

Web-формы в чат-боте и Web Apps

- Введение. Виды форм. Принцип работы.
- Создание HTML формы.
- Универсальная форма в виде ссылки (для любого мессенджера)
- Web App форма с использованием inline-кнопки (только для Telegram)
- Web App форма с использованием keyboard-кнопки (только для Telegram)

Введение. Виды форм.

Принцип работы.

Теоретическая часть для разработчика сайта (разработчика HTML-формы) и для разработчика бота.

Веб-формы расширяют канал коммуникаций с ботом и позволяют в текстовом боте использовать все возможности HTML и JavaScript, таким образом бот, по функционалу становится полноценной заменой любому веб-сайту. Бот теперь не ограничивает пользователя в коммуникациях только обменом текстом, аудио или файлами.

Актуальный исходный код веб-формы реализующий все три вида форм смотрите по ссылке: go-to-the-mars.html

Пример работы веб-формы приведенной выше смотрите в Telegram боте https://t.me/metabot_test_form_bot

Разработка веб-формы, расширяющей диалоговую коммуникацию происходит в несколько шагов:

1. Перед началом разработки необходимо выбрать один из вариантов реализации формы и согласовать с разработчиком веб-формы и разработчиком чат-бота (если форма и чат-бот разрабатываются параллельно). Также необходимо согласовать формат обмена данными.
 1. [Универсальная в виде ссылки \(для любого мессенджера\)](#).
 2. [WebApp, на основе inline-кнопки \(только для Telegram\)](#).
 3. [WebApp, на основе keyboard-кнопки \(только для Telegram\)](#).
2. После того как вы определились с видом формы, необходимо [разработать HTML-форму](#), разместить его на вашем сервере, подключить ее к backend вашего сайта (для обработки AJAX / API запросов), чтобы backend мог принять данные с формы и отправить их в API в Metabot.

Для разработчика формы: реализация любого вида формы не имеет принципиальных архитектурных отличий, одна и также веб-форма может

использоваться для любого вида формы путем добавления не сложных разветвлений в JS-коде веб-формы. Отличие только в том, что вариант с keyboard-кнопкой не требует дополнительного слоя для backend, но при этом имеет ряд ограничений, подробнее см в описании этого вида формы ниже. Мы рекомендуем использовать вариант Web App с inline-кнопок + универсальную форму для любого мессенджера.

Для разработчика бота: см отдельную страницу документации для реализации необходимого вида формы в боте.

[Дополнительное изучение возможностей работы Telegram Web Apps.](#)

Универсальная форма в виде ссылки

Ключевая особенность: универсальный вариант, подходит для любого мессенджера.

Форма открывается в отдельной странице браузера. Как обычная страница web-браузера.

Для разработчика бота: вы можете добавить в боте условия, проверяя текущий мессенджер лида, чтобы открывать для нативного Telegram-канала форму с помощью inline кнопки, а для других каналов отправлять ссылку на форму в виде сообщения. Учтите, что в других мессенджерах нельзя удалять сообщения, поэтому заранее предусмотрите время жизни хэш-кода лида который указывается в ссылке на форму

Для Telegram ссылку можно отправить в виде inline-кнопки, но это все равно будет не "Web App приложение", а обычная страница открытая в браузере.

Если в мессенджере открытие ссылок во внешнем браузере выключено то форма откроется в браузере мессенджера, но все равно работа с данным видом формы будет отличаться от формы в виде Web App для inline/keyboard кнопки. При этом пользователю все равно доступна возможность позволяющая открыть ссылку во внешнем браузере. После закрытия формы нет гарантий, что пользователь вернется в браузер, поэтому желательно прислать пользователю уведомление в бота, чтобы он нажал на него и вернулся в бота.

Если вы планируете использовать формы только в Telegram, то можете пропустить данный вид формы и перейти к описанию формы на основе Web App, открывающейся с помощью inline-кнопки. Но, желательно понимать отличия и изучить весь раздел документации.

Принцип работы, по шагам:

1. Бот генерирует ссылку с уникальным хэш-кодом, привязанным к лиду.
2. Бот отправляет эту ссылку в мессенджер в виде обычного сообщения.

3. Пользователь бота нажимает на ссылку, открывается браузер с формой.
4. Пользователь бота заполняет форму и нажимает кнопку для отправки формы.
5. Форма отправляет данные на backend сайта, где размещена форма.
6. Backend сайта отправляет данные формы в API Metabot, в данные должен быть включен уникальный хэш лида.
7. Metabot принимает и сохраняет данные заполненной формы .
8. Если все ок, то страница с формой в браузере должна быть закрыта, это делается с помощью простого JS кода (подробности рассматриваются на отдельной странице документации с описанием создания HTML-формы).

Пример JS кода для открытия ссылки на Telegram бота и закрытия страницы

```
<script>
  function closeForm() {
    location.href="tg://resolve?domain=metabot_test_form_bot";
    window.close();
  }
</script>
```

После заполнения формы, данные введенные пользователем необходимо отправить на бэк вашего сайта, а затем в Metabot API используя токен бота (чтобы токен бота не фигурировал в коде HTML-формы). Далее страницу браузера необходимо автоматически закрыть, с помощью JS-кода встраиваемого в HTML-форму. Рекомендуется отправлять данные на бэк с помощью REST API, для того чтобы после отправки не выполнять дополнительный рендеринг страницы (чтобы потом ее просто закрыть). Т.е. отправляем данные на бэк, убеждаемся что ответ от API - 200, т.е. все ОК и сразу закрываем форму, пользователь возвращается в бота. При этом желательно учесть в коде HTML-формы варианты, если что-то пошло не так, или пользователь остался на форме, а хэш код лида для формы истек (если вы генерируете токен со сроком действия), в этих случаях можно вывести сообщение о том что форма устарела, что пользователю необходимо вернуться в бота и запросить форму повторно, также дополнительно можно уведомить Metabot по API, чтобы бот автоматически выслал ссылку на новую форму.

Есть нюанс: перед закрытием формы можно вызвать открытие ссылки для перехода в мессенджер (на случай если после закрытия страницы не будет открываться мессенджер). Но, для PC, например, открытие ссылки с Telegram ботом не срабатывает автоматически и у пользователя остается открытой другая вкладка в браузере (если у него включено открытие ссылок мессенджера во внешнем браузере и в браузере открыто несколько вкладок).

Ссылка может быть отправлена в бота в виде простого текста или inline-кнопки Telegram, содержащей ссылку на форму. Но в любом случае, для рассматриваемого варианта формы, нажатие на ссылку или кнопку приведет к открытию странице в браузере и после заполнения формы страницу необходимо автоматически закрывать. Здесь речь идет именно об обычной inline-кнопке с ссылкой для Telegram, а не о

Telegram WebApp-форме, на основе inline-кнопки которая открывает форму в Web App (во всплывающем окне, на котором расположен WebView). Описание вариантов форм с WebApp смотрите ниже.

Информация для разработчика бота:

Отличие между Inline Web App Button и обычной inline кнопки в виде ссылки заключается в том, что Inline Web App Button формируется передачей параметров `web_app` и `url` а для обычная inline-кнопка в виде ссылки просто передачей параметра `url`. Детали можно найти в описании API Telegram.

<https://core.telegram.org/bots/webapps#inline-button-web-apps>

<https://core.telegram.org/bots/api#inlinekeyboardbutton>

<https://core.telegram.org/bots/api#inlinekeyboardmarkup>

<https://core.telegram.org/bots/api#sendMessage>

Inline кнопка с ссылкой для Telegram реализуется на основе JS Callback команды бота и метода `bot.sendMessage` или с помощью плагина `Common.TelegramComponents.MenuHelper.sendMessage` с передачей дополнительных параметров для работы inline-кнопки как ссылки.

Отличие `bot.sendMessage` от плагина `Common.TelegramComponents.MenuHelper.sendMessage` в том что функция плагина сама внутри вызывает `bot.sendMessage`, и при этом автоматически сохраняет ID последних сообщений в атрибутах лида. Этот атрибут нужен для использования в коде JS Callback, а также маршруте бота, если требуется удаление кнопки-ссылки на форму или самого сообщения с ссылкой. Функционал хранения идентичен работе с Фото-слайдером для Telegram (описано в документации по работе с командой JS Callback и файлами Telegram).

Команда JS callback необходима для обработки нажатия других кнопок которые могут быть отправлены вместе с кнопкой-ссылкой на форму или для fallback, если форму не заполнили, но отправили любой текст в мессенджер.

Команда бота JS Callback не должна в данном случае обрабатывать событие отправки формы. Т.к. событие отлавливается с помощью Internal API Endpoint. В JS Callback можно отлавливать событие заполнения формы только для формы реализованной на основе keyboard-кнопки в Telegram.

WebApp-форма, на основе inline-кнопки

Данный вид формы работает только в Telegram.

Рекомендуемый вид формы для Telegram бота.

Ключевая особенность: форма открывается в всплывающем окне на котором расположен WebView (встроенный браузер).

Отличие данного варианта от универсальной формы в виде ссылки, в том, что форма работает как Web App, т.е. это специальный режим браузера встроенного в Telegram. Этот режим гарантирует возврат в бота после заполнения формы, а также не позволит открыть ссылку во внешнем браузере (а также включает дополнительный функционал для взаимодействия веб-страницы с ботом). В обычном же режиме для универсальной формы, нет гарантий, что когда веб-страница будет закрыта, пользователя перекинет в бота.

Принцип работы, по шагам:

1. Бот генерирует ссылку с уникальным хэш-кодом, привязанным к лиду.
2. Бот отправляет эту ссылку в виде inline-кнопки в Telegram-мессенджер.
3. Пользователь бота нажимает на кнопку, открывается Web App с формой (т.е. пользователь остается в Telegram, в мессенджере открывается встроенный браузер).
4. Пользователь бота заполняет форму и нажимает кнопку для отправки формы.
5. Форма отправляет данные на backend сайта, где размещена форма.
6. Backend сайта отправляет данные формы в API Metabot, в данные должен быть включен уникальный хэш лида.
7. Metabot принимает и сохраняет данные заполненной формы.
8. Если все ок, то страница с формой в браузере должна быть закрыта, это делается с помощью простого JS кода (подробности рассматриваются на отдельной странице документации с описанием создания HTML-формы).

WebApp-форма, на основе keyboard-кнопки

Данный вид формы работает только в Telegram.

Альтернативный вид формы для бота в Telegram. Используется, если для вас крайне затратно реализовать дополнительный backend-слой для отправки данных в Metabot API

Имеет ряд ограничений и неудобств (список смотрите ниже)

Ключевая особенность: форма открывается в всплывающем окне на котором расположен WebView (встроенный браузер). Для отправки данных в Metabot API не требуется дополнительный backend.

Для пользователя форма работает аналогично виду формы на основе inline-кнопки, но только кнопка для открытия формы расположена в нижней клавиатуре, в так называемой "кнопке-калькулятора".

Отличие данного варианта от универсальной формы в виде ссылки, в том, что форма работает как Web App, т.е. это специальный режим браузера встроенного в Telegram. Этот режим гарантирует возврат в бота после заполнения формы, а также не позволит открыть ссылку во внешнем браузере (а также включает дополнительный функционал для взаимодействия веб-страницы с ботом). В обычном же режиме для универсальной формы, нет гарантий, что когда веб-страница будет закрыта, пользователя перекинет в бота.

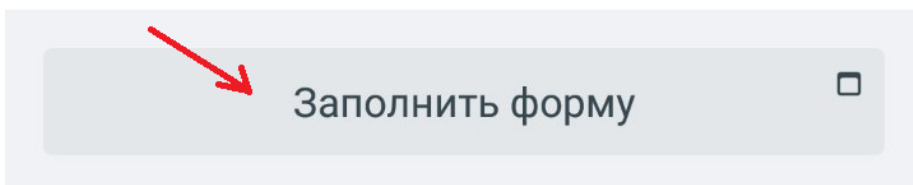
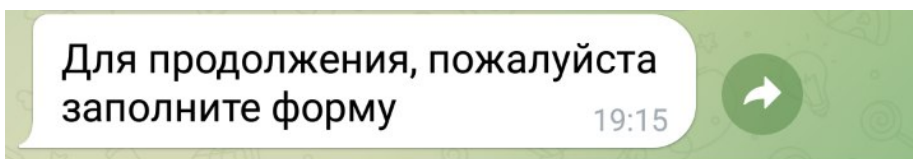
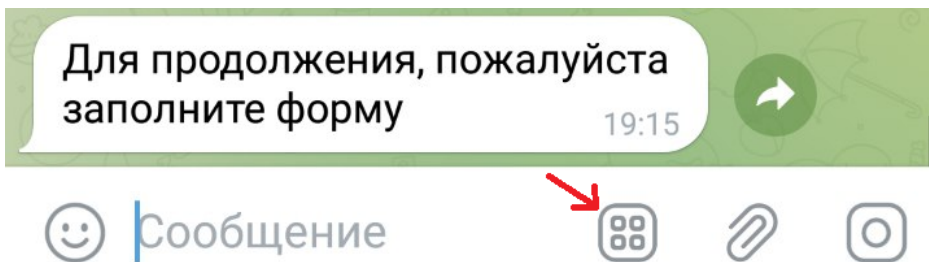
Форма открываемая с помощью keyboard-кнопки (так называемой "кнопки-калькулятора"), это упрощенный вариант, чтобы не подключать дополнительный back-end к HTML-форме для пересылки результата сбора данных с HTML-формы в Metabot API. На первый взгляд такой вариант более простой, но имеет ограничения, описанные ниже, поэтому **рекомендуется для Telegram использовать вариант WebApp-формы открываемой с помощью inline-кнопки.**

Принцип работы, по шагам:

1. Бот генерирует ссылку с уникальным хэш-кодом, привязанным к лиду.
2. Бот отправляет эту ссылку в виде keyboard-кнопки в Telegram-мессенджер ("кнопка-калькулятор" отображаемая в нижней части мессенджера).
3. Пользователь бота нажимает на кнопку, открывается Web App с формой (т.е. пользователь остается в Telegram, в мессенджере открывается встроенный браузер).
4. Пользователь бота заполняет форму и нажимает кнопку для отправки формы.
5. В коде HTML формы размещается JS код, для передачи введенных данных в телеграмм: `window.Telegram.WebApp.sendData(JSON.stringify(formData))`.
6. Для дополнительной защиты, желательно, в отправляемые данные включить уникальный хэш-код лида.
7. Metabot принимает данные заполненной формы от Telegram и сохраняет эти данные.
8. Страница с формой в браузере будет закрыта автоматически, после выполнения метода `Telegram.WebApp.sendData`, если же вы не используете `sendData`, то можно вызвать метод `Telegram.WebApp.close()`.

Ограничения и неудобства данного вида формы:

- Для такого вида формы: максимальный размер данных отправляемых с формы в бота составляет 4096 байт;
- Менее удобный UI/UX в виде кнопки отображаемой в нижней части браузера, также это меняет шаблон поведения пользователя, например, пользователь использовал inline-кнопки в боте, а теперь ему нужно дополнительно отслеживать кнопки "калькулятора";
- Кнопка может скрываться, если пользователь напишет что-то в чат-бот (возможен вариант повторного обновления кнопки, чтобы она не исчезала);
- В некоторых альтернативных мобильных приложениях для Telegram (например Plus Messenger для Android) список keyboard-кнопок может быть изначально схлопнут и необходимо нажимать в нижней части мессенджера на иконку для отображения кнопок меню (см скрины ниже).



Создание HTML формы.

Инструкция для разработчика веб-формы.

Актуальный исходный код веб-формы реализующий все три вида форм смотрите по ссылке: go-to-the-mars.html

Пример работы веб-формы приведенной выше смотрите в Telegram боте https://t.me/metabot_test_form_bot

Исходный код примера веб-формы

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <title>Заявка для полета на Марс</title>

  <!-- Подключаем Telegram Web App -->
  <script src="https://telegram.org/js/telegram-web-app.js"></script>

  <!-- Подключаем JQuery и JQ suggestions для dadata -->
  <!-- PS: JQuery подключать не обязательно, у вас могут быть свои библиотеки для работы с
  формой и отправки ajax запросов -->
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
    integrity="sha256-/xUj+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
    crossorigin="anonymous"></script>
  <script
    src="https://app.metabot24.com/lib/jquery.suggestions/js/jquery.suggestions.min.js"></script>
  <link href="https://app.metabot24.com/lib/jquery.suggestions/css/suggestions.min.css"
    rel="stylesheet"/>
```

```

<script language="JavaScript">
    let botId = ID_ВАШЕГО_БОТА
    let botToken = 'ТОКЕН_ВАШЕГО_БОТА'
    let dadataToken = 'ТОКЕН_DADATA'

    // URL для POST запроса на который будет отправлять данные с формы
    // В качестве примера мы отправляем напрямую в бота
    // Но в вашем production боте вы должны отправлять на данные на бэк
    // А с бэка уже пересылать в бота, чтобы, например "не светить" токен бота в коде
html-формы
    let sendUrl = '/api/v1/bots/' + botId + '/call/submit-form'

    let sendBody = {}
    let sendHeaders = {}
    let mode = null
    let tgWebApp = null

    $(function () {
        /* Определяем режим работы формы

            Режим передается в query url (GET параметр mode=)

            Режимы текущей html:
            - '' - пустая строка или null, когда в чат-боте ссылка на форму приходит в виде
ссылки
                это универсальный вариант который будет работать в любом мессенджере
            - 'tg_inline' - для Telegram чат-бота, когда ссылка на форму приходит в виде
inline-кнопки
            - 'tg_keyboard' - для Telegram чат-бота, когда ссылка на форму приходит в виде
keyboard-кнопки

            Для вашего бота может быть достаточно одного из режимов
            В качестве примера просто приведены 3 варианта, чтобы вы могли выбрать
подходящий и понять разницу
        */
        if (typeof (urlParams["mode"]) === 'string') {
            mode = urlParams["mode"]
        }

        // Переменная для доступа к Telegram Web App, чтобы не писать везде

```

```

window.Telegram.WebApp

if (mode === 'tg_inline' || mode === 'tg_keyboard') {
  if (window.Telegram && window.Telegram.WebApp) {
    tgWebApp = window.Telegram.WebApp
  }
}

// Инициализация Dadata для автокомплита поля с адресом
$(".dadata-suggestion").suggestions({
  token: dadataToken,
  type: "ADDRESS"
})

// Получаем хэш-код лида из request url (GET параметр q=)
$('#q').val(urlParams["q"])

// Событие нажатия на кнопку "Отправить данные"
$(document).on('click', '#submit-mars-form', (e) => {
  let form = $('#form-mars')
  if (!form[0].checkValidity()) {
    form[0].reportValidity()
    return
  }

  let formData = getFormData(form)

  formData['tg_query_id'] = ''
  formData['mode'] = mode

  if (mode === 'tg_inline') {
    if (tgWebApp && tgWebApp.initDataUnsafe && tgWebApp.initDataUnsafe.query_id) {
      formData['tg_query_id'] = tgWebApp.initDataUnsafe.query_id
    }
  }

  if (tgWebApp) {
    // https://core.telegram.org/bots/webapps#initializing-web-apps
    tgWebApp.expand() // необязательно
    tgWebApp.ready() // необязательно
  }
}

```

```

}

// Для универсального режима или режима tg_inline
if (mode !== 'tg_keyboard') {

    // Отправляем данные внутри script_request_params и указываем токен и др
заголовки для Metabot API

    // Но в вашем production, здесь вы должны просто отправить данные на ваш бэк, а
с бэка уже в Metabot API

    // отправлять необязательно через JSON API, можно делать это просто через ACTION
формы SUBMIT кнопку на форме

    //

    // Если вы реализуете универсальный режим и выполняете отправку через action
форма(сабмит без REST API),

    // то ваша форма разывается на два шага
    // - заполнение формы клиентом
    // - получаем данные на бэке
    //      и после приема данных рендерим опять HTML в коде которого размещаем
JavaScript который закроет страницу (вызовет метод closeForm())

    // Поэтому проще данные на бэк отправить по REST API и сразу же закрыть форму
    // Именно такой вариант и реализован в данной HTML форме
    sendBody = {"script_request_params": formData} //form.serializeArray()
    sendHeaders = {
        "Authorization": "Bearer " + botToken,
        'Content-Type': 'application/json',
        'Accept': 'application/json',
    }

    // Отправка запроса с помощью библиотеки JQuery
    sendJQueryRequest('POST', sendUrl, sendBody, sendHeaders, function (response,
isError, jqXHR, textStatus, errorThrown) {
        if (!isError) {
            // Запрос завершён. Здесь можно обрабатывать результат.
            //console.log(response)

            closeForm()
        } else {
            // Произошла ошибка
            alert("Ошибка обработки API запроса")
        }
    })
}

```

```

        console.log(jqXHR)
    }
})

// Отправка запроса без дополнительных библиотек (с помощью XHR)
// Могут быть проблемы с отправкой, возможно нужна корректировка кода под ваш
бэкенд вашего сайта

/*sendXhrRequest('POST', sendUrl, sendBody, sendHeaders, function(xhr) {

//https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest/send#%D0%BF%D1%80%D0%B8%D0%
BC%D0%B5%D1%80_get
    if(xhr.readyState == XMLHttpRequest.DONE && xhr.status == 200) {
        // Запрос завершён. Здесь можно обрабатывать результат.
        console.log(xhr)
        closeForm()
    } else {
        // Произошла ошибка
        alert("Ошибка обработки API запроса")
        console.log(xhr)
    }
})*/*
} else {
    // Если это режим tg_keyboard

    // Внимание! Лимит строки для sendData - 4096 байт !
    // Поэтому такой режим менее универсален, хотя на первый взгляд проще и не
требует бэкенда
    // Но в будущем могут возникнуть проблемы, если форма будет усложнена

    // Если необходимо вывести сообщение (например для отладки),
    // тк обычные alert и console.log для telegram Web App не работают
    //tgWebApp.showAlert('Все ок!')

    tgWebApp.sendData(JSON.stringify(formData));

    // Если не выполняем sendData, то закрываем форму сами
    //closeForm()
}

```

```

    })
  })

  /**
   * Метод для закрытия формы
   */
  function closeForm() {
    if (mode === null || mode === '') {
      location.href = "tg://resolve?domain=metabot_test_form_bot"
      window.close()
    } else {
      tgWebApp.close()
    }
  }
}

```

// <https://stackoverflow.com/questions/11338774/serialize-form-data-to-json>

```

function getFormData($form) {
  var unindexed_array = $form.serializeArray();
  var indexed_array = {};

  $.map(unindexed_array, function (n, i) {
    indexed_array[n['name']] = n['value'];
  });

  return indexed_array;
}

```

//<https://stackoverflow.com/questions/901115/how-can-i-get-query-string-values-in-javascript>

```

var urlParams = (function (a) {
  if (a == "") return {}
  var b = {}
  for (var i = 0; i < a.length; ++i) {
    var p = a[i].split('=', 2)
    if (p.length == 1)
      b[p[0]] = ""
    else
      b[p[0]] = decodeURIComponent(p[1].replace(/\+/g, " "))
  }
}

```

```

    return b
})(window.location.search.substr(1).split('&'))

// Отправка POST запроса по API с помощью JQuery
// https://reqbin.com/code/javascript/wzp2hxwh/javascript-post-request-example
function sendJQueryRequest(method, url, data, jsonHeaders, callback) {
    $.ajax({
        url: url,
        type: method,
        contentType: 'application/json; charset=utf-8',
        dataType: 'json',
        async: false,

        headers: jsonHeaders,
        data: JSON.stringify(data),

        success: function (response) {
            callback(response, false)
        },
        error: function (jqXHR, textStatus, errorThrown) {
            //alert("Произошла ошибка при обработке API запроса")
            callback(null, true, jqXHR, textStatus, errorThrown)
        }
    })
}

// Отправка POST запроса по API с помощью XHR
// https://reqbin.com/code/javascript/wzp2hxwh/javascript-post-request-example
function sendXhrRequest(method, url, data, jsonHeaders, callback) {
    let xhr = new XMLHttpRequest();
    xhr.open(method, url);

    xhr.setRequestHeader("Accept", "application/json")
    xhr.setRequestHeader("Content-Type", "application/json")

    if (typeof (jsonHeaders) != "undefined") {
        for (let key in jsonHeaders) {
            xhr.setRequestHeader(key, jsonHeaders[key])
        }
    }
}

```

```
}

xhr.onload = () => console.log(xhr.responseText)

xhr.onload = function () {
  // Запрос завершён. Здесь можно обрабатывать результат.
  callback(xhr)
};

//Вызывает функцию при смене состояния.
//xhr.onreadystatechange = function() {
//  callback(xhr)
//}

xhr.send(JSON.stringify(data))
}
</script>
```

```
<!-- Стили формы, в вашей релизации будет свой блок кода, подключаемый в виде css файла
-->
```

```
<style>
body {
  background-color: #070619;
  /*width: 100%;
  height: 100%; */
  font-size: 18px;
}
```

```
.main-container {
  width: 95%;
  height: 95%;
}
```

```
form {
  color: #fff;
  border: 1px solid silver;
  border-radius: 10px;
  padding: 10px;
```



```
margin: 10px auto 0 auto;
width: 95%;
max-width: 500px;
height: 100%;
}
```

```
form .form-title {
  font-size: 20px;
  text-align: center;
  font-weight: bold;
}
```

```
form .form-group {
  margin: 10px;
}
```

```
form .form-group input {
  font-size: 18px;
}
```

```
form .form-group input[ type=checkbox] {
  width: 20px;
  height: 20px;
}
```

```
form .form-group input.dadata-suggestion {
  max-width: 75%;
}
```

```
form .form-group select {
  font-size: 18px;
}
```

```
.suggestions-suggestions {
  color: #000
}
```

```
form .form-button {
  text-align: center;
```

```

        margin-bottom: 10px;
    }

    form .send-button {
        font-size: 18px;
        border: 1px solid #fff;
        border-radius: 3px;
        background-color: #fff;
        padding: 5px;
        font-weight: bold;
        text-decoration: none;
    }

    form .form-button button {
        font-size: 18px;
        font-weight: bold;
    }

    .mars {
        position: absolute;
        left: calc(55vw);
        /*right: 0;*/
        top: 0;
        z-index: -10;
        opacity: 0.8;
    }
</style>
</head>
<body>

<!-- HTML КОД формы -->

<div class="main-container">
    <!-- Для отправки с помощью submit укажите атрибут action данной формы-->
    <form id="form-mars" autocomplete="off">
        <div class="form-title">Заявка для полета на Марс</div>

        <input type="hidden" id="q" name="q" value="">

```

```
<div class="form-group">
  <label for="name">
    Ваше имя:
  </label>
  <div>
    <input type="text" name="name" id="name" placeholder="Юрий Гагарин" required
autofocus>
  </div>
</div>
```

```
<div class="form-group">
  <label for="email">
    Почта:
  </label>
  <div>
    <input type="email" name="email" id="email" placeholder="yuri@gagarin.ru">
  </div>
</div>
```

```
<div class="form-group">
  <label for="age">
    Возраст:
  </label>
  <div>
    <input type="number" name="age" id="age" min=12 max=777 step=1>
  </div>
</div>
```

```
<div class="form-group">
  <label for="specialization">
    Профессия:
  </label>
  <div>
    <select name="specialization" id="specialization" required>
      <option value="engineer" selected>Инженер</option>
      <option value="scientist">Учёный</option>
      <option value="psychologist">Психолог</option>
      <option value="other">Другая</option>
    </select>
  </div>
</div>
```

```

    </div>
</div>

<div class="form-group">
    <label for="address">
        Ваш адрес проживания:
    </label>
    <input class="dadata-suggestion" type="text" name="address" id="address"
placeholder="" required>
</div>

<div class="form-group">
    <label for="is_qualified">
        Прошел курсы в Центре<br>подготовки космонавтов
        <input type="checkbox" name="is_qualified" id="is_qualified" value="1">
    </label>
</div>

<div class="form-group">
    <label for="has_experience">
        Я уже летал в космос (имею опыт)
        <input type="checkbox" name="has_experience" id="has_experience" value="1">
    </label>
</div>

<!-- Если нужна отправка фото -->
<!--<div class="form-group">
    <label>
        Фото:
        <input type="file" accept="image/jpeg" name="photo" required>
    </label>
</div>-->

<div class="form-button">
    <!--<button type="submit">Отправить заявку</button>--> <!-- Для отправки с помощью
submit формы -->
    <a class="send-button" id="submit-mars-form" href="#">Отправить заявку</a>
</div>
</form>

```

```

</div>
</body>
</html>
```

При использовании данного примера замените в следующих строках данные на актуальные для вашего бота:

- let botId = ID_ВАШЕГО_БОТА;
- let botToken = 'TOKEN_ВАШЕГО_БОТА';
- let dadataToken = 'TOKEN_DADATA'.

И измените код отправки заполненных данных, чтобы данные сначала отправлялись на ваш бэк, а затем с бэка в Metabot API. Также уберите отправку данных внутри параметра `script_request_params`, чтобы данные к вам на бэк отправлялись без этого параметра, но когда отправляете в Metabot API учитывайте эту особенность.

Речь идет про строку:

```
sendBody = {"script_request_params": formData}
```

Которую для вас нужно заменить на:

```
sendBody = formData
```

Также для понимания принципа работы изучите комментарии в HTML-коде.

Код не требует большого уровня владения HTML или JS для создания подобной формы, но заметим, что важной частью является UI/UX и может потребоваться профессиональный Frontend-разработчик, также необходимо учесть отработку всех исключительных ситуаций, например если произошел сбой API, о чем было упомянуто в теоретической части, во введении, заметим что выше приведен просто пример, все аспекты не проработаны досконально, тк это возможно только на конкретном рабочем примере для вашего бота.

Универсальная форма в виде ссылки (для любого мессенджера)

Инструкция для разработчика бота.

Актуальный исходный код веб-формы реализующий все три вида форм смотрите по ссылке: [go-to-the-mars.html](https://t.me/metabot_test_form_bot)

Пример работы веб-формы приведенной выше смотрите в Telegram боте https://t.me/metabot_test_form_bot

Форма открывается в отдельной странице браузера. Как обычная страница web-браузера.

Для Telegram ссылку можно отправить в виде inline-кнопки, но это все равно будет не "Web App приложение", а обычная страница открытая в браузере.

Если в мессенджере открытие ссылок во внешнем браузере выключено то форма откроется в браузере мессенджера, но все равно работа с данным видом формы будет отличаться от формы в виде Web App для inline/keyboard кнопки. При этом пользователю все равно доступна возможность позволяющая открыть ссылку во внешнем браузере. После закрытия формы нет гарантий, что пользователь вернется в браузер (например для PC), поэтому желательно прислать пользователю уведомление в бота, чтобы он нажал на него и вернулся в бота.

Если вы планируете использовать формы только в Telegram, то можете перейти к разделу описывающему форму на основе Web App, открывающейся с помощью inline-кнопки.

Краткое описание принципа работы бота:

1. В бот отправляется ссылка на форму (или inline-кнопка в виде ссылки). При нажатии на ссылку (или inline-кнопку в виде ссылки) открывается внешний браузер (или браузер встроенный в Telegram).

2. После заполнения формы данные должны быть отправлены на сторонний бэк, а из бэка данные должны быть отправлены в Metabot API. Для понимания смотрите исходный код веб-формы.
3. Данные пришедшие с формы сохраняются в JS Internal API Endpoint, который сохраняет данные и вызывает скрипт бота для продолжения беседы, этот скрипт работает с сохраненными данными формы, запрашивает подтверждения, что все введено верно, а также предлагает заполнить форму повторно.

Принцип работы, по шагам:

1. Бот генерирует ссылку с уникальным хэш-кодом, привязанным к лиду.
2. Бот отправляет эту ссылку в мессенджер в виде обычного сообщения.
3. Пользователь бота нажимает на ссылку, открывается браузер с формой.
4. Пользователь бота заполняет форму и нажимает кнопку для отправки формы.
5. Форма отправляет данные на backend сайта, где размещена форма.
6. Backend сайта отправляет данные формы в API Metabot, в данные должен быть включен уникальный хэш лида.
7. Metabot принимает и сохраняет данные заполненной формы.
8. Если все ок, то страница с формой в браузере должна быть закрыта, это делается с помощью простого JS кода (подробности рассматриваются на отдельной странице документации с описанием создания HTML-формы).

Таблицы для работы с формами

Чтобы рассматриваемый пример бота с формами работал, вам необходимо создать две кастомные таблицы.

Таблица: Хэш-коды лидов

Таблица необходима для хранения соответствий между лидом и хэшем для лида.

При каждом вызове формы выполняется поиск хэша по лиду, если хэш не найден, то формируется новый. После отправки формы, запись с хэш-кодом удаляется, чтобы ссылка на каждый новый опрос содержала уникальный хэш. Поиск предыдущего хэша необходим, если есть формы, ожидающие заполнения, для случаев, если лид закрыл форму, а затем спустя время решил ее заполнить, нажав на ту же кнопку вызова формы. При этом в таблицу заложено поле (`expire_at`) - дата истечения хэш-кода, вы можете использовать это поле для реализации времени жизни хэш-кода, но в таком случае в веб-форме необходимо предусмотреть уведомление о том, что форма истекла, или закрывать ее автоматом после истечения, или формировать в боте новую ссылку, если лид пытается отправить форму, хэш которой истек. Это требование не обязательно, может реализовываться по мере необходимости и зависит от конкретной задачи.

Структура таблицы:

| ID | АКТИВНА | СИХР. СТРУКТУРЫ | ОТОБРАЖАТЬ В МЕНЮ | НАИМЕНОВАНИЕ | ЗАГОЛОВOK В ИНТЕРФЕЙСЕ |
|----|---------|-----------------|-------------------|----------------|------------------------|
| 69 | Да | Да | Да | lead_form_hash | Leads Hashes for Forms |

Структура таблицы (JSON словарь):

| АКТИВНО | СИХР. СТРУКТУРЫ | НАИМЕНОВАНИЕ | ЗАГОЛОВOK | ТИП | ОБЯЗАТЕЛЬНОЕ | ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ | РАЗМЕР | ТОЧНОСТЬ | ПОДСКАЗКА | ВСПЛЫВАЮЩАЯ ПОДСКАЗКА |
|---------|-----------------|--------------|------------|---------------|--------------|-----------------------|--------|----------|-----------|-----------------------|
| Да | Да | id | ID | AUTOINCREMENT | Да | NULL | | | | |
| Да | Да | form | Form | TEXT | Да | | | | | |
| Да | Да | lead_id | Lead ID | BIG_INT | Да | NULL | | | | |
| Да | Да | hash | Hash | TEXT | Да | | | | | |
| Да | Да | expire_at | Expire At | DATETIME | | NULL | | | | |
| Да | Да | created_at | Created At | CREATED_AT | Да | NOW | | | | |

Таблица: Данные форм

Таблица необходима для хранения данных заполненных форм.

Данная таблица реализована как универсальная, все данные пришедшие с формы сохраняются в текстовом поле в виде JSON, поэтому она может работать с любой формой, но работать с строкой в виде JSON в JS неудобно, тк для доступа к данным приходится выполнять `JSON.parse()`, вы можете нормализовать данные, реализуя для каждой формы бота отдельную таблицу с необходимыми полями для хранения, например такими как: Имя, Фамилия, Адрес и т.д., чтобы в дальнейшем было проще работать с данными и выполнять поиск по таблице. Также, в вашем случае таблица для хранения данных может и не потребоваться, если всплывающая форма очень простая и предназначена, например, для заполнения одного поля, например, адреса с авто комплитом от Dadata или для выбора выпадающего списка или даты в календаре. Т.е. форма может реализовывать даже один компонент для заполнения которого необходим HTML + JS.

Структура таблицы:

| ID | АКТИВНА | СИХР. СТРУКТУРЫ | ОТОБРАЖАТЬ В МЕНЮ | НАИМЕНОВАНИЕ | ЗАГОЛОВOK В ИНТЕРФЕЙСЕ |
|----|---------|-----------------|-------------------|--------------|------------------------|
| 70 | Да | Да | Да | form_results | Form Results |

Структура таблицы (JSON словарь):

| активно | схр. структуры | наименование | заголовок | тип | обязательное | значение по умолчанию | размер | точность | подсказка | всплывающая подсказка |
|---------|----------------|--------------|------------|---------------|--------------|-----------------------|--------|----------|-----------|-----------------------|
| Да | Да | id | ID | AUTOINCREMENT | Да | NULL | | | | |
| Да | Да | form | Form | TEXT | Да | | | | | |
| Да | Да | lead_id | Lead ID | BIG_INT | Да | NULL | | | | |
| Да | Да | data | Data | TEXT | | | | | | |
| Да | Да | created_at | Created At | CREATED_AT | Да | NOW | | | | |
| Да | Да | updated_at | Updated At | UPDATED_AT | | NULL | | | | |

Чтобы посмотреть [исходный код](#) внутреннего API, сначала перейдите в бота по ссылке на любой из скриптов.

Нажмите в предупреждающем сообщении на ссылку для смены активного бота, а затем перейдите на страницу с внутренним API в настройках бота или [по ссылке](#).

Если у вас нет доступа к указанному скрипту, то запросите шаблон бота в технической поддержке Metabot.

Скрипт для формирования и отправки ссылки в мессенджер

Пример скрипта:

ID36182 Форма в виде ссылки

Тип: Стандарт Раздел: (корень)

КОМАНДЫ

Эти команды бот выполняет до того как покажет меню.

| КОМАНДА | СОДЕРЖИМОЕ |
|----------------------|--|
| Выполнить JavaScript | <pre>// СКРИПТ ФОРМИРОВАНИЯ ХЭШ-КОДА И ОТПРАВКИ ССЫЛКИ НА ФОРМУ // PS: Здесь же можно отправить ссылку в виде inline кнопки (не inline Web App, а обычную кнопку с ссылкой) // реализовав это в виде команды JS Callback // но такой вариант будет работать только в Telegram // поэтому чтобы данный вариант формы был универсальным отправляем ее просто в виде ссылки</pre> |
| Отправить текст | <p>Перейдите по ссылке ниже и заполните форму</p> <p>{{ &\$formUrl }}</p> |
| Выполнить скрипт | <p>Главное меню</p> |

Скрипт отправки ссылки состоит из следующих команд бота:

- **Выполнить JavaScript:**

```
let md5 = require('Common.Utls.Md5')

let expireAt = new Date()
expireAt.setSeconds(expireAt.getSeconds() + 300)

// Todo: Можно вынести в снippet
let hashItems = table.find('lead_form_hash', [], [
  ['form', '=', 'mars'],
  ['lead_id', '=', leadId],
])

let leadHash
if (hashItems.length > 0) {
  leadHash = hashItems[0].hash
} else {
  const salt = 'YourSuperSecretString' + (new Date()).getTime()
  leadHash = md5(salt + leadId)

  table.createItem('lead_form_hash', {
    'form': 'mars',
    'lead_id': leadId,
    'hash': leadHash,
    'expireAt': expireAt
  })
}

// Вы можете вынести эту ссылку в атрибуты бота
let url = 'https://app.metabot24.com/testWidgets/go-to-the-mars.html?q=' + leadHash

memory.setAttr('formUrl', url)
```

PS: Здесь же можно отправить ссылку в виде inline кнопки (не inline Web App, а обычную кнопку с ссылкой)
реализовав это в виде команды JS Callback
но такой вариант будет работать только в Telegram
поэтому чтобы данный вариант формы был универсальным отправляем ее просто в виде ссылки

- Команда **Отправить текст:**

Перейдите по ссылке ниже и заполните форму

```
{{ &$formUrl }}
```

- Команда **Выполнить скрипт** — команда необходима для вывода меню.

Внутреннее API для приема данных с формы и сохранения их в таблицу

Это точка API, куда приходит запрос после заполнения формы.

ВНУТРЕННЕЕ API. КОНЕЧНЫЕ ТОЧКИ

| ID | АКТИВНА | НАИМЕНОВАНИЕ | АЛИАС | МЕТОД | ФОРМАТ |
|----|---------|--------------|-------------|-------|--------|
| 94 | Да | submit-form | submit-form | POST | JSON |

JavaScript для вычисления API ответа (Response Body):

```
let requestParams = request.json
let formLeadId = null

if (!request.json || !request.json.q) {
  return {"result": false, "message": "Invalid Lead Hash"}
}

// Todo: Можно вынести в снippet
let hashItems = table.find('lead_form_hash', [], [
  ['form', '=', 'mars'],
  ['hash', '=', request.json.q],
])

let found = 0

if (hashItems.length > 0) {
  formLeadId = hashItems[0].lead_id
}
```

```

hashItems.forEach(hashItem => hashItem.delete())
}

if (formLeadId > 0) {
  // Удаляем предыдущие ответы
  // Todo: Можно вынести в снippet
  oldResults = table.find('form_results', [], [
    ['form', '=', 'mars'],
    ['lead_id', '=', formLeadId],
  ])
  if (oldResults.length > 0) {
    oldResults.forEach(oldResult => oldResult.delete())
  }

  // Предварительно сохраняем данные в таблице
  table.createItem('form_results', {
    "form": "mars",
    "lead_id": formLeadId,
    "data": request.string
  })

  bot.scheduleScriptByCode('after_submit', formLeadId)

  //Для передачи данных в скрипт без предварительного сохранения в таблицу
  //bot.scheduleScriptByCode('after_submit', leadId, null, {"script_request_params":
requestParams})
  return {"result": true}
} else {
  return {"result": false, "message": "Lead by hash is not found"}
}

```

Скрипт выполняемый после вызова API

Данный скрипт нужен чтобы пользователь продолжил диалог с ботом, а также демонстрирует как использовать данные сохраненные в таблицу результатов и вывести их в виде текста.

[Пример скрипта.](#)

Скрипт отправки ссылки состоит из следующих команд бота и пунктов меню:

- Команда **Выполнить JavaScript**:

```
const menuHelper = require('Common.TelegramComponents.MenuHelper')

// -----
// Удаляем кнопку с формой для Inline формы, если она выведена

if (menuHelper.hasLastTelegramMessageId()) {
    // Удаляем кнопку с ссылкой на форму
    menuHelper.removeInlineKeyboard()

    // Сбрасываем ID последнего сообщения
    menuHelper.clearLastTelegramMessageId()
}
// -----

// -----
// Для Inline кнопки с формой
// Удаляем кнопку с формой , если она выведена
// Удаляем здесь - тк после заполнения формы в JS-Callback мы уже не попадаем
// Тк прием данных с формы осуществляет через Internal Endpoint
// Для keyboard кнопки ссылку на форму с кнопкой здесь удалять не нужно,
// тк для keyboard кнопки мы возвращаемся в JS-Callback после заполнения формы
if (menuHelper.hasLastTelegramMessageId()) {
    // Удаляем кнопку
    menuHelper.removeInlineKeyboard()

    // Сбрасываем ID последнего сообщения
    menuHelper.clearLastTelegramMessageId()
}
// -----

// Если получаем данные напрямую, без таблицы form_results
//let data = request.json

// Если получаем данные из таблицы form_results
// Todo: Можно вынести в сниппет
let data = table.find('form_results', [], [
    ['form', '=', 'mars'],
    ['lead_id', '=', leadId]
```

```

])

if (data.length > 0 && typeof(data[0].data) === 'string' && data[0].data.length > 0) {
    data = JSON.parse(data[0].data)
} else {
    data = {}
    bot.sendText('Ошибка! Данные не найдены')
}

// Для определения какую кнопку показывать в меню ниже
// См условие аунктов меню текущего скрипта
memory.setAttr('mode', '')
if (typeof(data.mode) === 'string' && data.mode.length > 0) {
    memory.setAttr('mode', data.mode)
}

/*
// Сообщение о результате отработки формы, отправляется в бота
// Но проблема что система распознает сообщение как вебхук от лида
// Поэтому использовать можно только в JS-Callback и игнорировать вебхук вручную
// Или через регулярку маршрута, запуская отдельны скрипт
//
// Данное сообщение также автоматом закрывает web-view
// Но пробелм с закрытием нет, это выполняется в JS итак с помощью
// кода window.Telegram.WebApp.sendData(JSON.stringify(formData));
// или при выполнении window.Telegram.WebApp.close()
//
// Обычно используется для игры в web_view для вывода очков после победы или проигрыша
if (data.tg_query_id && data.tg_query_id.length > 0) {
    //https://core.telegram.org/bots/api#sentwebappmessage
    //https://core.telegram.org/bots/api#inlinequeryresultarticle
    bot.sendPayload('answerWebAppQuery', {
        "web_app_query_id": data.tg_query_id,
        "result": {
            "type": "article",
            "id": "1", // Unique identifier for this result, 1-64 Bytes
            "title": "Форма заполнена",
            // "description": "Description",
            "input_message_content": {"message_text": "Спасибо! Данные сохранены!"}
        }
    })
}

```

```
    }  
  })  
}  
*/  
  
memory.setJsonAttr('data', data)  
memory.setAttr('is_qualified', data.is_qualified ? 'Да' : 'Нет')  
memory.setAttr('has_experience', data.has_experience ? 'Да' : 'Нет')
```

- Команда **Отправить текст:**

Ваши данные:

Имя: {{ &\$\$data.name }}

Email: {{ &\$\$data.email }}

Возраст: {{ &\$\$data.age }}

Профессия: {{ &\$\$data.specialization }}

Ваш адрес проживания: {{ &\$\$data.address }}

Прошел курсы в Центре подготовки космонавтов: {{ &\$\$is_qualified }}

Я уже летал в космос (имею опыт): {{ &\$\$has_experience }}

Все данные в виде JSON:

{{ &\$\$data }}

- Команда **Отправить текст:**

Все верно?

- Пункты меню данного скрипта, с условиями:

| код | надпись | скрипт | условие вывода |
|-----|-------------------------|------------------------------|--|
| 1 | Да, все верно! | Подтвердил сохранение | |
| 2 | Нет, заполнить повторно | Форма в виде ссылки | <pre>let mode = memory.getAttr('mode') if (mode !== 'tg_inline' && mode !== 'tg_keyboard') { return true }</pre> |
| 2 | Нет, заполнить повторно | Форма в виде inline кнопки | <pre>let mode = memory.getAttr('mode') if (mode === 'tg_inline') { return true }</pre> |
| 3 | Нет, заполнить повторно | Форма в виде keyboard кнопки | <pre>let mode = memory.getAttr('mode') if (mode === 'tg_keyboard') { return true }</pre> |

Условия кнопок необходимы для вызова скрипта соответствующего текущему формату формы.

Условие для вывода универсальной формы в виде ссылки:

```
let mode = memory.getAttr('mode')
if (mode === 'tg_inline') {
  return true
}
```

Условие для вывода формы в Web App при нажатии на inline-кнопку:

```
let mode = memory.getAttr('mode')
if (mode !== 'tg_inline' && mode !== 'tg_keyboard') {
  return true
}
```

Условие для вывода формы в Web App при нажатии на keyboard-кнопку:

```
let mode = memory.getAttr('mode')
if (mode === 'tg_keyboard') {
  return true
}
```


Web App форма с использованием inline-кнопки (только для Telegram)

Инструкция для разработчика бота.

Данный вид формы работает только в Telegram.

Рекомендуемый вид формы для Telegram бота.

Актуальный исходный код веб-формы реализующий все три вида форм смотрите по ссылке: [go-to-the-mars.html](https://t.me/metabot_test_form_bot)

Пример работы веб-формы приведенной выше смотрите в Telegram боте https://t.me/metabot_test_form_bot

Отличие данного варианта от универсальной формы в виде ссылки, в том, что форма работает как Web App, т.е. это специальный режим браузера встроенного в Telegram. Этот режим гарантирует возврат в бота после заполнения формы, а также не позволит открыть ссылку во внешнем браузере (а также включает дополнительный функционал для взаимодействия веб-страницы с ботом). В обычном же режиме для универсальной формы, нет гарантий, что когда веб-страница будет закрыта, пользователя перекинет в бота.

Краткое описание принципа работы бота:

1. В бот отправляется inline-кнопка (в формате Telegram Web App). При нажатии на кнопку открывается всплывающее окно, в которое встроено Web View, пользователь не может открыть эту ссылку во внешнем браузере, а также просмотреть исходный

код формы (если только не откроет сам Telegram в браузере).

2. После заполнения формы данные должны быть отправлены на сторонний бэк, а из бэка данные должны быть отправлены в Metabot API. Для понимания смотрите исходный код веб-формы.
3. Данные пришедшие с формы сохраняются в JS Internal API Endpoint, который сохраняет данные и вызывает скрипт бота для продолжения беседы, этот скрипт работает с сохраненными данными формы, запрашивает подтверждения, что все введено верно, а также предлагает заполнить форму повторно.

Принцип работы, по шагам:

1. Бот генерирует ссылку с уникальным хэш-кодом, привязанным к лиду.
2. Бот отправляет эту ссылку в виде inline-кнопки в Telegram-мессенджер.
3. Пользователь бота нажимает на кнопку, открывается Web App с формой (т.е. пользователь остается в Telegram, в мессенджере открывается встроенный браузер).
4. Пользователь бота заполняет форму и нажимает кнопку для отправки формы.
5. Форма отправляет данные на backend сайта, где размещена форма.
6. Backend сайта отправляет данные формы в API Metabot, в данные должен быть включен уникальный хэш лида.
7. Metabot принимает и сохраняет данные заполненной формы.
8. Если все ок, то страница с формой в браузере должна быть закрыта, это делается с помощью простого JS кода (подробности рассматриваются на отдельной странице документации с описанием создания HTML-формы).

Таблицы для работы с формами

Чтобы рассматриваемый пример бота с формами работал, вам необходимо создать две кастомные таблицы.

Таблица: Хэш-коды лидов

Таблица необходима для хранения соответствий между лидом и хэшем для лида.

При каждом вызове формы выполняется поиск хэша по лиду, если хэш не найден, то формируется новый. После отправки формы, запись с хэш-кодом удаляется, чтобы ссылка на каждый новый опрос содержала уникальный хэш. Поиск предыдущего хэша необходим, если есть формы, ожидающие заполнения, для случаев, если лид закрыл форму, а затем спустя время решил ее заполнить, нажав на ту же кнопку вызова формы. При этом в таблицу заложено поле (`expire_at`) - дата истечения хэш-кода, вы можете использовать это поле для реализации времени жизни хэш-кода, но в таком случае в веб-форме необходимо предусмотреть уведомление о том, что форма истекла, или закрывать ее автоматом после истечения, или формировать в боте новую ссылку, если лид пытается отправить форму, хэш которой истек. Это требование не обязательно, может реализовываться по мере

необходимости и зависит от конкретной задачи.

Структура таблицы:

| ID | АКТИВНА | СИХР. СТРУКТУРЫ | ОТОБРАЖАТЬ В МЕНЮ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК В ИНТЕРФЕЙСЕ |
|----|---------|-----------------|-------------------|----------------|------------------------|
| 69 | Да | Да | Да | lead_form_hash | Leads Hashes for Forms |

Структура таблицы (JSON словарь):

| АКТИВНО | СИХР. СТРУКТУРЫ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК | ТИП | ОБЯЗАТЕЛЬНОЕ | ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ | РАЗМЕР | ТОЧНОСТЬ | ПОДСКАЗКА | ВСПЛЫВАЮЩАЯ ПОДСКАЗКА |
|---------|-----------------|--------------|------------|---------------|--------------|-----------------------|--------|----------|-----------|-----------------------|
| Да | Да | id | ID | AUTOINCREMENT | Да | NULL | | | | |
| Да | Да | form | Form | TEXT | Да | | | | | |
| Да | Да | lead_id | Lead ID | BIG_INT | Да | NULL | | | | |
| Да | Да | hash | Hash | TEXT | Да | | | | | |
| Да | Да | expire_at | Expire At | DATETIME | | NULL | | | | |
| Да | Да | created_at | Created At | CREATED_AT | Да | NOW | | | | |

Таблица: Данные форм

Таблица необходима для хранения данных заполненных форм.

Данная таблица реализована как универсальная, все данные пришедшие с формы сохраняются в текстовом поле в виде JSON, поэтому она может работать с любой формой, но работать с строкой в виде JSON в JS неудобно, тк для доступа к данным приходится выполнять `JSON.parse()`, вы можете нормализовать данные, реализуя для каждой формы бота отдельную таблицу с необходимыми полями для хранения, например такими как: Имя, Фамилия, Адрес и т.д., чтобы в дальнейшем было проще работать с данными и выполнять поиск по таблице. Также, в вашем случае таблица для хранения данных может и не потребоваться, если всплывающая форма очень простая и предназначена, например, для заполнения одного поля, например, адреса с авто комплитом от Dadata или для выбора выпадающего списка или даты в календаре. Т.е. форма может реализовывать даже один компонент для заполнения которого необходим HTML + JS.

Структура таблицы:

| ID | АКТИВНА | СИХР. СТРУКТУРЫ | ОТОБРАЖАТЬ В МЕНЮ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК В ИНТЕРФЕЙСЕ |
|----|---------|-----------------|-------------------|--------------|------------------------|
| 70 | Да | Да | Да | form_results | Form Results |

Структура таблицы (JSON словарь):

| АКТИВНО | СИХР. СТРУКТУРЫ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК | ТИП | ОБЯЗАТЕЛЬНОЕ | ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ | РАЗМЕР | ТОЧНОСТЬ | ПОДСКАЗКА | ВСПЛЫВАЮЩАЯ ПОДСКАЗКА |
|---------|-----------------|--------------|------------|---------------|--------------|-----------------------|--------|----------|-----------|-----------------------|
| Да | Да | id | ID | AUTOINCREMENT | Да | NULL | | | | |
| Да | Да | form | Form | TEXT | Да | | | | | |
| Да | Да | lead_id | Lead ID | BIG_INT | Да | NULL | | | | |
| Да | Да | data | Data | TEXT | | | | | | |
| Да | Да | created_at | Created At | CREATED_AT | Да | NOW | | | | |
| Да | Да | updated_at | Updated At | UPDATED_AT | | NULL | | | | |

Чтобы посмотреть [исходный код](#) внутреннего API, сначала перейдите в бота по ссылке на любой из скриптов.

Нажмите в предупреждающем сообщении на ссылку для смены активного бота, а затем перейдите на страницу с внутренним API в настройках бота или [по ссылке](#).

Если у вас нет доступа к указанному скрипту, то запросите шаблон бота в технической поддержке Metabot.

Скрипт для формирования ссылки и отправки кнопки в мессенджер

[Пример скрипта](#).

Скрипт отправки ссылки состоит из единственной команды бота:

- **Выполнить JavaScript Callback:**

КОМАНДЫ

Эти команды бот выполняет до того как покажет меню.

| КОМАНДА | СОДЕРЖИМОЕ | ОПЕРАЦИИ |
|-------------------------------|---|---|
| Выполнить JavaScript Callback | <div>Запустить Callback сразу при старте этой команды: Да</div> <div>JavaScript Callback:</div> <div>// Код скрипта</div> | <div></div> <div></div> |

```
const menuHelper = require(' Common. TelegramComponents. MenuHelper' )
let md5 = require(' Common. Utils. Md5' )
```

```
//
```

```
-----
```

```
// Инициализация компонента
```

```
if (menuHelper.isFirstImmediateCall()) {
```

```
    // !!! ИНИЦИАЛИЗАЦИЯ !!!
```

```
    // Скрываем предыдущее меню с кнопкой формы
```

```
    menuHelper.removeInlineKeyboard()
```

```
    // Сбрасываем ID последнего сообщения
```

```
    menuHelper.clearLastTelegramMessageId()
```

```
// -----
```

```
//ГЕНЕРИРУМ ССЫЛКУ С ХЭШЕМ
```

```
let expireAt = new Date()
```

```
expireAt.setSeconds(expireAt.getSeconds() + 300)
```

```
let hashItems = table.find(' lead_form_hash', [], [
    ['form', '=', 'mars'],
    ['lead_id', '=', leadId],
])
```

```
// Todo: Можно вынести в снippet
```

```
let leadHash
```

```
if (hashItems.length > 0) {
```

```

    leadHash = hashItems[0].hash
  } else {
    const salt = 'YourSuperSecretString' + (new Date()).getTime()
    leadHash = md5(salt + leadId)

    table.createItem('lead_form_hash', {
      'form': 'mars',
      'lead_id': leadId,
      'hash': leadHash,
      'expireAt': expireAt
    })
  }

  // Вы можете вынести эту ссылку в атрибуты бота
  let formUrl = 'https://app.metabot24.com/testWidgets/go-to-the-mars.html?mode=tg_inline&q='
+ leadHash
  // -----

  // Отправляем кнопку с формой при инициализации
  let message = 'Для продолжения, пожалуйста заполните форму'
  let buttons = [
    [
      /*{
        "text": 'Отмена',
        "callback_data": "btn_static_cancel",
      },*/
      {
        "text": 'Заполнить форму',
        //"callback_data": "btn_static_web_app",
        "web_app": {
          "url": formUrl
        },
      },
    ]
  ]

  let apiAdditionalParams = {
    "reply_markup": {
      "inline_keyboard": buttons
    }
  }
}

```

```

menuHelper.sendMessage(message, null, null, apiAdditionalParams)

return false
}
//
-----

//
-----

// ЛОГИКА КОМПОНЕНТА
//
-----

//
-----

// Обработка нажатых кнопок или входящего текста

if (["btn_static_cancel", "0"].includes(bot.getIncomingMessage().toLowerCase())) {
    // Скрываем предыдущее меню с кнопкой формы
    menuHelper.removeInlineKeyboard()

    // Или удаляем предыдущее сообщение, если необходимо
    // Не забываем выполнять аналогичное удаление при выходе из компонента с помощью маршрута
    //this.deletePrevMessage()

    // Или останавливаем анимацию ожидания на кнопке
    //menuHelper.answerCallbackQuery()

    // Выходим из команды
    return {
        "break": true,
        //"run_script_by_code": "код_запускаемого_скрипта",
    }
} else {
    // Если получаем любое другое текстовое сообщение
    if (bot.getIncomingMessage() !== "") {

```

```

bot.sendText(' Пожалуйста заполните форму' )

return false

// Чтобы вопрос не уходил вверх, тогда после удаления необходимо повторить вопрос
//this.deletePrevMessage()

// Повторяем вопрос, если удаляем пред. вопрос строкой выше с помощью deletePrevMessage
// ...
} //else if(1 === 2) {
    // Данные отправленные из webview обрабатываем не здесь, а в Internal API Endpint
    //return true
//}
}
//
-----
-----

```

Внутреннее api для приема данных с формы и сохранения их в таблицу

Это точка API, куда приходит запрос после заполнения формы.

ВНУТРЕННЕЕ API. КОНЕЧНЫЕ ТОЧКИ

| ID | АКТИВНА | НАИМЕНОВАНИЕ | АЛИАС | МЕТОД | ФОРМАТ |
|----|---------|--------------|-------------|-------|--------|
| 94 | Да | submit-form | submit-form | POST | JSON |

JavaScript для вычисления API ответа (Response Body):

```

let requestParams = request.json
let formLeadId = null

if (!request.json || !request.json.q) {

```



```

    return {"result": false, "message": "Invalid Lead Hash"}
}

// Todo: Можно вынести в снippet
let hashItems = table.find('lead_form_hash', [], [
    ['form', '=', 'mars'],
    ['hash', '=', request.json.q],
])

let found = 0

if (hashItems.length > 0) {
    formLeadId = hashItems[0].lead_id
    hashItems.forEach(hashItem => hashItem.delete())
}

if (formLeadId > 0) {
    // Удаляем предыдущие ответы
    // Todo: Можно вынести в снippet
    oldResults = table.find('form_results', [], [
        ['form', '=', 'mars'],
        ['lead_id', '=', formLeadId],
    ])
    if (oldResults.length > 0) {
        oldResults.forEach(oldResult => oldResult.delete())
    }

    // Предварительно сохраняем данные в таблице
    table.createItem('form_results', {
        "form": "mars",
        "lead_id": formLeadId,
        "data": request.string
    })

    bot.scheduleScriptByCode('after_submit', formLeadId)

    //Для передачи данных в скрипт без предварительного сохранения в таблицу
    //bot.scheduleScriptByCode('after_submit', leadId, null, {"script_request_params":
requestParams})
    return {"result": true}
}

```

```
} else {  
    return {"result": false, "message": "Lead by hash is not found"}  
}
```

Скрипт выполняемый после вызова API

Данный скрипт нужен чтобы пользователь продолжил диалог с ботом, а также демонстрирует как использовать данные сохраненные в таблицу результатов и вывести их в виде текста.

Пример скрипта.

Скрипт отправки ссылки состоит из следующих команд бота и пунктов меню:

- Команда **Выполнить JavaScript**:

```
const menuHelper = require('Common.TelegramComponents.MenuHelper')  
  
// -----  
// Удаляем кнопку с формой для Inline формы, если она выведена  
  
if (menuHelper.hasLastTelegramMessageId()) {  
    // Удаляем кнопку с ссылкой на форму  
    menuHelper.removeInlineKeyboard()  
  
    // Сбрасываем ID последнего сообщения  
    menuHelper.clearLastTelegramMessageId()  
}  
// -----  
  
// -----  
// Для Inline кнопки с формой  
// Удаляем кнопку с формой , если она выведена  
// Удаляем здесь - тк после заполнения формы в JS-Callback мы уже не попадаем  
// Тк прием данных с формы осуществляет через Internal Endpoint  
// Для keyboard кнопки ссылку на форму с кнопкой здесь удалять не нужно,  
// тк для keyboard кнопки мы возвращаемся в JS-Callback после заполнения формы  
if (menuHelper.hasLastTelegramMessageId()) {  
    // Удаляем кнопку  
    menuHelper.removeInlineKeyboard()
```

```

// Сбрасываем ID последнего сообщения
menuHelper.clearLastTelegramMessageId()
}
// -----

// Если получаем данные напрямую, без таблицы form_results
//let data = request.json

// Если получаем данные из таблицы form_results
// Todo: Можно вынести в снippet
let data = table.find('form_results', [], [
  ['form', '=', 'mars'],
  ['lead_id', '=', leadId]
])

if (data.length > 0 && typeof(data[0].data) === 'string' && data[0].data.length > 0) {
  data = JSON.parse(data[0].data)
} else {
  data = {}
  bot.sendText('Ошибка! Данные не найдены')
}

// Для определения какую кнопку показывать в меню ниже
// См условие аунктов меню текущего скрипта
memory.setAttr('mode', '')
if (typeof(data.mode) === 'string' && data.mode.length > 0) {
  memory.setAttr('mode', data.mode)
}

/*
// Сообщение о результате отработки формы, отправляется в бота
// Но проблема что система распознает сообщение как вебхук от лида
// Поэтому использовать можно только в JS-Callback и игнорировать вебхук вручную
// Или через регулярку маршрута, запуская отдельны скрипт
//
// Данное сообщение также автоматом закрывает web-view
// Но пробелм с закрытием нет, это выполняется в JS итак с помощью
// кода window.Telegram.WebApp.sendData(JSON.stringify(formData));

```

```
// или при выполнении window.Telegram.WebApp.close()
//
// Обычно используется для игры в web_view для вывода очков после победы или проигрыша
if (data.tg_query_id && data.tg_query_id.length > 0) {
    //https://core.telegram.org/bots/api#sentwebappmessage
    //https://core.telegram.org/bots/api#inlinequeryresultarticle
    bot.sendPayload('answerWebAppQuery', {
        "web_app_query_id": data.tg_query_id,
        "result": {
            "type": "article",
            "id": "1", // Unique identifier for this result, 1-64 Bytes
            "title": "Форма заполнена",
            // "description": "Description",
            "input_message_content": {"message_text": "Спасибо! Данные сохранены!"}
        }
    })
}
*/

memory.setJsonAttr('data', data)
memory.setAttr('is_qualified', data.is_qualified ? 'Да' : 'Нет')
memory.setAttr('has_experience', data.has_experience ? 'Да' : 'Нет')
```

- Команда **Отправить текст:**

Ваши данные:

Имя: {{ &\$\$data.name }}

Email: {{ &\$\$data.email }}

Возраст: {{ &\$\$data.age }}

Профессия: {{ &\$\$data.specialization }}

Ваш адрес проживания: {{ &\$\$data.address }}

Прошел курсы в Центре подготовки космонавтов: {{ &\$is_qualified }}

Я уже летал в космос (имею опыт): {{ &\$has_experience }}

Все данные в виде JSON:

{{ &\$\$data }}

- Команда **Отправить текст:**

Все верно?

- Пункты меню данного скрипта, с условиями:

| код | надпись | скрипт | условие вывода |
|-----|-------------------------|--|--|
| 1 | Да, все верно! | Подтвердил сохранение | |
| 2 | Нет, заполнить повторно | Форма в виде ссылки | <pre>let mode = memory.getAttr('mode') if (mode !== 'tg_inline' && mode !== 'tg_keyboard') { return true }</pre> |
| 2 | Нет, заполнить повторно | Форма в виде inline кнопки | <pre>let mode = memory.getAttr('mode') if (mode === 'tg_inline') { return true }</pre> |
| 3 | Нет, заполнить повторно | Форма в виде keyboard кнопки | <pre>let mode = memory.getAttr('mode') if (mode === 'tg_keyboard') { return true }</pre> |

Условия кнопок необходимы для вызова скрипта соответствующего текущему формату формы.

Условие для вывода универсальной формы в виде ссылки:

```
let mode = memory.getAttr('mode')
if (mode === 'tg_inline') {
  return true
}
```

Условие для вывода формы в Web App при нажатии на inline-кнопку:

```
let mode = memory.getAttr('mode')
if (mode !== 'tg_inline' && mode !== 'tg_keyboard') {
  return true
}
```

Условие для вывода формы в Web App при нажатии на keyboard-кнопку:

```
let mode = memory.getAttr('mode')
if (mode === 'tg_keyboard') {
  return true
}
```

Web App форма с использованием keyboard-кнопки (только для Telegram)

Инструкция для разработчика бота.

Данный вид формы работает только в Telegram.

Ключевая особенность: форма открывается в всплывающем окне на котором расположен WebView (встроенный браузер). Для отправки данных в Metabot API не требуется дополнительный backend.

Актуальный исходный код веб-формы реализующий все три вида форм смотрите по ссылке: [go-to-the-mars.html](https://github.com/metabot-dev/go-to-the-mars.html)

Пример работы веб-формы приведенной выше смотрите в Telegram боте https://t.me/metabot_test_form_bot

Отличие данного варианта от универсальной формы в виде ссылки, в том, что форма работает как Web App, т.е. это специальный режим браузера встроенного в Telegram. Этот режим гарантирует возврат в бота после заполнения формы, а также не позволит открыть ссылку во внешнем браузере (а также включает дополнительный функционал для взаимодействия веб-страницы с ботом). В обычном же режиме для универсальной формы, нет гарантий, что когда веб-страница будет закрыта, пользователя перекинет в бота.

Краткое описание принципа работы бота:

1. В бот отправляется keyboard-кнопка (в формате Telegram Web App). При нажатии на кнопку открывается всплывающее окно, в которое встроен Web View, пользователь не может открыть эту ссылку во внешнем браузере, а также просмотреть исходный код формы (если только не откроет сам Telegram в браузере).
2. После заполнения формы и нажатия на кнопку Отправить данные, все данные будут отправлены в Metabot, в команду JavaScript Callback, Internal API endpoint в данном случае не требуется. Для понимания смотрите исходный код веб-формы.
3. Данные пришедшие с формы доступны в команде JavaScript Callback в виде обычного вебхука. Необходимо сохранить их в таблице.
4. Далее необходимо вызвать скрипт бота для продолжения беседы, этот скрипт работает с сохраненными данными формы, запрашивает подтверждение, что все введено верно, а также предлагает заполнить форму повторно.

Принцип работы, по шагам:

- Бот генерирует ссылку с уникальным хэш-кодом, привязанным к лиду.
- Бот отправляет эту ссылку в виде inline-кнопки в Telegram-мессенджер.
- Пользователь бота нажимает на кнопку, открывается Web App с формой (т.е. пользователь остается в Telegram, в мессенджере открывается встроенный браузер).
- Пользователь бота заполняет форму и нажимает кнопку для отправки формы.
- Форма отправляет данные на backend сайта, где размещена форма.
- Backend сайта отправляет данные формы в API Metabot, в данные должен быть включен уникальный хэш лида.
- Metabot принимает и сохраняет данные заполненной формы.
- Если все ок, то страница с формой в браузере должна быть закрыта, это делается с помощью простого JS кода (подробности рассматриваются на отдельной странице документации с описанием создания HTML-формы).

Таблицы для работы с формами

Чтобы рассматриваемый пример бота с формами работал, вам необходимо создать две кастомные таблицы.

Таблица: Хэш-коды лидов

Таблица необходима для хранения соответствий между лидом и хэшем для лида.

При каждом вызове формы выполняется поиск хэша по лиду, если хэш не найден, то формируется новый. После отправки формы, запись с хэш-кодом удаляется, чтобы ссылка на каждый новый опрос содержала уникальный хэш. Поиск предыдущего хэша необходим, если есть формы, ожидающие заполнения, для случаев, если лид закрыл форму, а затем спустя время решил ее заполнить, нажав на ту же кнопку вызова формы. При этом в

таблицу заложено поле (expire_at) - дата истечения хэш-кода, вы можете использовать это поле для реализации времени жизни хэш-кода, но в таком случае в веб-форме необходимо предусмотреть уведомление о том, что форма истекла, или закрывать ее автоматом после истечения, или формировать в боте новую ссылку, если лид пытается отправить форму, хэш которой истек. Это требование не обязательно, может реализовываться по мере необходимости и зависит от конкретной задачи.

Структура таблицы:

| ID | АКТИВНА | СИХР. СТРУКТУРЫ | ОТОБРАЖАТЬ В МЕНЮ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК В ИНТЕРФЕЙСЕ |
|----|---------|-----------------|-------------------|----------------|------------------------|
| 69 | Да | Да | Да | lead_form_hash | Leads Hashes for Forms |

Структура таблицы (JSON словарь):

| АКТИВНО | СИХР. СТРУКТУРЫ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК | ТИП | ОБЯЗАТЕЛЬНОЕ | ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ | РАЗМЕР | ТОЧНОСТЬ | ПОДСКАЗКА | ВСПЛЫВАЮЩАЯ ПОДСКАЗКА |
|---------|-----------------|--------------|------------|---------------|--------------|-----------------------|--------|----------|-----------|-----------------------|
| Да | Да | id | ID | AUTOINCREMENT | Да | NULL | | | | |
| Да | Да | form | Form | TEXT | Да | | | | | |
| Да | Да | lead_id | Lead ID | BIG_INT | Да | NULL | | | | |
| Да | Да | hash | Hash | TEXT | Да | | | | | |
| Да | Да | expire_at | Expire At | DATETIME | | NULL | | | | |
| Да | Да | created_at | Created At | CREATED_AT | Да | NOW | | | | |

Таблица: Данные форм

Таблица необходима для хранения данных заполненных форм.

Данная таблица реализована как универсальная, все данные пришедшие с формы сохраняются в текстовом поле в виде JSON, поэтому она может работать с любой формой, но работать с строкой в виде JSON в JS неудобно, тк для доступа к данным приходится выполнять JSON.parse(), вы можете нормализовать данные, реализуя для каждой формы бота отдельную таблицу с необходимыми полями для хранения, например такими как: Имя, Фамилия, Адрес и т.д., чтобы в дальнейшем было проще работать с данными и выполнять поиск по таблице. Также, в вашем случае таблица для хранения данных может и не потребоваться, если всплывающая форма очень простая и предназначена, например, для заполнения одного поля, например, адреса с авто комплитом от Dadata или для выбора выпадающего списка или даты в календаре. Т.е. форма может реализовывать даже один компонент для заполнения которого необходим HTML + JS.

Структура таблицы:

| ID | АКТИВНА | СИХР. СТРУКТУРЫ | ОТОБРАЖАТЬ В МЕНЮ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК В ИНТЕРФЕЙСЕ |
|----|---------|-----------------|-------------------|--------------|------------------------|
| 70 | Да | Да | Да | form_results | Form Results |

Структура таблицы (JSON словарь):

| АКТИВНО | СИХР. СТРУКТУРЫ | НАИМЕНОВАНИЕ | ЗАГОЛОВОК | ТИП | ОБЯЗАТЕЛЬНОЕ | ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ | РАЗМЕР | ТОЧНОСТЬ | ПОДСКАЗКА | ВСПЛЫВАЮЩАЯ ПОДСКАЗКА |
|---------|-----------------|--------------|------------|---------------|--------------|-----------------------|--------|----------|-----------|-----------------------|
| Да | Да | id | ID | AUTOINCREMENT | Да | NULL | | | | |
| Да | Да | form | Form | TEXT | Да | | | | | |
| Да | Да | lead_id | Lead ID | BIG_INT | Да | NULL | | | | |
| Да | Да | data | Data | TEXT | | | | | | |
| Да | Да | created_at | Created At | CREATED_AT | Да | NOW | | | | |
| Да | Да | updated_at | Updated At | UPDATED_AT | | NULL | | | | |

Чтобы посмотреть [исходный код](#) внутреннего API, сначала перейдите в бота по ссылке на любой из скриптов.

Нажмите в предупреждающем сообщении на ссылку для смены активного бота, а затем перейдите на страницу с внутренним API в настройках бота или [по ссылке](#).

Если у вас нет доступа к указанному скрипту, то запросите шаблон бота в технической поддержке Metabot.

Скрипт для формирования ссылки и отправки кнопки в мессенджер



[Пример скрипта](#).

Скрипт отправки ссылки состоит из единственной команды бота:

- **Выполнить JavaScript Callback:**

КОМАНДЫ

Эти команды бот выполняет до того как покажет меню.

| КОМАНДА | СОДЕРЖИМОЕ | ОПЕРАЦИИ |
|-------------------------------|---|---|
| Выполнить JavaScript Callback | <div>Запустить Callback сразу при старте этой команды: Да</div> <div>JavaScript Callback:</div> <div>// Код скрипта</div> |   |

```
const menuHelper = require(' Common. TelegramComponents. MenuHelper' )
let md5 = require(' Common. Utils. Md5' )
```

```
//
```

```
-----
```

```
// Инициализация компонента
```

```
if (menuHelper.isFirstImmediateCall()) {
```

```
    // !!! ИНИЦИАЛИЗАЦИЯ !!!
```

```
    // Скрываем предыдущее меню с кнопкой формы
```

```
    menuHelper.removeInlineKeyboard()
```

```
    // Сбрасываем ID последнего сообщения
```

```
    menuHelper.clearLastTelegramMessageId()
```

```
    // -----
```

```
    //ГЕНЕРИРУМ ССЫЛКУ С ХЭШЕМ
```

```
    let expireAt = new Date()
```

```
    expireAt.setSeconds(expireAt.getSeconds() + 300)
```

```
    let hashItems = table.find(' lead_form_hash', [], [
        ['form', '=', 'mars'],
        ['lead_id', '=', leadId],
    ])
}
```

```
// Todo: Можно вынести в снippet
```

```
let leadHash
```

```
if (hashItems.length > 0) {
```

```

    leadHash = hashItems[0].hash
  } else {
    const salt = 'YourSuperSecretString' + (new Date()).getTime()
    leadHash = md5(salt + leadId)

    table.createItem('lead_form_hash', {
      'form': 'mars',
      'lead_id': leadId,
      'hash': leadHash,
      'expireAt': expireAt
    })
  }

  // Вы можете вынести эту ссылку в атрибуты бота
  let formUrl = 'https://app.metabot24.com/testWidgets/go-to-the-
mars.html?mode=tg_keyboard&q=' + leadHash
  // -----

  // Отправляем кнопку с формой при инициализации
  let message = 'Для продолжения, пожалуйста заполните форму'
  let buttons = [
    [
      {
        "text": "Заполнить форму",
        "web_app": {
          "url": formUrl
        }
      }
    ]
  ]
  let apiAdditionalParams = {
    "reply_markup": {
      "keyboard": buttons,
      "resize_keyboard": true // Для того чтобы кнопка не была слишком большой по высоте
    }
  }
  menuHelper.sendMessage(message, null, null, apiAdditionalParams)

  return false
}

```

```

//
-----

//
-----

// ЛОГИКА КОМПОНЕНТА
//
-----

//
-----

// Обработка нажатых кнопок или входящего текста

let webhook = bot.getWebhookPayload()
let webAppData = null
let data = null
let rawData = null

if (typeof webhook.message == 'object') {
  if (typeof webhook.message.web_app_data == 'object') {
    webAppData = webhook.message.web_app_data

    //Example:
    // "web_app_data": {
    //   "button_text": "Заполнить форму",
    //   "data": "{...}"
    // }

    if (typeof webAppData.data === 'string') {
      rawData = webAppData.data
      data = JSON.parse(rawData)
      if (typeof data !== 'object') {
        rawData = null
        data = null
      }
    }
  }
}

```

```

    }
  }

  // -----
  if (data !== null && typeof data === 'object' && typeof data.q === 'string') {
    // СОХРАНЕНИЕ ДАННЫХ
    // КОД ИДЕНТИЧНЫЙ API INTERNAL ENDPOINT
    // TODO: МОЖНО ВЫНЕСТИ В СНИППЕТ

    // Доп защита
    // Все равно проверяем хэш (на случай чтобы не взломали)
    // А также чтобы почистить формы ожидающие заполнения по лиду (хэши по лиду)
    // Можно не выполнять эту проверку здесь
    // Todo: Можно вынести в сниппет
    let hashItems = table.find('lead_form_hash', [], [
      ['form', '=', 'mars'],
      ['hash', '=', data.q],
    ])

    let found = 0

    if (hashItems.length > 0) {
      formLeadId = hashItems[0].lead_id
      hashItems.forEach(hashItem => hashItem.delete())
    }

    // Доп защита
    // Проверяем что к нам пришла форма принадлежащая лиду
    // Можно не выполнять эту проверку здесь
    if (formLeadId === leadId) {
      // Удаляем предыдущие ответы
      // Todo: Можно вынести в сниппет
      oldResults = table.find('form_results', [], [
        ['form', '=', 'mars'],
        ['lead_id', '=', formLeadId],
      ])
      if (oldResults.length > 0) {
        oldResults.forEach(oldResult => oldResult.delete())
      }
    }
  }
}

```

```

// Предварительно сохраняем данные в таблице
table.createItem('form_results', {
    "form": "mars",
    "lead_id": formLeadId,
    "data": rawJsonData
})

// Удаляем кнопку калькулятора, если она есть !
// Внимание! Удалять можно только с отправкой сообщения!
menuHelper.removeReplyKeyboardWithMessage(' Спасибо, данные получены! ')

//bot.scheduleScriptByCode('after_submit', formLeadId)

// Выходим из цикла и запускаем скрипт уведомления о принятой форме
return {
    "break": true,
    "run_script_by_code": "after_submit"
}
} else {
    // Можно вывести сообщение - что чтото прилетело из того что не ожидали
    // или выслать письмо администратору
    // ...
}
}
// -----

// Todo: Здесь можно повторно отправить кнопку, тк она скрывается с экрана если отправить
сообщение в бота
bot.sendText('Пожалуйста, заполните форму, для этого нажмите на кнопку в нижней части
мессенджера, '
    + ' после этого откроется форма, которую вам необходимо заполнить'
    + ' и нажать кнопку на форме "Отправить заявку"')

return false

//
-----
-----

```

Обратите внимание на строки с кодом запуска скрипта, после сохранения данных с формы:

```
// Выходим из цикла и запускаем скрипт уведомления о принятой форме
return {
  "break": true,
  "run_script_by_code": "after_submit"
}
```

Этот код может быть заменен на "return true" для выхода из цикла и на команду бота - "Выполнить скрипт", которую в таком случае необходимо разместить после команды JavaScript Callback. Подробнее смотрите в описании команды Выполнить JavaScript Callback.

Скрипт выполняемый после сохранения данных

Данный скрипт запускается из JavaScript Callback, код которой приведен выше.

Данный скрипт нужен чтобы пользователь продолжил диалог с ботом, а также демонстрирует как использовать данные сохраненные в таблицу результатов и вывести их в виде текста.

[Пример скрипта.](#)

Скрипт отправки ссылки состоит из следующих команд бота и пунктов меню:

- Команда **Выполнить JavaScript**:

```
const menuHelper = require('Common.TelegramComponents.MenuHelper')

// -----
// Удаляем кнопку с формой для Inline формы, если она выведена

if (menuHelper.hasLastTelegramMessageId()) {
  // Удаляем кнопку с ссылкой на форму
  menuHelper.removeInlineKeyboard()

  // Сбрасываем ID последнего сообщения
  menuHelper.clearLastTelegramMessageId()
}

// -----
// -----
```

```

// Для Inline кнопки с формой
// Удаляем кнопку с формой , если она выведена
// Удаляем здесь - тк после заполнения формы в JS-Callback мы уже не попадаем
// Тк прием данных с формы осуществляет через Internal Endpoint
// Для keyboard кнопки ссылку на форму с кнопкой здесь удалять не нужно,
// тк для keyboard кнопки мы возвращаемся в JS-Callback после заполнения формы
if (menuHelper.hasLastTelegramMessageId()) {
    // Удаляем кнопку
    menuHelper.removeInlineKeyboard()

    // Сбрасываем ID последнего сообщения
    menuHelper.clearLastTelegramMessageId()
}
// -----

// Если получаем данные напрямую, без таблицы form_results
//let data = request.json

// Если получаем данные из таблицы form_results
// Todo: Можно вынести в сниппет
let data = table.find('form_results', [], [
    ['form', '=', 'mars'],
    ['lead_id', '=', leadId]
])

if (data.length > 0 && typeof(data[0].data) === 'string' && data[0].data.length > 0) {
    data = JSON.parse(data[0].data)
} else {
    data = {}
    bot.sendText('Ошибка! Данные не найдены')
}

// Для определения какую кнопку показывать в меню ниже
// См условие аунктов меню текущего скрипта
memory.setAttr('mode', '')
if (typeof(data.mode) === 'string' && data.mode.length > 0) {
    memory.setAttr('mode', data.mode)
}

```



```

/*
// Сообщение о результате обработки формы, отправляется в бота
// Но проблема что система распознает сообщение как вебхук от лида
// Поэтому использовать можно только в JS-Callback и игнорировать вебхук вручную
// Или через регулярку маршрута, запуская отдельны скрипт
//
// Данное сообщение также автоматом закрывает web-view
// Но пробелм с закрытием нет, это выполняется в JS итак с помощью
// кода window.Telegram.WebApp.sendData(JSON.stringify(formData));
// или при выполнении window.Telegram.WebApp.close()
//
// Обычно используется для игры в web_view для вывода очков после победы или проигрыша
if (data.tg_query_id && data.tg_query_id.length > 0) {
    //https://core.telegram.org/bots/api#sentwebappmessage
    //https://core.telegram.org/bots/api#inlinequeryresultarticle
    bot.sendPayload('answerWebAppQuery', {
        "web_app_query_id": data.tg_query_id,
        "result": {
            "type": "article",
            "id": "1", // Unique identifier for this result, 1-64 Bytes
            "title": "Форма заполнена",
            //"description": "Description",
            "input_message_content": {"message_text": "Спасибо! Данные сохранены!"}
        }
    })
}
*/

memory.setJsonAttr('data', data)
memory.setAttr('is_qualified', data.is_qualified ? 'Да' : 'Нет')
memory.setAttr('has_experience', data.has_experience ? 'Да' : 'Нет')

```

- Команда **Отправить текст:**

Ваши данные:

Имя: {{ &\$\$data.name }}

Email: {{ &\$\$data.email }}

Возраст: {{ &\$\$data.age }}

Профессия: {{ &\$\$data.specialization}}

Ваш адрес проживания: {{ &\$\$data.address}}

Прошел курсы в Центре подготовки космонавтов: {{ &\$is_qualified }}

Я уже летал в космос (имею опыт): {{ &\$has_experience }}

Все данные в виде JSON:

{{ &\$\$data }}

- Команда **Отправить текст:**

Все верно?

- Пункты меню данного скрипта, с условиями:

| КОД | НАДПИСЬ | СКРИПТ | УСЛОВИЕ ВЫВОДА |
|-----|-------------------------|------------------------------|--|
| 1 | Да, все верно! | Подтвердил сохранение | |
| 2 | Нет, заполнить повторно | Форма в виде ссылки | <pre>let mode = memory.getAttr('mode') if (mode !== 'tg_inline' && mode !== 'tg_keyboard') { return true }</pre> |
| 2 | Нет, заполнить повторно | Форма в виде inline кнопки | <pre>let mode = memory.getAttr('mode') if (mode === 'tg_inline') { return true }</pre> |
| 3 | Нет, заполнить повторно | Форма в виде keyboard кнопки | <pre>let mode = memory.getAttr('mode') if (mode === 'tg_keyboard') { return true }</pre> |

Условия кнопок необходимы для вызова скрипта соответствующего текущему формату формы.

Условие для вывода универсальной формы в виде ссылки:

```
let mode = memory.getAttr('mode')
if (mode === 'tg_inline') {
  return true
}
```

Условие для вывода формы в Web App при нажатии на inline-кнопку:

```
let mode = memory.getAttr('mode')
if (mode !== 'tg_inline' && mode !== 'tg_keyboard') {
```

```
    return true  
}
```

Условие для вывода формы в Web App при нажатии на keyboard-кнопку:

```
let mode = memory.getAttr('mode')  
if (mode === 'tg_keyboard') {  
    return true  
}
```