1. Поясните принцип работы LongPoll-сервера?

Бот подключается к Telegram API через <https://api.telegram.org/bot><token> и запрашивает обновления методом getUpdates с использованием Long Poll (параметр timeout=60).  
Каждое сообщение, которое бот получает, обрабатывается в handleUpdate, где извлекается text и chatId, а затем вызывается sendMessage с echo: сообщение.  
Функция longPoll() вызывается рекурсивно через setTimeout — это и есть реализация Long Poll.

Принцип работы:

1.Клиент (в данном случае бот) отправляет HTTP-запрос к серверу Telegram с методом getUpdates, добавляя timeout — например, 60 секунд.  
2.Сервер не отвечает сразу, а ждет:  
либо появления новых сообщений (update),  
либо окончания тайм-аута.  
3.Как только появляется новое сообщение, сервер отправляет ответ с данными.  
4.После получения ответа бот сразу отправляет следующий getUpdates, начиная с offset = последний update\_id + 1, чтобы не обрабатывать одно и то же дважды.  
Итог: Long Poll позволяет получать обновления почти в реальном времени, но без постоянных коротких опросов сервера, экономя ресурсы и трафик.

express — нужен, чтобы запустить простой веб-сервер и показать сообщение "бот работает".  
node-fetch — нужен, чтобы отправлять HTTP-запросы к Telegram API.

const app = express();

const PORT = process.env.PORT || 3000;

Создаем приложение express. Оно будет слушать порт **3000** (или тот, который указан в переменных окружения).

const TELEGRAM\_API\_URL = 'https://api.telegram.org/bot<токен>';

Это базовый адрес Telegram API. Ты подставил токен своего бота.

const LONG\_POLL\_URL = ${TELEGRAM\_API\_URL}/getUpdates;  
Этот URL используется для запроса новых сообщений от пользователей (через long poll).

let lastUpdateId = 0;  
Сохраняем ID последнего обработанного сообщения, чтобы не получать одно и то же снова.

Что происходит при новом сообщении:  
const handleUpdate = (update) => {  
const message = update.message;  
if (message) {  
 const chatId = message.chat.id;  
 const text = message.text;  
  
 sendMessage(chatId, `echo: ${text}`);  
}  
};  
```  
Когда приходит новое сообщение,  
Берем [chat.id](https://web.telegram.org/a/chat.id) — это ID пользователя/чата,  
Берем text — это текст сообщения,  
Отправляем обратно: echo: тот же текст.

Отправляем сообщение пользователю  
const sendMessage = (chatId, text) => {  
 const url = `${TELEGRAM\_API\_URL}/sendMessage?chat\_id=${chatId}&text=${encodeURIComponent(text)}`;  
 fetch(url).catch(err => console.error('Error sending message: ', err));  
};  
```  
Формируем URL sendMessage,  
Отправляем запрос fetch,  
Используем encodeURIComponent, чтобы текст не сломал URL (если есть пробелы, точки и т.д.).

``const longPoll = () => {  
fetch(${LONG\_POLL\_URL}?offset=${lastUpdateId + 1}&timeout=60)  
.then(response => response.json())  
.then(data => {  
 if (data.result && data.result.length > 0) {  
 data.result.forEach(update => {  
 lastUpdateId = update.update\_id;  
 handleUpdate(update);  
});  
}  
})  
.catch(err => console.error('Error getting updates: ', err));  
 setTimeout(longPoll, 1000);  
};``

getUpdates спрашивает: “Есть новые сообщения?”  
offset=${lastUpdateId + 1} — чтобы не получать старые сообщения  
timeout=60 — ждём до 60 секунд, если сообщений нет  
Если получены result[], то:  
вызываем handleUpdate для каждого,  
сохраняем последний update\_id  
Через setTimeout перезапускаем функцию снова через 1 секунду — это наш бесконечный цикл.

**Long Polling** — это способ общения между клиентом (например, ботом) и сервером (например, Telegram), при котором клиент **ждет ответа дольше обычного**. В отличие от обычного запроса, который быстро завершается, Long Poll может **висеть до 60 секунд**, ожидая события.

Обычный запрос (Short Poll):

1. Клиент спрашивает: "Есть новое сообщение?"

2. Сервер: "Нет"

3. Клиент снова спрашивает через 1 секунду...

Это создает **много запросов** и **нагрузку**.

Long Poll:

1. Клиент спрашивает: "Есть новое сообщение? Я подожду до 60 секунд."

2. Сервер:

─ Если новое сообщение появилось через 10 сек: "Вот оно!"

─ Если ничего нет за 60 секунд: "Пусто"

3. Клиент сразу делает новый запрос.

Это экономнее: **меньше запросов, меньше трафика, быстрее реакция**.

Как работает Long Poll в Telegram

1. Бот отправляет запрос:

GET [https://api.telegram.org/bot<token>/getUpdates?offset=123&timeout=60](https://api.telegram.org/bot%3ctoken%3e/getUpdates?offset=123&timeout=60)

offset=123 — с какого сообщения начать (чтобы не повторяться)

timeout=60 — бот ждет **до 60 секунд** новое сообщение

2. Сервер Telegram делает следующее:

Если **новое сообщение уже есть** — отправляет сразу

Если **ничего нет** — **держит соединение** открытым до 60 секунд

Если **сообщение появилось в это время** — отправляет его

Если **ничего не пришло за 60 секунд** — возвращает пустой ответ

**3. Бот обрабатывает ответ и делает новый запрос:**

setTimeout(longPoll, 1000); // через 1 секунду снова ждет

1. Твой бот (клиент) отправляет запрос на сервер Telegram:

fetch(`${LONG\_POLL\_URL}?offset=${lastUpdateId + 1}&timeout=60`)

Это строка — твой **бот делает запрос**.

LONG\_POLL\_URL — это URL **серверной части Telegram**.

offset и timeout говорят серверу:

* жди до 60 секунд,
* дай только новые сообщения.

1. Сервер Telegram отвечает:

.then(data => {

if (data.result && data.result.length > 0) {

data.result.forEach(update => {

lastUpdateId = update.update\_id;

handleUpdate(update); // обработать новое сообщение

});

}

})

Если пришло сообщение — **сервер Telegram отдает данные**

Ты (бот) читаешь результат и вызываешь handleUpdate

3. Бот отвечает пользователю:

sendMessage(chatId, `echo: ${text}`);

Тут **бот делает новый запрос на сервер Telegram**, чтобы ответить пользователю:

const url = `${TELEGRAM\_API\_URL}/sendMessage?...`

fetch(url)

1. Сервер Telegram доставляет сообщение обратно пользователю в Telegram.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным. Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.