

Technische Dokumentation

-


Chatbot Adapter Whatsapp

Installation

- [ngrok](#) installieren
- [node.js](#) installieren
- [twilio](#) Account erstellen, erste Nummer erhalten

Setup

- ngrok/terminal öffnen und Server mit dem Befehl “ngrok http 3000” starten

 D:\ngrok\ngrok.exe

```
D:\ngrok>ngrok http 3000
```

```
Auswählen D:\ngrok\ngrok.exe - ngrok http 3000
ngrok
Session Status      online
Account             [REDACTED]
Version             3.0.3
Region              Europe (eu)
Latency              55.893ms
Web Interface        http://127.0.0.1:4040
Forwarding            https://43b0-2003-d6-e71b-8d00-d4e4-6729-bdd6-68cd.eu.ngrok.io -> http://localhost:3000

Connections          ttl    opn    rt1    rt5    p50    p90
                     0      0      0.00   0.00   0.00   0.00
```

- bei Twilio einloggen und unter [Messaging -> Settings -> WhatsApp Sandbox Settings](#) bei dem Eingabefeld “When a message comes in” die Forwarding-Adresse + “/whatsapp” einfügen.
 - Beispiel:
`https://43b0-2003-d6-e71b-8d00-d4e4-6729-bdd6-68cd.eu.ngrok.io/whatsapp`
`[forwarding-address]/whatsapp`

```
Auswählen D:\ngrok\ngrok.exe - ngrok http 3000
ngrok
Session Status      online
Account             [REDACTED]
Version             3.0.3
Region              Europe (eu)
Latency              55.893ms
Web Interface        http://127.0.0.1:4040
Forwarding            https://43b0-2003-d6-e71b-8d00-d4e4-6729-bdd6-68cd.eu.ngrok.io -> http://localhost:3000

Connections          ttl    opn    rt1    rt5    p50    p90
                     0      0      0.00   0.00   0.00   0.00
```

Sandbox Configuration


To send and receive messages from the Sandbox to your Application, configure your endpoint URLs. [Learn more](#)

WHEN A MESSAGE COMES IN	<input type="text" value="https://43b0-2003-d6-e71b-"/>	HTTP Post ▾
STATUS CALLBACK URL	<input type="text"/>	HTTP Post ▾

- Eigene Nummer in die Sandbox aufnehmen
 - Unter der Sandbox Configuration im gleichen Fenster findet man diese Info:

Sandbox Participants

Invite your friends to your Sandbox. Ask them to send a **WhatsApp message** to  +1 415 523 8886 with code join en 

USERID
whatsapp:+4915 

- An die angegebene Nummer muss einfach der für den Account einzigartigen Sandbox code gesendet werden
 - join [sandbox-code]
- Das Projekt starten
 - In der .env "TWILIO_ACCOUNT_SID" und "TWILIO_AUTH_TOKEN" mit Werten des eigenen Accounts aktualisieren

```

❏ .env
1  TWILIO_ACCOUNT_SID=AC47488231768321392dbcd4bcf1c3fc9b
2  TWILIO_AUTH_TOKEN=9fdac793706b46457722ca3a8a705db1
3  PORT=3000

```

- Im Projekt-Terminal mit "npm start"
 - (beim erstmaligen Start zuvor einmal "npm install")

```

PS D:\Uni\SEM6\wazzupbot> npm start

> whatsappbot@1.0.0 start D:\Uni\SEM6\wazzupbot
> node index.js

Server is up and running at 3000

```

→ Nun kann dem Bot über Whatsapp geschrieben werden und die Antworten des Nordibots werden ausgegeben.

Index.js

In der "Index.js" befindet sich das Setup des Whatsapp Bots. Hier wird die von Whatsapp empfangene Nachricht an den verbundenen Bot geleitet. Die Antwort des Bots wird an die "helper-functions" übergeben, welche die verschiedenen Darstellungsvarianten generieren.

Verbindung zu Whatsapp

Wie genau die Verbindung des Whatsapp Bots mit dem lokalen Server funktioniert wird in diesem Video gezeigt: [Send & Receive Messages on WhatsApp using Twilio, NodeJS](#)

```
15 // Start the webapp
16 const webApp = express();
17
18 // Webapp settings
19 webApp.use(
20   bodyParser.urlencoded({
21     extended: false,
22   })
23 );
24 webApp.use(bodyParser.json());
25
26 // Server Port
27 const PORT = process.env.PORT;
28
29 // Home route
30 webApp.get("/", (req, res) => {
31   res.send(`Hello World!`);
32 });
```

Verbindung mit Bot

.env

In der .env lässt sich festlegen, welcher Bot vom Adapter angesprochen werden soll. Hierfür müssen in den Variablen die URL des anzusprechenden Bots, sein Refer-Key und die Conversation-ID in den entsprechenden Variablen hinterlegt werden.

```
4
5 BOT_URL=https://devbot-multichannel.assono.de/req/
6 REFER_KEY=https://devbot-multichannel.assono.de/req/
7 CONV_ID=baee3817-965c-42ed-a221-18c709c8eb64
8
```

Auslesen der Nachricht aus Whatsapp

Die aus Whatsapp erhaltene Nachricht wird in der Variable “message” gespeichert und später via HTTP-Request an die API des verbundenen Bots gesendet. Zusätzlich wird die ID des Absenders gespeichert, um dieser die entsprechende Antwort zuordnen zu können.

```
34 // Route for WhatsApp
35 webApp.post("/whatsapp", async (req, res) => {
36   let message = req.body.Body;
37
38   // check if buttons or disambiguation have been sent before, check if user response contains integer
39   if ((await WA.getMessageContext()) == "buttons" && (await WA.containsNumber(message))) {
40     message = await WA.getButtonAnswer(message); //set user response to bot to button content matching integer sent by user
41   }
42
43   let senderID = req.body.From;
44 }
```

Config und Axios

Das Projekt nutzt den HTTP-Client “axios”, welcher HTTP-Requests auf einem NodeJS-Server oder im Browser ermöglicht. Innerhalb der Variable “config” werden alle Header der HTTP-Request definiert. Die Variable “data” enthält den Body der Request, welcher die Daten enthält, die an die API gesendet werden.

Sobald die HTTP-Request beantwortet wird, werden die übermittelten Daten in der Variable “answer” gespeichert, über welche der Rest des Codes auf die Daten der API-Response zugreift

conversation_id	Jede Konversation hat ihre eigene ID, damit unter Konversationen unterschieden werden kann. (Konfiguration siehe .env)
parentReferrer	Teilt Server mit, woher/worüber die Anfrage kommt (Konfiguration siehe .env)
message	Usernachricht (siehe Auslesen der User-Nachrichten) Variable: txt
text	Usernachricht (siehe Auslesen der User-Nachrichten) Variable: txt

```
45 // HTTP-Request body from user message in whatsapp
46 var data = JSON.stringify({
47   context: {
48     conversation_id: process.env.CONV_ID,
49     frontend_params: {},
50     parentReferrer: process.env.REFER_KEY,
51   },
52   message: message,
53   input: {
54     text: message,
55   },
56 });
57
58 // HTTP-Request Header configuration
59 var config = {
60   method: "post",
61   url: process.env.BOT_URL,
62   headers: {
63     Referer: process.env.REFER_KEY,
64     "Content-Type": "application/json",
65   },
66   data: data,
67 };
```

Parsen der Antwort des angebundenen Bots

Nach Erhalt wird der Body der HTTP-Response an die Funktion "send-message".

```
69 // on http response, save response data in var "answer"
70 let answer;
71 await axios(config)
72   .then(function (response) {
73     answer = response.data;
74   })
75   .catch(function (error) {
76     console.log(error);
77   });
78
79 // parse answer and senderID to helper functions for further processing
80 await WA.sendMessage(answer, senderID);
81 });
```

whatsapp-send-message.js

Diese Funktion ist quasi der Kreisverkehr des Programms. Sie wird für jeden Eintrag des Array "messages" aus der vom Bot erhaltenen Antwort ausgeführt. Über den sich im Eintrag befindenden Typ der Message lässt sich bestimmen, wie sie dargestellt werden muss, bzw. wie sich ihr Content extrahieren lässt.

Der Typ der Nachricht wird in der Variable "messageType" gespeichert, und je nach Wert wird eine andere helper-funktion aufgerufen, die die Antwort des Bots verarbeitet.

```
11 // Function to send message to WhatsApp
12 const sendMessage = async (answer, senderID) => {
13   //Filter out the messageType from Nordi Bot message
14   for (var i = 0; i < answer.messages.length; i++) {
15     for (let j = 0; j < answer.messages[i].content.length; j++) {
16       var messageType = JSON.stringify(answer.messages[i].content[j].type).slice(1, -1);
17
18       // call different helper-function for every message depending on messageType for further message processing
19       if (messageType == "markdown" || messageType == "html" || messageType == "plaintext") {
20         await html.sendMessage(answer, senderID, i, j);
21       } else if (messageType == "button") {
22         messageContext = "buttons"; // Log that buttons have been sent
23         button.sendMessage(answer, senderID, i);
24         buttonAnswers = await button.getOptions(); // get content of last buttons sent
25       } else if (messageType == "link" || messageType == "youtube") {
26         await link.sendLink(answer, senderID, i);
27       } else if (messageType == "slider") {
28         await slider.sendSlider(answer, senderID, i);
29       } else if (messageType == "video" || messageType == "audio" || messageType == "image") {
30         await media.sendMedia(answer, senderID, i);
31       } else if (messageType == "disambiguation") {
32         messageContext = "buttons"; // Log that buttons have been sent
33         disambiguation.sendDisambiguation(answer, senderID, i);
34         buttonAnswers = await disambiguation.getOptions(); // get content of last buttons sent
35       }
36     }
37   }
38 };
```


send-button.js

Die Buttons des Nordi-Bots werden mit Hilfe einer Liste in Whatsapp dargestellt, da das Anzeigen von tatsächlich klickbaren Buttons in dieser Form nicht möglich ist. Um mit einem den Buttons zu interagieren, können Nutzer*innen die jeweilige Stelle des Buttons in der Liste als Zahl antworten. Der Adapter sendet automatisch den richtigen Button-Text an Nordi. Der Kontext dieser Buttons wird gespeichert, so dass Nutzer*innen auch nach dem Absenden anderer Nachrichten noch Zugriff auf die Buttons durch die Eingabe von Zahlen haben.

Dazu werden in "send-button.js" zwei Arrays angelegt. In "content" werden die einzelnen Buttons gespeichert. Diese werden später an "buttonAnswers" übergeben, was ermöglicht das Array auch außerhalb dieser Funktion, mit "getOptions" auszulesen.

Das zweite Array "bullets" enthält ebenfalls die Buttons, jedoch so formatiert, dass bei der Ausgabe in WhatsApp eine nummerierte Liste mit den Begriffen erscheint. Zudem gibt es eine Variable "messageContext", die den Kontext des Chatverlaufes speichert. Sobald eine Nachricht mit Buttons gesendet wird, wird der Kontext auf "buttons" gesetzt.

```
1  const { get } = require("express/lib/request");
2
3  const accountSid = process.env.TWILIO_ACCOUNT_SID;
4  const authToken = process.env.TWILIO_AUTH_TOKEN;
5
6  const client = require("twilio")(accountSid, authToken, {
7    lazyLoading: true,
8  });
9
10 var buttonAnswers = []; // externally accessible array meant for export of button content
11
12 // sends list of possible responses instead of buttons to user
13 const sendMessage = async (answer, senderID, i) => {
14   let address = process.env.ADDRESS_USER + ": \n"; //formatted address of user, configurable in .env-file
15   let buttonAmount = answer.messages[i].content[0].buttons.length; //logs how many buttons will be sent
16   let options = []; // array for button content
17   let bullets = []; // array for buttons in message, displayed as bullet points
18
19   // store button content in options, store formatted bulletpoints in bullets
20   for (let j = 0; j < buttonAmount; j++) {
21     options.push(answer.messages[i].content[0].buttons[j].content[0].text);
22     bullets.push("\n" + (j + 1) + '. ' + options[j] + '*');
23   }
24   buttonAnswers = options; // button content gets stored in buttonAnswers
25   try {
26     await client.messages.create({
27       to: senderID,
28       body: address + bullets.join(" "), // send user address and bulletpoints as message
29       from: `whatsapp:+14155238886`,
30     });
31   } catch (error) {
32     console.log(`Error at sendMessage --> ${error}`);
33   }
34 };
35
36 // get/export buttoncontent for reuse on user response
37 const getOptions = async () => {
38   return buttonAnswers;
39 };
40
41 module.exports = {
42   sendMessage,
43   getOptions,
44 };
```

In "whatsapp-send-message.js" befinden sich getter-Methoden für "messageContext" und "buttonAnswers". So kann man auch in der "index.js" darauf zugreifen.

```

21     } else if (messageType == "button") {
22         messageContext = "buttons"; // Log that buttons have been sent
23         button.sendMessage(answer, senderID, i);
24         buttonAnswers = await button.getOptions(); // get content of last buttons sent

```

```

40 // export to index.js wheather buttons have been sent
41 const getMessageContext = async () => {
42     return messageContext;
43 };
44
45 //get integer sent by user and return matching button content from last buttons sent
46 const getButtonAnswer = async (i) => {
47     return buttonAnswers[i - 1];
48 };
49
50 // check if number given contain integer
51 const containsNumber = async (str) => {
52     return /\d/.test(str);
53 };

```

Um die Nachrichten von Nutzer*innen von einer Zahl zu einem Button-Text zu ändern, damit Nordi darauf antworten kann, muss die Variable "message" in der "index.js" geändert werden.

Ist der "messageContext" = "buttons" und die User-Nachricht eine Zahl, so wird der Buttontext, der sich an dieser Stelle im Array befindet, statt der Zahl in "message" gespeichert.

```

38 // check if buttons or disambiguation have been sent before, check if user response contains integer
39 if ((await WA.getMessageContext()) == "buttons" && (await WA.containsNumber(message))) {
40     message = await WA.getButtonAnswer(message); //set user response to bot to button content matching integer sent by user
41 }

```

Zusätzlich lässt sich in der [.env](#) in der Variable "ADRESS_USER" definieren, wie der Bot Nutzer*innen über die funktionsweise der Buttons, bzw. der Listen aufmerksam machen soll.

```

8
9 ADRESS_USER="Über folgende Stichworte können weitere Information erlangt werden"
10

```

send-disambiguation.js

Die Nachrichten vom Typ "disambiguation" funktionieren genauso, wie die Nachrichten vom Typ "buttons". Einziger Unterschied ist der Aufbau des JSONs, weshalb die Abfrage seines Inhalts leicht verändert werden muss.

```
1  const { get } = require("express/lib/request");
2
3  const accountSid = process.env.TWILIO_ACCOUNT_SID;
4  const authToken = process.env.TWILIO_AUTH_TOKEN;
5
6  const client = require("twilio")(accountSid, authToken, {
7    lazyLoading: true,
8  });
9
10 var buttonAnswers = []; // externally accessible array meant for export of button content
11
12 // sends list of possible responses instead of buttons to user
13 const sendDisambiguation = async (answer, senderID, i) => {
14   let address = process.env.ADDRESS_USER + ": \n"; //formatted adress of user, configurable in .env-file
15   let buttonAmount = answer.messages[i].content[0].options.length; //logs how many buttons will be sent
16   let bullets = []; // array for button content
17   let options = []; // array for buttons in message, displayed as bullet points
18
19   for (let j = 0; j < buttonAmount; j++) {
20     options.push(answer.messages[i].content[0].options[j].label);
21     bullets.push("\n" + (j + 1) + '. ' + options[j] + '*');
22   }
23   buttonAnswers = options; // button content gets stored in buttonAnswers
24   messageContext = "buttons";
25   try {
26     await client.messages.create({
27       to: senderID,
28       body: address + bullets.join(" "), // send user adress and bulletpoints as message
29       from: `whatsapp:+14155238886`,
30     });
31   } catch (error) {
32     console.log(`Error at sendMessage --> ${error}`);
33   }
34 };
35
36 // get/export buttoncontent for reuse on user response
37 const getOptions = async () => {
38   return buttonAnswers;
39 };
40
41 module.exports = {
42   sendDisambiguation,
43   getOptions,
44 };
```

send-html.js

Diese Funktion bearbeitet Nachrichten des Typs "markdown", "HTML" oder "plainText". Da WhatsApp keine anderen Formate als die eigene Syntax unterstützt, müssen etwaige Tags, welche der Darstellung des Textes dienen vor dem Senden der Nachricht entfernt werden. Gleichzeitig werden möglicherweise hinterlegte Links mit Hilfe einer Regular Expression aus der Nachricht extrahiert.

Im Anschluss wird die Nachricht ohne HTML-Tags gesendet. Sollte sie zuvor einen Link enthalten wird dieser direkt in einer separaten Nachricht hinterher gesendet.

Sollte der Link im Body und im Tag des HTML-Tags enthalten sein, wird der Body direkt mitgeschickt und die zweite Nachricht bleibt aus.

```
1  const accountId = process.env.TWILIO_ACCOUNT_SID;
2  const authToken = process.env.TWILIO_AUTH_TOKEN;
3  const address = require("./send-address");
4
5  const client = require("twilio")(accountId, authToken, {
6    lazyLoading: true,
7  });
8
9  const sendMessage = async (answer, senderID, i, j) => {
10    let string;
11
12    // Check if content contains an address
13    if (answer.messages[i].content[j].type == "address") {
14      string = await address.sendAddress(answer, i, j); // get formatted google maps Link from send.Address()
15    } else {
16      string = answer.messages[i].content[j].text;
17    }
18
19    //strip html
20    if (string.includes("href")) {
21      //extract link from message and create a new message with the link
22      var urlRegex = /(https?|ftp|file):\/\/[!-A-Z0-9+&@#\/%?~_!:\.,;]*[!-A-Z0-9+&@#\/%?~_]/i;
23      var link = string.match(urlRegex)[0];
24    }
25
26    let strippedstring = string.replace(/<[^>*>?/gm, ""); //remove html-tags from string
27
28    try {
29      await client.messages.create({
30        to: senderID,
31        body: strippedstring,
32        from: `whatsapp:+14155238886`,
33      });
34
35      // if the Link is not included in body of first message, sent second message with Link.
36      if (link != null && strippedstring.includes(link) == false) {
37        try {
38          await client.messages.create({
39            to: senderID,
40            body: link,
41            from: `whatsapp:+14155238886`,
42          });
43
44          //catch for inner try statement
45        } catch (error) {
46          console.log(`Error at sendMessage --> ${error}`);
47        }
48      }
49
50      //catch for outer try statement
51    } catch (error) {
52      console.log(`Error at sendMessage --> ${error}`);
53      console.log("hier is auch n Fehler");
54    }
55  };
56
57  module.exports = {
58    sendMessage,
59  };
60
```

send-link.js

URLs können in WhatsApp nur als normale Nachricht versendet werden, die Kennzeichnung des Links, sowie das generieren einer Vorschau, übernimmt WhatsApp selbst. Daher wird, sobald der MessageType "link" erkannt wird, nur die direkte URL aus der Antwort des Bots als Nachricht versendet.

```
1  const accountSid = process.env.TWILIO_ACCOUNT_SID;
2  const authToken = process.env.TWILIO_AUTH_TOKEN;
3
4  const client = require("twilio")(accountSid, authToken, {
5    lazyLoading: true,
6  });
7
8  // get url from answer and send it as message
9  const sendLink = async (answer, senderID, i) => {
10   try {
11     await client.messages.create({
12       to: senderID,
13       body: answer.messages[i].content[0].url,
14       from: `whatsapp:+14155238886`,
15     });
16   } catch (error) {
17     console.log(`Error at sendMessage --> ${error}`);
18   }
19 };
20
21 module.exports = {
22   sendLink,
23 };
24
```

send-media.js

Medien werden ähnlich wie Links direkt von WhatsApp selbst dargestellt. Im Vergleich hierzu wird kein Link als Nachricht versendet, sondern eine Media-URL an WhatsApp übergeben.

Wichtig hierbei ist, dass die Größe des darzustellenden Mediums 500kb nicht überschreiten darf. Diese Limitierung liegt an der verwendeten Twilio-API, welche Medien zwar "duldet", sie jedoch behandelt, wie eine versendete MMS.

```
1  const accountSid = process.env.TWILIO_ACCOUNT_SID;
2  const authToken = process.env.TWILIO_AUTH_TOKEN;
3
4  const client = require("twilio")(accountSid, authToken, {
5    lazyLoading: true,
6  });
7
8  // send content-URL as media-URL, which whatsapp load internally (file size can't exceed 500kb)
9  const sendMedia = async (answer, senderID, i) => {
10   try {
11     await client.messages.create({
12       to: senderID,
13       mediaUrl: [answer.messages[i].content[0].url],
14       from: `whatsapp:+14155238886`,
15     });
16   } catch (error) {
17     console.log(`Error at sendMessage --> ${error}`);
18   }
19 };
20
21 module.exports = {
22   sendMedia,
23 };
24
```

send-slider.js

Slider werden in WhatsApp als einzelne Nachrichten dargestellt. Zunächst wird eine “mediaUrl” übergeben, zu dem ein “body” mit der dazugehörigen Beschreibung gefügt wird. Da sich in WhatsApp keine Hyperlinks darstellen lassen, wird der dazugehörige Link ebenfalls zum “body” hinzugefügt.

```
1  const accountSid = process.env.TWILIO_ACCOUNT_SID;
2  const authToken = process.env.TWILIO_AUTH_TOKEN;
3
4  const client = require("twilio")(accountSid, authToken, {
5    lazyLoading: true,
6  });
7
8  // sends a message for every image in answer
9  const sendSlider = async (answer, senderID, i) => {
10   for (let j = 0; j < answer.messages[i].content[0].slides.length; j++) {
11     try {
12       await client.messages.create({
13         to: senderID,
14         mediaUrl: answer.messages[i].content[0].slides[j].image.url, // content-URL for image in whatsapp
15         body:
16           answer.messages[i].content[0].slides[j].image.name + // add image name as headline to image description
17           "\n\n" + // empty line between headline and url in image description
18           answer.messages[i].content[0].slides[j].link.url, // add Topic-URL to description
19         from: `whatsapp:+14155238886`,
20       });
21     } catch (error) {
22       console.log(`Error at sendMessage --> ${error}`);
23     }
24   }
25 };
26
27 // lajloK ollaH
28
29 module.exports = {
30   sendSlider,
31 };
```

send-address.js

Der messageType “address” folgt im Aufbau als einziger einem anderen Muster. Wenn eine Google Map angezeigt/verlinkt werden soll, müssen die Inhalte des zweiten Content-Objekts ausgewertet werden. Diese werden in der Variable “mapsquery” gespeichert und helfen einen Google-Maps-Link zu generieren. Der generierte Link kann dann als einfach String zurück gegeben werden.

```
1  // receives answer and returns formatted google-maps link for given adress
2  const sendAddress = async (answer, i, j) => {
3    let mapstring = answer.messages[i].content[j].address.replace(/ /g, "+"); // get adress from answer and replace whitespaces in adress with "+"
4    return "https://www.google.com/maps/place/" + mapstring; // return google maps link
5  };
6
7  module.exports = {
8    sendAddress,
9  };
```

Beispiel

Detailliertes Beispiel Anhand des Inputs "bild":

1. Auslesen der User Nachricht

Die Nachricht der User*innen wird in der Variable "message" gespeichert und in ein Template-JSON für die API eingefügt.

```
{
  "context": {
    "conversation_id": "convID",
    "frontend_params": {},
    "parentReferrer": "referKey"
  },
  "message": "bild",
  "input": {
    "text": "bild"
  }
}
```

Im gleichen Schritt wird die Telefonnummer der User*innen ausgelesen und gespeichert, um die Antwort an die selbe Nummer zurück senden zu können.

```
63 let senderID = req.body.From;
```

2. API-Call wird ausgeführt

Ein HTTP-Request mit vorher definierter Konfiguration wird ausgeführt. Bei Erhalt einer Antwort des Servers wird diese in der Variable "answer" gespeichert. Sollte die Request fehlschlagen, wird der Fehler in der Konsole geloggt.

3. Nordis Antwort Parsen

Rohe Nordi-Antwort:

```
{
  "id": "6298edb0d93426768bfead49",
  "disableCustomMessage": false,
  "avatar-state": null,
  "messages": [
    {
      "content": [
        {
          "type": "markdown",
          "text": "Mein Hintergrundbild beruhigt mich immer sehr!"
        }
      ],
      "language": "de"
    },
    {
      "content": [
        {
          "type": "image",
          "url": "https://devbot-multichannel.assono.de/files/ca4719ad-5cca-4147-9f67-e9e6d12d7e03.JPG",
          "alt": ""
        }
      ],
      "language": "de"
    }
  ],
}
```

Die Default-Funktion parsed die Länge des Message Arrays.
Jeder Index wird einmal auf den Parameter "type" überprüft.

Zuerst wird der messageType "markdown" erkannt, somit greift das erste Statement in der if-Abfrage.

Der Nachrichtentext der Antwort wird als Text geparsed und eine einfache Message wird zurück an den User geschickt.

Als nächstes wird "image" als messageType erkannt.

Auch hier greift das entsprechende Statement in der if-Abfrage. Ausschließlich die URL des Bildes muss von der Antwort von Nordi geparsed und als "mediaUrl" in die Nachricht eingefügt werden. So wird das verlinkte Bild direkt zu der angegebenen Telefonnummer gesendet.

```
29     } else if (messageType == "video" || messageType == "audio" || messageType == "image") {  
30         await media.sendMedia(answer, senderID, i);
```

```
8 // send content-URL as media-URL, which whatsapp load internally (file size can't exceed 500kb)  
9 const sendMedia = async (answer, senderID, i) => {  
10     try {  
11         await client.messages.create({  
12             to: senderID,  
13             mediaUrl: [answer.messages[i].content[0].url],  
14             from: `whatsapp:+14155238886`,  
15         });  
16     } catch (error) {  
17         console.log(`Error at sendMessage --> ${error}`);  
18     }  
19 };  
20  
21 module.exports = {  
22     sendMedia,  
23 };
```