ChatGPT

Реалізація бота для вивчення англійської у WhatsApp: адаптація з Telegram

Вступ

Створення чат-бота у **WhatsApp** з логікою, аналогічною існуючому Telegram-боту, вимагає врахування відмінностей платформи. Поточний бот побудований на основі .NET 9 з використанням PostgreSQL, високопродуктивного кешу Dragonfly та патерну CQRS, розділяючи бізнес-логіку на **сцени** (наприклад, onboarding, lesson, idle). Він підтримує **квізи з варіантами відповіді**, зберігає прогрес користувачів, використовує уроки у форматі Markdown та відправляє мультимедійний контент. У цьому звіті проаналізовано, як перенести цю архітектуру до WhatsApp, які інструменти/API обрати, та які зміни необхідні з урахуванням особливостей WhatsApp. Ми порівняємо можливості Telegram vs WhatsApp (кнопки, форматування, мультимедіа, інтерактивність) і надамо рекомендації щодо адаптації сценової архітектури, обробки вводу користувача та врахування обмежень.

Вибір платформи/API для інтеграції WhatsApp

Для розробки бота у WhatsApp існує два основні підходи: - **Офіційний WhatsApp Business API (Cloud API)** від Meta. - **API-провайдери на зразок Twilio** (Twilio WhatsApp API) або інші партнери (360Dialog, etc).

WhatsApp Business Cloud API: Цей варіант надає прямий REST API доступ до WhatsApp. Ви отримуєте токен доступу і взаємодієте з endpoint-ами Meta. Переваги: повний контроль над функціональністю (у т.ч. інтерактивні повідомлення, медіа), відсутність додаткових націнок окрім офіційної плати за сесію/повідомлення. Недоліки: потрібно самостійно підтримувати вебхук-сервер для отримання повідомлень, керувати автентифікацією Facebook App і **дотримуватися політик WhatsApp** (наприклад, шаблони повідомлень для виходу за межі 24-годинного вікна, про що далі).

Twilio API для WhatsApp: Twilio виступає посередником, надаючи зручні SDK (зокрема для .NET) та єдиний інтерфейс для різних каналів. Twilio може спростити налаштування вебхуків і постачає інструменти (Content API, шаблони) для відправки повідомлень. Однак, варто врахувати дві речі: **вартість** та **обмеження функціоналу**. Twilio стягує додаткову плату ~$0.005 за повідомлення поверх офіційних тарифів WhatsApp [1]. Також, історично Twilio дозволяв кнопки лише через шаблони повідомлень (що потребують затвердження) або свій Content API. Наприклад, Twilio підтримує до 3 кнопок у звичайному (in-session) повідомленні [2] і до 10 через спеціальні шаблони, причому для більш ніж 3 кнопок може знадобитися попереднє затвердження контенту. Це дещо ускладнює динамічні квізи, оскільки кожне нове запитання з кнопками могло б вимагати шаблон.

Рекомендація: Для гнучкого навчального бота, де контент (уроки, питання) часто змінюється, **WhatsApp Cloud API** є доцільнішим. Він підтримує інтерактивні повідомлення **без необхідності шаблонів у межах 24-годинної сесії**, дозволяючи динамічно надсилати варіанти відповідей на квізи. Twilio варто розглядати лише якщо потрібна мультиплатформність або вже

використовується їхній стек. В іншому випадку, пряме підключення до WhatsApp API дасть більше контролю і менше додаткових витрат. В .NET можна реалізувати інтеграцію через прості HTTP-запити або використати сторонні SDK, що обгортають WhatsApp Cloud API.

Порівняння можливостей Telegram та WhatsApp

Щоб успішно перенести бот на WhatsApp, необхідно розуміти ключові відмінності між платформами:

Кнопки та інтерактивні елементи

Telegram: підтримує як *Reply-кнопки* (клавіатура, що підставляє текст у поле введення), так і *Inline-кнопки* під повідомленнями. Inline-кнопки можуть викликати callback-запити, які не відображаються як окремі повідомлення в чаті [3]. Це зручно для квізів – користувач натискає варіант, а бот отримує лише ідентифікатор кнопки і може відреагувати, наприклад, надіслати пояснення. Кількість кнопок на повідомлення в Telegram досить велика (можна вивести сітку кнопок).

WhatsApp: традиційно є **текстовим мессенджером** з обмеженою підтримкою кнопок. З появою Business API з'явились *інтерактивні повідомлення* двох видів: **Quick reply buttons** (до 3 штук) та **List messages** (список до 10 елементів) [4] [5]. Quick replies відображаються під повідомленням як кнопки; при натисканні WhatsApp надсилає бізнесу відповідь. Важливо: у WhatsApp **не існує прихованого callback** – коли користувач натискає кнопку, у чаті зазвичай з'являється повідомлення від користувача з текстом кнопки або вибраним елементом. Бот отримує цей вибір через webhook: зокрема, API передає або текст кнопки, або її `id` (ідентифікатор), якщо він заданий розробником [2] [6]. Таким чином, вибір користувача у квізі буде явно видимий у розмові (наприклад, "Відповідь: Option 1").

Наслідки для бота: Квізи з варіантами можливі, але потрібно врахувати ліміт **до 3 кнопок** на одне повідомлення (більше – через List, але списки доступні тільки на мобільних клієнтах). Якщо в Telegram квіз мав більше опцій, доведеться переглянути дизайн (наприклад, розбити на підпитання чи скоротити варіанти). Також бот повинен обробляти випадки, коли користувач не натиснув кнопку, а **ввів свою відповідь текстом** (таке більш імовірно в WhatsApp). Рекомендовано все ж використовувати інтерактивні **Quick reply-кнопки**, встановлюючи для них зрозумілі заголовки. Від Twilio або в самому WhatsApp API можна задати невидимий payload/ID кнопки [7], щоб бот однозначно ідентифікував вибір незалежно від тексту.

Форматування тексту (Markdown/HTML)

Telegram: підтримує форматування повідомлень через Markdown або HTML. Розробник може надсилати текст з позначками (*жирний*, *курсив*, гіперпосилання тощо) і задати parse_mode, тоді Telegram-клієнт відобразить відповідне форматування. Це дозволяло легко використовувати наявні Markdown-уроки – бот просто пересилав текст з Markdown і Telegram сам форматував.

WhatsApp: форматування набагато простіше. Немає підтримки повноцінного Markdown або HTML-тегів, але доступні **спеціальні символи для базового оформлення**. Зокрема, WhatsApp дозволяє: *жирний текст* за допомогою `*зірочок*`, *курсив* через `_нижнє підкреслення_`, ~перекреслений~ через `~тильди~`. Монопростір (код) задається обрамленням у `трійні бектики` або `одинарні бектики` для короткого фрагменту [8] [9]. Нещодавно WhatsApp також

додав підтримку **списків** та цитат: список формується, якщо рядки починаються з `-` чи `*` (маркер) або `1.` (нумерація), а цитата – якщо рядок починається з `>` [10] [11].

Наслідки для бота: Більшість текстового контенту уроків можна повторно використати. Markdown-форматування, що було в уроках, доведеться **конвертувати в сумісний з WhatsApp вигляд**. Хороша новина – синтаксис WhatsApp дуже схожий на Markdown: ті самі *зірочки* для жирного, *для курсиву* і т.д. Отже, якщо ContentService зберігає уроки у Markdown, їх можна використовувати майже без змін, просто відправляючи як звичайний текст. **Виключення**: елементи Markdown, які **WhatsApp не підтримує**, наприклад, явні гіперпосилання текстом. У WhatsApp при відправці такого синтаксису користувач побачить його як є, без форматування. Тому посилання слід подавати повним URL (клієнт WhatsApp автоматично його робить клікабельним) або, за потреби, використати **кнопку типу "call-to-action"** з URL (WhatsApp дозволяє до 2 кнопок з відкриттям посилання, але лише в шаблонних повідомленнях). В контексті навчального бота, простіше надавати посилання прямо текстом. Решта стилів (жирний, списки, код) буде відображена, якщо правильно розставити символи [8] [12]. ContentService можна доопрацювати, щоб **видаляти або трансформувати непідтримувані конструкції** (наприклад, прибрати квадратні дужки Markdown-лінків, залишивши URL).

Підтримка мультимедіа

Telegram: дуже гнучкий у надсиланні медіа. Підтримуються фото, відео, аудіо, документи, стікери, GIF; бот може відправляти голосові повідомлення чи відео-ноти. Розмір файлів – до 2 ГБ, тож практично без обмежень для навчального контенту.

WhatsApp: підтримує основні типи медіа – **зображення, аудіо, відео, документи, місцеположення, контакти**. Формати трохи обмеженіші (наприклад, відео лише MP4/3GP, аудіо – MP3/AAC/OGG). Файли також мають ліміт розміру: *WhatsApp Cloud API дозволяє до ~100 МБ* на файл [13], хоча деякі провайдери радять тримати відео до 16 МБ для надійності [14]. Це менше, ніж у Telegram, але все одно достатньо для більшості освітніх матеріалів (наприклад, короткі відео, аудіо вимови тощо).

Особливості надсилання: Telegram Bot API дозволяє просто надіслати файл multipart-запитом або за file_id. В WhatsApp відправка медіа відбувається через **посилання або попереднє завантаження на сервер Meta**. Наприклад, щоб надіслати зображення, треба або надати URL до нього (HTTPS-посилання на файл), або завантажити файл через медіа-ендпоінт і отримати media_id для подальшого використання. Це додає кроків: можливо, ContentService доведеться навчити **завантажувати/зберігати медіафайли** перед відправкою у WhatsApp. Якщо зображення уроків уже розміщені онлайн (на CDN чи сервері), можна використовувати URL-варіант.

Наслідки для бота: Бот цілком може надавати мультимедійний контент і у WhatsApp. Формат уроків з картинками, аудіо для вимови тощо – усе підтримується, просто через інші виклики API. Наприклад, в Telegram Markdown міг містити `![Alt](image.png)` – у WhatsApp цей блок слід замінити на відправку самого зображення окремим повідомленням. **Програвання аудіо:** користувачі WhatsApp зможуть отримувати аудіофайли (MP3/OGG) і прослуховувати їх у додатку. Голосові повідомлення (як формат) бот надіслати не може безпосередньо, але можна відправити аудіо з mime-type `opus/ogg` – воно фактично відобразиться як голосове (WhatsApp розпізнає опус-файл як voice message).

Інтерактивність та UX (меню, команди, реакції)

Меню та команди: У Telegram боти можуть мати *команди* (через символ `/`) і навіть кнопки-меню (бот-меню під полем вводу). Користувач може знайти бот через пошук, почати його командою `/start`, або обрати готову команду з меню. **WhatsApp не має концепції команд або меню** для ботів – взаємодія завжди ініціюється користувачем як звичайне повідомлення. Щоб почати спілкування, користувач повинен написати на номер бізнесу (або натиснути click-to-chat лінк). Отже, онбординг у WhatsApp, ймовірно, почнеться з того, що користувач надіслав будь-яке повідомлення ("Привіт" чи спеціальне слово). Бот має це розпізнати як сигнал запустити сцену **Onboarding** (аналогічно до /start). Можна вказувати в інструкціях, що новим користувачам треба надсилати певне слово, але зазвичай достатньо реагувати на перше звернення.

Callback і підтвердження дій: У Telegram після натискання inline-кнопки бот міг відредагувати повідомлення, сховати кнопки або надіслати нове з результатом. WhatsApp **не дозволяє редагувати уже надіслане повідомлення** – бот просто відправляє наступне. Наприклад, у квізі: на Telegram можна було в тому самому повідомленні показати правильну відповідь, замінивши текст після вибору. У WhatsApp доведеться надіслати окреме повідомлення: "**Правильно!** " або "**Неправильно**. Правильна відповідь: ...". Це впливає на UX: ланцюжок повідомлень може виростати, але для користувача WhatsApp це звична справа, оскільки так працюють всі чати.

Реакції на повідомлення: Обидві платформи дозволяють користувачу реагувати емодзі на повідомлення. Але на момент написання, **боти Telegram не отримують подій про реакції**. WhatsApp Business API, навпаки, *може сповіщати про реакції* – у вебхуку може прийти подія, якщо користувач поставив смайлик на повідомлення бізнесу [15]. В контексті навчального бота це не надто суттєво, але теоретично можна використовувати реакції як просту форму відповіді (наприклад, попросити реагувати 👍/👎). Однак це ускладнює сценарій, тому зосередимось на текстових відповідях.

Загальна взаємодія: WhatsApp надає **високий рівень довіри і охоплення**, але менш багатий інтерфейс для ботів. Telegram – відкритіший, публічніший, безкоштовний і функціонально гнучкий [16] [4]. В таблиці порівнянь зазначається, що Telegram – це "пісочниця" із творчою свободою (inline-кнопки, меню, групи тощо), тоді як WhatsApp – "шосе з правилами" (все через офіційний API, потрібні шаблони для певних повідомлень, є плата за обмін повідомленнями) [4] [17]. Для нашої задачі це означає, що **усі особливості треба ретельно спланувати під правила WhatsApp**: від частоти повідомлень до способів отримання відповіді.

Адаптація сценової архітектури під WhatsApp

Існуюча архітектура бота поділена на **сцени** – окремі логічні модулі, що відповідають за різні етапи взаємодії (онбординг нового користувача, проведення уроку/квізу, стан очікування тощо). Кожна сцена, імовірно, реалізована класом з методами `EnterAsync()` (виконується при вході в сцену, щоб надіслати початкове повідомлення/меню) та `HandleUpdateAsync()` (викликається при отриманні оновлення від користувача в межах цієї сцени). Такий підхід можна зберегти і для WhatsApp, проте потрібно адаптувати спосіб отримання та обробки оновлень, а також виклики API для відправки повідомлень.

Обробка вебхуків та універсальна модель оновлень

У Telegram бот, швидше за все, отримував **Update**-об'єкти через Long Polling або Webhook, використовуючи Telegram.Bot API. Ці `Update` містять різні типи подій – повідомлення, callback-

запити від кнопок, тощо. В WhatsApp ми також будемо отримувати події через **Webhook** (HTTP POST запити від серверів Meta або Twilio). Структура цих подій інша: вона містить ID бізнес-аккаунта, телефон користувача, тип повідомлення і контент. Наприклад, повідомлення з текстом матиме JSON поле `text.body`, вибір кнопки – поле `button.text` та `button.payload` або `interactive.button_reply.id` [6] [18].

Необхідно реалізувати шар інтеграції, який: - Приймає HTTP-запит від WhatsApp, **валідуючи** токен (щоб пересвідчитись, що це Meta, а не хтось інший). - Парсить JSON у внутрішню структуру, яку вже знає наш бот. Можна створити свій клас `BotUpdate` з полями: `UserId` (номер телефону або WhatsApp ID), `Text` (текст повідомлення, якщо є), `Payload` (ідентифікатор кнопки, якщо це натиснута кнопка), `Media` (якщо прийшов файл), `Type` (тип події: текст, кнопка, зображення, тощо). Цей об'єкт – аналог Telegram.Update, спрощений і уніфікований.

Далі, головний диспетчер бота визначає, **в якій сцені знаходиться користувач** (це можна зберігати в базі або в кеші Dragonfly за ключем користувача). Наприклад, у таблиці `Users` в PostgreSQL може бути поле `CurrentScene` або стан зберігається окремо у Dragonfly для швидкого доступу.

Знайшовши відповідну сцену, система викликає `HandleUpdateAsync()` тієї сцени, передаючи їй отриманий `BotUpdate`. Усередині методу буде описана логіка реакції на той чи інший ввід. **Наприклад:** якщо це сцена уроку і очікувалась відповідь на запитання, метод перевірить `Text` або `Payload` вхідного оновлення, звірить з правильним варіантом, збереже результат (використовуючи CQRS – надішле команду оновити прогрес у базі) і надішле користувачу наступне повідомлення (чи то пояснення, чи наступний блок уроку).

Архітектурно **можна використовувати ті самі класи сцен**, що й для Telegram, але із певними умовами компіляції або параметрами платформи. Реалістичніше – виділити спільну бізнес-логіку (питання, перевірка відповіді, зміна прогресу) і мати *адаптер* під платформу: - **PlatformService / MessengerClient**: інтерфейс, який містить операції надіслати повідомлення, надіслати кнопки, надіслати медіа. Для Telegram реалізація одна (через Telegram API), для WhatsApp – інша (через WhatsApp API). - **Scene** може викликати, скажімо, `await _client.SendText(userId, text, options)` де `options` – набір кнопок чи форматування. Всередині конкретна реалізація сформує правильний запит. В Telegram це буде `SendMessageAsync(chatId, text, replyMarkup)`, а в WhatsApp – виклик POST до `/messages` endpoint з JSON.

Це дозволить **максимально перевикористати код сцен**, змінюючи лише механіку інтеграції. В контексті .NET/CQRS, можна написати окремі CommandHandlers для відправки повідомлень, або сервіси типу `WhatsAppMessageSender`, які буде викликати Scene.

EnterAsync та навігація між сценами

Принцип залишиться тим самим: коли бот переводить користувача в іншу сцену, він: 1. Змінює стан користувача (в базі або кеші) на нову сцену. 2. Викликає `EnterAsync()` нової сцени.

Метод `EnterAsync()` має надіслати користувачу початкове повідомлення цієї сцени. Наприклад, **OnboardingScene.EnterAsync** може відправити привітання і запитати, чи готовий користувач почати урок (з двома кнопками "Так" і "Ні"). В Telegram це дві inline-кнопки з callback "YES_START"/"NO_LATER". У WhatsApp можна надіслати повідомлення: *"Вітаємо! Готові розпочати урок зараз?"* і додати **2 quick reply-кнопки**: "Так" (id: YES_START) і "Ні" (id: NO_LATER). Користувач побачить ці кнопки і натисне, припустимо, "Так". WhatsApp надішле нашому вебхуку

повідомлення з типом *interactive*, де буде `button_reply.id = "YES_START"` (або текст "Так", ідентифікатор – залежно від реалізації)[18]. Диспетчер побачить, що користувач у сцені Onboarding, викликає її `HandleUpdateAsync()`, котрий по цьому `id` зрозуміє вибір і переведе на **LessonScene**. Далі **LessonScene.EnterAsync** відправить, скажімо, перший урок/квіз.

Важливо, що **EnterAsync сцен** в WhatsApp може відправляти *кілька* повідомлень підряд (наприклад, текст + картинка + варіанти). Однак, слід бути обережним: якщо надіслати серію з 5-6 повідомлень одразу, користувач може отримати їх неструктуровано. Краще комбінувати, де можливо, в одне повідомлення (наприклад, використати повідомлення з **заголовком зображенням** і **тілом-текстом** плюс кнопки – WhatsApp дозволяє в інтерактивному повідомленні додати заголовок-картинку або документ [19]). Таким чином урок, що містить картинку і питання, може бути представленим однією інтерактивною карткою: картинка (заголовок), текст питання (тіло) і кнопки варіантів.

HandleUpdateAsync та обробка відповіді

У Telegram `HandleUpdateAsync` міг отримувати як `Message` (текст, фото тощо), так і `CallbackQuery` при натисканні кнопки. У WhatsApp ми фактично всі взаємодії отримуємо як **повідомлення від користувача**. Розрізняють лише їх тип: - **Текст**: звичайне текстове повідомлення. - **Натиснення кнопки (quick reply)**: теж приходить як повідомлення, але з певною структурою JSON (`type: interactive` + `button_reply` або `list_reply`). У спрощеній моделі ми можемо інтерпретувати це як повідомлення з заповненим `Payload` (ідентифікатор кнопки) і/або `Text` (текст кнопки). - **Медіа від користувача**: якщо користувач надіслав фото чи аудіо боту, це прийде як повідомлення типу image/audio із лінком для завантаження. В освітньому боті, можливо, захочемо приймати **голосові відповіді** (наприклад, вимова). Це можна реалізувати: WA API пришле нам voice повідомлення з посиланням, ми можемо передати його в бекенд для обробки (наприклад, розпізнавання мови) і дати фідбек. Але це виходить за межі базових вимог, тож за необхідності така обробка додається як окрема команда/функціональність.

Отже, `HandleUpdateAsync` в сценах WhatsApp повинен: - Аналізувати, чи є вхідний апдейт очікуваною відповіддю. Наприклад, якщо сцена чекала вибір варіанту на квіз, а прийшов `Payload` чи текст, що точно співпадає з одним із варіантів – опрацювати його. Якщо користувач замість кнопки написав щось зовсім інше, можна відправити ввічливе нагадування: *"Виберіть один з варіантів, будь ласка"* і знову показати кнопки. На Telegram таке менше потрібно (там користувачі звикли натискати кнопки), а в WhatsApp це корисно, бо деякі будуть відповідати своїми словами. - В разі валідної відповіді – виконати бізнес-логіку: зберегти результат у базі (наприклад, позначити, що на питання №3 відповів правильно, оновити прогрес уроку), підготувати наступний крок. - Перевести сцену, якщо потрібно. Наприклад, після завершення уроку – перейти в IdleScene або наступний LessonScene, викликавши `EnterAsync` нової сцени.

Приклад: У **LessonScene** на початку уроку `EnterAsync` надіслав текст уроку і запитання: "*Choose the correct translation:* ..." з кнопками A, B, C. `HandleUpdateAsync` отримує від користувача вибір. Припустимо, користувач натиснув кнопку A. Ми отримаємо або payload `"A"` або текст `"Option A"` – залежно як відправили. Код `HandleUpdateAsync` перевіряє: якщо правильна відповідь, шлемо "*Correct!* ", якщо ні – "*Incorrect.* Правильна відповідь: ...*". У Telegram ми могли б редагувати попереднє повідомлення, а тут надсилаємо нове. Далі, можна одразу надіслати наступне запитання уроку (якщо є) – знову з кнопками. Це все відбувається всередині `HandleUpdateAsync` до виходу. За потреби, між повідомленнями можна робити невеликі паузи (WhatsApp допускає швидке надсилання кількох повідомлень, але для кращого сприйняття можна витримати 1-2 секунди, аби користувач прочитав фідбек перед наступним питанням).

Суттєвий момент – **відсутність окремих callback-апдейтів** у WhatsApp означає, що **логіка обробки відповідей і повідомлень об'єднується**. Якщо у вас для Telegram були розділені методи (наприклад, одні на `OnCallbackQuery`, інші на `OnMessage`), то для WhatsApp це буде єдиний шлях. Можна структурувати код, щоб він спочатку перевіряв Payload (тобто натиснення кнопки), інакше брав текст. Фактично, payload кнопки – аналог callback data.

Обмеження та важливі відмінності, які слід врахувати

При проектуванні бота під WhatsApp необхідно зважати на декілька **обмежень платформи** і змін у підході:

- **24-годинне вікно сесії:** Основне **правило WhatsApp** – бізнес може вільно листуватися з користувачем лише протягом 24 годин після останнього повідомлення користувача [20]. Якщо користувач довго не відповідає (>24 год), бот **не може ініціювати повідомлення довільного змісту**. Дозволені тільки заздалегідь затверджені шаблонні повідомлення (Message Templates) поза цим вікном. Для навчального бота це означає, що сценарії щоденних нагадувань або автоматичної розсилки уроків потребують окремого плану. Можна:
- Попросити користувача, щоб він сам написав боту для отримання наступного уроку (тобто зробив **pull-механізм**).
- Або підготувати **шаблонні повідомлення** ("Ваш новий урок готовий! Відповідайте 'Старт', щоб почати.") і використовувати їх для пуш-нотифікацій. Шаблони треба затвердити у Facebook і вони **не можуть містити кнопок** відразу, лише текст і змінні. Кнопки можна додати лише в шаблонах особливого виду (тільки CTA або швидка відповідь "Reply to msg", але для нашого випадку, ймовірно, простіше надсилати текст).

При активній сесії (користувач взаємодіє щодня) це правило не заважає. Але варто **відстежувати час останньої активності** і при потребі або не відправляти несподіваних повідомлень, або використовувати шаблон. Також Twilio та інші провайдери можуть автоматично відмовити в відправці, якщо сесія закрита, тож код повинен це обробляти (наприклад, отримати помилку 360, що потрібен шаблон).

- **Ліміти швидкості та черги:** На старті WhatsApp-номер має обмеження на кількість користувачів, яким можна писати на день (наприклад, 1000 розмов на добу на першому рівні) та швидкість повідомлень в секунду. В навчальному боті, якщо аудиторія невелика, це не буде проблемою. Але якщо планується масштаб, треба моніторити показник **Throughput** і **Quality Rating** номера [21]. Telegram-боти не мають таких жорстких лімітів, і масштабуються легше (сервери Telegram самі витримують навантаження, а бот тільки приймає апдейти). На WhatsApp бот-сервер відповідальний за черги повідомлень. Використання **Dragonfly** як кешу може допомогти контролювати черги – наприклад, ставити завдання відправки повідомлень у фон (через CQRS-команди) і слідкувати, щоб не перевищувати rate limit.

- **Відсутність групового використання:** На відміну від Telegram, де бот міг бути доданий у груповий чат (хоча навчальні боти зазвичай приватні), **WhatsApp Business API не дозволяє приєднувати номер до груп**. Тобто бот тільки для індивідуального спілкування. Це спрощує стан – кожен чат це окремий користувач.

- **Доступність бота для користувачів:** Telegram-бот достатньо знайти по юзернейму або посилання. WhatsApp-бот – це фактично номер телефону. Щоб люди почали з ним вчитися, вони повинні зберегти номер, перейти за посиланням *wa.me*, або ініціювати через форму

на сайті. Це не архітектурне, але UX-обмеження – можливо, треба буде забезпечити інструкцію чи кнопку "Навчатися в WhatsApp" на вашому сервісі, яка відкриє чат. Також, для використання офіційного API, потрібно зареєструвати **WhatsApp Business Account**, прив'язати номер, пройти верифікацію Facebook Business Manager і отримати "зелений значок" (опціонально). Цей процес повільніший за миттєве створення Telegram-бота [4], але одноразовий.

- **Обмеження кількості кнопок та їх вигляду:** Як згадано, максимум 3 кнопки (quick replies) в повідомленні. Якщо урок передбачає більше варіантів, можливо, доведеться *розбити питання*. WhatsApp не підтримує **ієрархічних меню** або випадаючих списків, окрім List message, яка показує список до 10 елементів (але вона займає весь екран при відкритті і менш інтуїтивна для маленьких опцій). Тому бажано тримати варіанти 2-3, максимум 5 (де 5 краще подати як список). **Inline-кнопки зі швидким підтвердженням** (як "лайк/дизлайк") також нема – доведеться все робити через повідомлення.

- **Неможливість редагування/видалення повідомлень ботом:** У Telegram бот міг відредагувати або видалити своє повідомлення (наприклад, для оновлення контенту чи приховування кнопок після натискання). В WhatsApp API такої можливості немає – якщо контент змінюється, слід надіслати нове повідомлення. Тому **сценарії треба планувати як лінійну розмову**. Наприклад, після відповіді на питання квіза можна або нічого не робити (залишити кнопки як є, користувач вже натиснув і бачив свою відповідь), або відправити пояснення. Але прибрати старі кнопки неможливо. Це дрібниця, оскільки користувачі WhatsApp звикли до історії переписки.

- **Ідентифікація користувачів:** У Telegram кожен користувач має `chat_id` (найчастіше це і є user_id) та username (може бути відсутній). У WhatsApp – ідентифікатором слугує номер телефону в форматі міжнародного номера (або спеціальний `wa_id`). Бот автоматично отримує номер та ім'я профілю користувача при першому повідомленні [22]. Це треба врахувати при збереженні профілю: можна використовувати телефон як ключ, або зв'язувати з існуючим акаунтом, якщо у вас є свій облік користувачів. Також, якщо планується і Telegram-версія, й WhatsApp-версія бота, можливо, доведеться **зводити прогрес окремо** або об'єднувати через якусь прив'язку (наприклад, користувач надав номер телефону в Telegram для синхронізації). Але це вже рішення бізнес-логіки, не обов'язкове.

Повторне використання контенту та ContentService (Markdown-уроки)

Ви вже маєте систему зберігання і подачі уроків (імовірно, Markdown розмітка, яка потім транслюється ботом у повідомлення). **ContentService** – компонент, що постачає текст уроку, питання, варіанти відповідей, а також посилання на медіа для уроку. Його, безперечно, варто використати повторно. Основні зміни стосуватимуться *рендерінгу* контенту: - **Форматування:** Як обговорювалося, Markdown потрібно привести до виду, прийнятного WhatsApp. Це можна зробити або на льоту (в ContentService додати режим "WhatsApp" – замінювати розмітку на символи WhatsApp, видаляти непідтримуване), або зберігати уроки відразу в спрощеному форматі. Перший варіант більш гнучкий, тим паче, якщо планується підтримувати обидва канали. - **Розбиття на повідомлення:** Telegram дозволяв великий обсяг тексту і навіть медіа + підпис (caption) до 1024 символів під зображенням. У WhatsApp кожне повідомлення теж може містити до ~4096 символів, тож довгі тексти влізуть. Але практика показує, що краще ділити урок на менші частини, надсилати поступово. ContentService вже може вміти це (наприклад, відправляв урок

частинами з паузами). Для WhatsApp це слід робити ще ретельніше, щоб користувач не отримав стіну тексту. - **Медіа контент:** Якщо урок містить зображення чи аудіо, ContentService повинен надати або URL, або сам файл. В Telegram могло бути достатньо ID файлу (якщо повторно використовували раніше завантажений). В WhatsApp, як згадано, **потрібен URL або попереднє завантаження**. Можна організувати зберігання медіа на сервері (наприклад, Amazon S3 чи Azure Blob) і зберігати URL в ContentService. Тоді при надсиланні бот передасть цей URL у запиті, і WhatsApp сам завантажить файл з нього. Це спрощує систему (не треба проксувати файли через свій сервер кожен раз).

Висновок: Загалом **той самий ContentService можна використовувати**, якщо забезпечити мінімальні адаптації для формату WhatsApp. Рекомендується додати шар форматування: функцію, яка приймає Markdown-вміст і: - замінює або видаляє конструкції на кшталт `[текст посилання](URL)` (можна замінити на просто `текст посилання (URL)` або на сам URL), - гарантує, що списки позначені символами `-` або `1.` з пробілом, щоб WhatsApp відобразив їх як списки [12] [23], - додає потрібні символи для жирного/курсиву, якщо вони були в Markdown (Telegram Markdown міг інакше трактувати _ vs *). *Наприклад, Markdown-заголовок* `## Розділ 2` *– WhatsApp не зробить його жирним автоматично. Його можна або залишити як є (буде просто "## Розділ 2" текстом), або явно обгорнути в* зірочки *для жирності. Можна в ContentService прописати правило, що всі заголовки перетворюємо на верхній регістр чи обгортаємо в …*. Це на розсуд дизайну уроків.

Архітектура рішення для WhatsApp-бота

З урахуванням всього вищесказаного, можна запропонувати таку архітектурну схему адаптації:

- **Веб-сервіс (.NET 9)**: виступає бекендом бота. Він містить:
- **Контролер Webhook** (WhatsAppWebhookController), який приймає запити від WhatsApp. Цей контролер реалізує перевірку підпису і розбирає вхідні дані. На вхід отримує JSON від WhatsApp Cloud API (або від Twilio, якщо той виступає посередником). Контролер витягає звідти потрібну інформацію і формує об'єкт, наприклад, `BotUpdate`.
- **Сервіс маршрутизації/менеджер сцен** (наприклад, `SceneManager`). Він отримує `BotUpdate` і визначає, яку сцену наразі має користувач. Можна використати **Dragonfly** як швидкий key-value: ключ – user (WhatsApp phone or wa_id), значення – код сцени або even об'єкт сцени в пам'яті. Якщо запису немає, значить користувач новий – призначити сцену Onboarding.
- **Самі сцени** (OnboardingScene, LessonScene, IdleScene, і т.д.), як частину бізнес-логіки. Вони можуть бути реалізовані як об'єкти, що використовують сервіси:
    - **ContentService** (для отримання тексту уроків, питань, варіантів, медіа).
    - **UserProgressService/Repository** (для зчитування і запису прогресу в PostgreSQL).
    - **MessengerClient** (інтерфейс до платформи).
- **MessengerClient / WhatsAppApiService**: компонент, що знає, як відправляти повідомлення через WhatsApp API. Він формує HTTP-запити до потрібних endpoint-ів (наприклад, `messages`), додає токен авторизації і потрібні поля (номери телефонів, тексти, структуру інтерактивних повідомлень).

- **CQRS (MediatR)**: усередині сцени або менеджера можуть використовуватися команди і запити. Наприклад, Scene може зробити `await _mediator.Send(new SaveAnswerCommand(userId, lessonId, questionId, answer))` щоб зберегти відповідь в БД, або `_mediator.Send(new GetNextQuestionQuery(...))` щоб отримати наступне питання. Це добре ізолює логіку і використовує існуючі обробники. Так само,

відправка повідомлення може бути командою `SendMessageCommand` з параметрами, яку виконає окремий handler. Але можна й напряму викликати метод MessengerClient – залежить від того, наскільки все побудовано на CQRS.

- **База даних (PostgreSQL)**: зберігає стани користувачів (прогрес уроків, відповіді, можливо профіль). Стан сцени можна або теж зберігати в БД, або (як згадано) в кеші Dragonfly для швидкості. Dragonfly як Redis-кеш може зберігати не критичну інформацію типу `User:12345:CurrentScene = "LessonScene"` для пришвидшення маршрутизації, а при перезапуску бота стан можна відновити з БД, якщо також пишеться туди.

- **Інтеграція з Telegram**: якщо планується підтримувати обидві платформи одночасно, можна зробити **окремий контролер** для Telegram вебхука або процес отримання. Основна логіка сцен та сервісів може бути спільною. Просто буде дві реалізації MessengerClient: TelegramClient і WhatsAppClient, і SceneManager визначатиме, звідки апдейт, та підключати відповідного клієнта. Можна навіть розділити на два різних проекти/сервіси для спрощення – один обробляє Telegram, інший WhatsApp, але використовують одну БД. Це питання зручності розвитку.

Нижче наведено *короткий приклад* обробки запиту з кнопками у WhatsApp API, щоб проілюструвати відмінності від Telegram.

Приклад: Надсилання питання з варіантами у WhatsApp та обробка відповіді

Сценарій: Бот знаходиться у сцені уроку і задає користувачу питання з декількома варіантами відповіді. Припустимо, це запитання типу “Виберіть правильний переклад слова *apple*”.

Відправка питання з кнопками (бот → користувач): На Telegram це могло бути зроблено методом `SendTextMessage` з параметром `InlineKeyboardMarkup` (кнопки "Яблуко", "Апельсин", "Груша", наприклад). У WhatsApp потрібно виконати HTTP POST запит до API. Якщо використовувати **WhatsApp Cloud API** напряму, запит виглядатиме приблизно так:

```
POST https://graph.facebook.com/v17.0/<PHONE_NUMBER_ID>/messages
Content-Type: application/json
Authorization: Bearer <ACCESS_TOKEN>

{
  "messaging_product": "whatsapp",
  "to": "<USER_PHONE_ID>",
  "type": "interactive",
  "interactive": {
    "type": "button",
    "body": {
      "text": "Переклад слова *apple*?"
    },
    "action": {
      "buttons": [
        {
          "type": "reply",
```

```
          "reply": {
            "id": "OPTION1",
            "title": "яблуко"
          }
        },
        {
          "type": "reply",
          "reply": {
            "id": "OPTION2",
            "title": "апельсин"
          }
        },
        {
          "type": "reply",
          "reply": {
            "id": "OPTION3",
            "title": "груша"
          }
        }
      ]
    }
  }
}
```

Цей запит надішле користувачу повідомлення з текстом "*Переклад слова apple?*" і трьома кнопками: "яблуко", "апельсин", "груша". Зверніть увагу, кожна кнопка має `id` (наприклад, OPTION1) — це наш аналог callback-data, який не видно користувачу [7]. Користувач побачить лише заголовки.

Відповідь користувача на кнопку (користувач → бот): Якщо користувач натисне, скажімо, "яблуко", WhatsApp відобразить цю відповідь у чаті і надішле нашому вебхуку JSON приблизно такого виду:

```
{
  "entry": [
    {
      "changes": [
        {
          "value": {
            "contacts": [ { "wa_id": "USER_PHONE", ... } ],
            "messages": [
              {
                "from": "USER_PHONE",
                "timestamp": "1694035763",
                "type": "interactive",
                "interactive": {
                  "button_reply": {
                    "id": "OPTION1",
                    "title": "яблуко"
                  },
```

```
                "type": "button_reply"
              }
            }
          ]
        }
      }
    ]
  }
```

Найважливіша частина – всередині `messages[0].interactive.button_reply` ми бачимо `id: "OPTION1"` і `title: "яблуко"` [18]. Це означає, що користувач обрав варіант з ID = OPTION1. Наш Webhook-контролер дістане цю інформацію і створить `BotUpdate` зі значенням `Payload = "OPTION1"` (таким самим, що ми призначили). Далі **LessonScene.HandleUpdateAsync** отримує цей BotUpdate, розуміє, що OPTION1 відповідає варіанту "яблуко". Припустимо, це правильний переклад. Тоді бот може відправити назад повідомлення: " *Вірно! «Apple» – це яблуко.*". Якщо б користувач вибрав інший ID, бот відправив би " *Неправильно.*". Після цього бот переходить до наступного питання або завершує урок.

Як обробити текстову відповідь замість кнопки: Уявімо, користувач вирішив не натискати кнопку, а сам написав слово "яблуко". Наш webhook тоді отримав би `messages[0].text.body = "яблуко"`. Payload в цьому випадку відсутній. Ми б у `HandleUpdateAsync` побачили, що `Text = "яблуко"`, і могли зіставити це з очікуваними варіантами (наприклад, порівняти без регістру). Якщо співпадає з правильним варіантом – прийняти як правильну відповідь. Якщо це щось неочікуване (наприклад, користувач написав "не знаю") – можна надіслати уточнення або повторити питання з кнопками.

Таким чином, логіка квізу зберігається, хоча реалізація відрізняється. Користувач у WhatsApp отримує зрозумілі інструкції і може або натиснути кнопку, або відповісти текстом – бот буде готовий до обох випадків.

Висновки та рекомендації

Реалізація бота для вивчення англійської у WhatsApp є цілком здійсненною з повторним використанням більшої частини логіки Telegram-бота. Основні роботи полягають у **адаптації інтеграційного шару** (виклики WhatsApp API, прийом вебхуків) та в **UI-перетвореннях** (кнопки, форматування, потік повідомлень). Ключові рекомендації:

- **Обрати відповідний API-провайдер:** пряма інтеграція з WhatsApp Cloud API надасть гнучкість для динамічних уроків і квізів. Twilio може спростити деякі речі, але вводить шаблонізацію для інтерактивності та додаткові витрати [1], що небажано для освітнього бота з часто змінним контентом.
- **Зберегти сценову архітектуру:** сценарний підхід (Onboarding → Lesson → Idle та ін.) залишається ефективним. Реалізуйте універсальний обробник оновлень, який маршрутизуватиме вхідні повідомлення до потрібної сцени. Методи `EnterAsync` та `HandleUpdateAsync` можна використовувати з мінімальними змінами, якщо інкапсулювати платформні виклики через інтерфейси.
- **Інтерактивність через quick replies:** замініть Telegram inline-кнопки на WhatsApp quick reply buttons. Подавайте до 3 варіантів одночасно, використовуйте списки для більшої

кількості (або, краще, спрощуйте). Обробляйте як вибір кнопки, так і введення текстом – щоб користувач завжди міг відповісти.

- **Адаптуйте форматування уроків:** переконайтеся, що контент відображається коректно в WhatsApp. Використовуйте просте форматування (жирний, курсив, списки) що підтримується клієнтом [8] [12]. Уникайте функцій, яких нема (вбудовані посилання, кнопки в тексті тощо). Якщо уроки містять багато емодзі чи специфічних символів – перевірте, як вони рендеряться в WhatsApp (більшість емодзі сумісні, проблем бути не повинно).
- **Врахуйте політики WhatsApp:** сплануйте механізм відправки розсилок/нагадувань з урахуванням 24-годинного правила. Зарезервуйте і затвердіть шаблони, якщо потрібні автоматичні повідомлення (наприклад, *"Ми скучили за вами! Відправте 'Старт', щоб продовжити урок."* для повторного залучення користувача після простою).
- **Тестуйте UX на реальних пристроях:** те, як бачить користувач послідовність повідомлень у WhatsApp, дещо відрізняється від Telegram. Перевірте, щоб повідомлення надсилалися в правильному порядку, кнопки працювали очікувано, медіа вантажилися. Зверніть увагу на дрібниці: наприклад, WhatsApp не дозволяє відправити два інтерактивних повідомлення підряд без тексту між ними – щоб уникнути такого обмеження, чергуйте контент або об'єднуйте його.
- **Безпека та стабільність:** налаштуйте ретрай-логіку для вебхуків (WhatsApp може надсилати повторно, якщо не отримала 200 OK). Логуйтесь події, особливо помилки від API, щоб швидко реагувати (наприклад, якщо випадково перевищили ліміт або невалідний контент відправляєте).

У підсумку, матимемо двохканальну систему навчального бота, де Telegram-версія і WhatsApp-версія працюють паралельно, забезпечуючи максимально широкий охоплення користувачів з однаково продуманою логікою навчання, адаптованою під можливості кожної платформи. **WhatsApp-бот**, хоч і має більше правил, дозволить достукатися до аудиторії, яка віддає перевагу цьому месенджеру, забезпечивши при цьому високий рівень залучення (відкриття повідомлень ~98% [24]) та довіри до вашого сервісу.

---

[1] What does WhatsApp API pricing involve? A comprehensive ...
https://medium.com/@peppercloud/what-does-whatsapp-api-pricing-involve-a-comprehensive-breakdown-729c430f63e6

[2] [7] twilio/quick-reply | Twilio
https://www.twilio.com/docs/content/twilio-quick-reply

[3] Telegram Bot Features
https://core.telegram.org/bots/features

[4] [16] [17] [24] Telegram bots vs. WhatsApp bots: which is better for your business? - BAZU
https://bazucompany.com/blog/telegram-bots-vs-whatsapp-bots-which-is-better-for-your-business/

[5] [6] [18] [22] Webhook Events (Partner & Messaging API) | 360Dialog
https://docs.360dialog.com/partner/integrations-and-api-development/webhook-events-and-setup/webhook-events-partner-and-messaging-api

[8] [9] [10] [11] [12] [23] How to format your messages | WhatsApp Help Center
https://faq.whatsapp.com/539178204879377/?cms_platform=web

[13] Media - WhatsApp Cloud API - Meta for Developers
https://developers.facebook.com/docs/whatsapp/cloud-api/reference/media/

[14] Supported message types on WhatsApp Business API - Cloud API
https://help.sleekflow.io/en_US/supported-message-types-on-whatsapp-business-api-cloud-a

15 Reaction Message on WhatsApp - API Code - Vonage
https://developer.vonage.com/en/messages/code-snippets/whatsapp/send-reaction

19 Interactive Wa With Buttons - Start Your Cloud Communication
https://docs.msg91.com/whatsapp/interactive-whatsapp-buttons

20 WhatsApp Business Messaging Policy
https://business.whatsapp.com/policy

21 Messaging Limits - WhatsApp Business Platform - Meta for Developers
https://developers.facebook.com/docs/whatsapp/messaging-limits/