як відсутня необхідність спілкування по HTTP різних мікросервісів між собою та знижує залежність від мережі. Проте, якщо застосунок планується бути постійно розширюваним, або початкові плани на застосунок досить грандіозні, то варто розглядати мікросервісну архітектуру як рішення. Мікросервісна архітектура дозволяє винести одиницю бізнесу наприклад: логіка обміну повідомленнями або логіка авторизації користувача – у окремий самодостатній сервіс – мікросервіс. Це дозволяє розграничити сфери відповідальності ккомпонентів, а також дозволяє горизонтально масштабувати окремі компоненти за стосунку, наприклад, сервіс обробки замовлень, тощо.

Було розроблено прототип застосунку для спілкування у режимі реального часу «Месенджер» на основі технологій ASP.NET. Реалізований застосунок має весь базовий функціонал, який притаманний програмному забезпеченню такого роду, а саме, чатам: авторизація користувачів, відправка повідомлення, отримання повідомлення. Цей застосунок свідчить про те, що платформа .NET, а точніше, фреймворк ASP.NET є підходящим інструментарієм для розробки веб-застосунків, в тому числі, за стосунків-месенджерів.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Ріхтер Дж. Г. CLR via C#, вид. 4 / Ріхтер Дж. Г. – Санкт-Пітербург : вид. «Питер», 2018. – 896 с.
2. Cesar de la Torre. .NET Microservices: Architecture for Containerized .NET Applications / Cesar de la Torre, Wagner B., Rousos M – Redmond: Microsoft Developer Division, 2018, – 309 c.
3. Ritcher J. .NET Applications: Architecture .NET Applications / Ritcher J, Wagner B., Rousos M – Redmond: Microsoft Developer Division, 2017, – 209 c.
4. Вікіпедія [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Київ: НБУВ, 2013-2019. – Режим доступу: [www.uk.wikipedia.org/wiki/Telegram](http://www.uk.wikipedia.org/wiki/Telegram) (дата звернення 30.03.2019) – Telegram.
5. Вікіпедія [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Київ: НБУВ, 2013-2019. – Режим доступу: [www.uk.wikipedia.org/wiki/Viber](http://www.uk.wikipedia.org/wiki/Viber) (дата звернення 30.03.2019) – Viber.
6. Вікіпедія [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Київ: НБУВ, 2013-2019. – Режим доступу: [www.uk.wikipedia.org/wiki/WhatsApp](http://www.uk.wikipedia.org/wiki/WhatsApp) (дата звернення 30.03.2019) – WhatsApp.
7. Вікіпедія [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Київ: НБУВ, 2013-2019. – Режим доступу: [www. uk.wikipedia.org/wiki/Facebook\_Messenger](http://www.ua.wikipedia.org/wiki/Messenger) (дата звернення 30.03.2019) – Messenger.
8. Сазерленд Дж. Scrum. Навчись робити вдвічі більше за менший час / Сазерленд Дж. – Київ: Книжковий Клуб "Клуб Сімейного Дозвілля", 2016, 280 c.
9. Еспозіто Д. Розробка Сучасних веб-застосунків: аналіз предметних областей і технологій / Еспозіто Д – Москва: «Діалектика-Вільямс», 2017, 464 с
10. Паван Вора. Шаблони проектування веб-додатків / Паван Вора [пер. Михайло Райтман] – Москва: «Ексмо», 2011