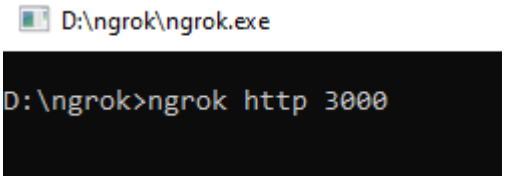
WA Technische Dokumentation

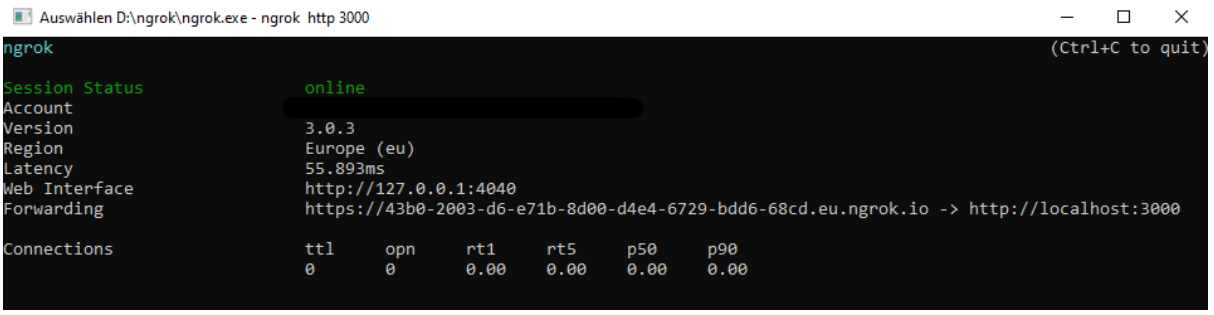
Installation

- ngrok installieren
- node.js installieren
- twilio Account erstellen, erste Nummer erhalten

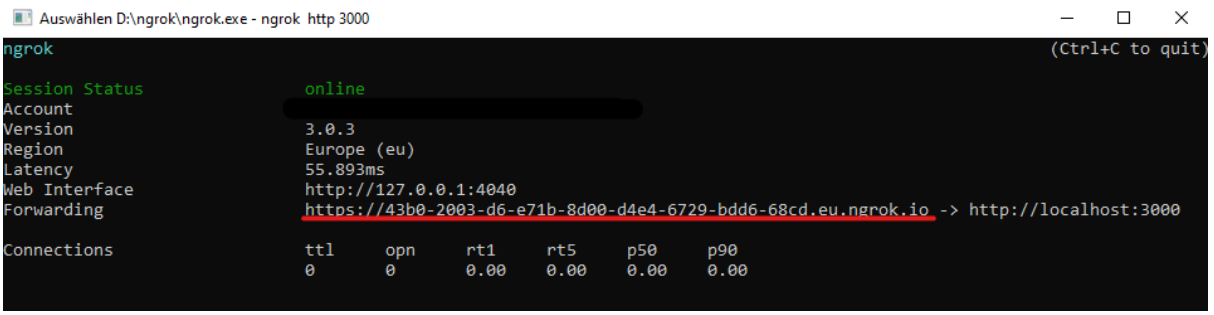
Setup

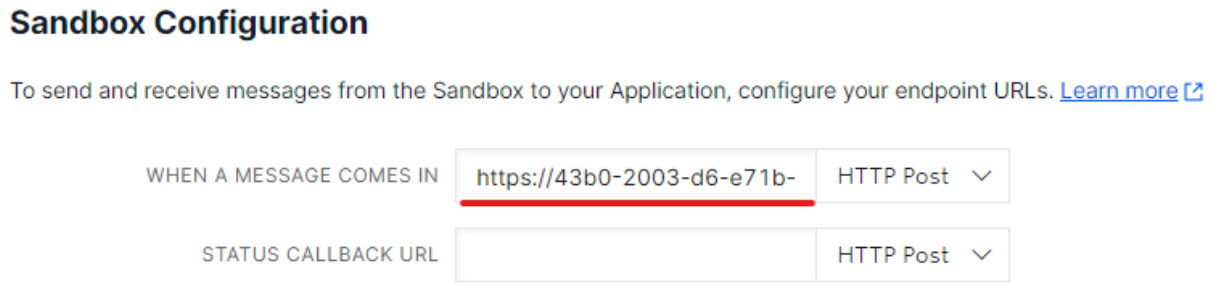
- ngrok/terminal öffnen und den server starten mit dem Befehl: “ngrok http 3000”



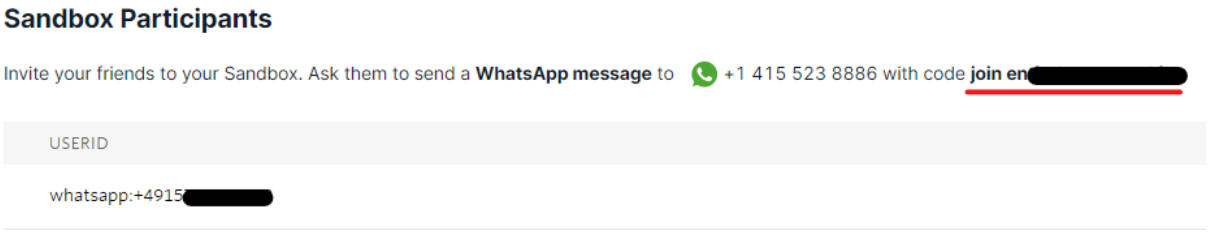


- bei Twilio einloggen und unter Messaging -> Settings -> WhatsApp Sandbox Settings bei dem Eingabefeld “When a message comes in” die Forwarding-Adresse + “/whatsapp” einfügen.
    - Beispiel: https://43b0-2003-d6-e71b-8d00-d4e4-6729-bdd6-68cd.eu.ngrok.io/whatsapp [forwarding-address]/whatsapp



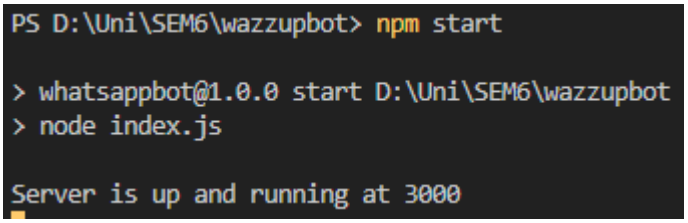


- Eigene Nummer in die Sandbox aufnehmen
    - Unter der Sandbox Configuration im gleichen Fenster findet man diese Info:



    - An die angegebene Nummer muss einfach der für den Account einzigartigen Sandbox code gesendet werden
        - join [sandbox-code]

- Das Projekt starten
    - Im Projekt-Terminal mit “npm start”
      (beim erstmaligen Start zuvor einmal “npm install”)



→ 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.

Serververbindung

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

```
// Start the webapp
const webApp = express();

// Webapp settings
webApp.use(
  bodyParser.urlencoded({
    extended: false,
  })
);
webApp.use(bodyParser.json());

// Server Port
const PORT = process.env.PORT;

// Home route
webApp.get("/", (req, res) => {
  res.send(`Hello World.!`);
});

```

Auslesen der Nordi Nachricht

```
var data = JSON.stringify({
  context: {
    conversation_id: convID,
    frontend_params: {},
    parentReferrer: referKey,
  },
  message: message,
  input: {
    text: message,
  },
});

```

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

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.

```
16  ////////////////////////////////////////////////////////////////////
17  ////////////////////BOT CONFIGURATION//////////////////////////////
18
19  // URL where the request to the bot is headed
20  const botURL = process.env.BOT_URL;
21
22  // URL the request refers to
23  const referKey = process.env.REFER_KEY;
24
25  // Conversation ID
26  const convID = process.env.CONV_ID;
27
```

```
77    var config = {
78      method: "post",
79      url: botURL,
80      headers: {
81        Referer: referKey,
82        "Content-Type": "application/json",
83      },
84      data: data,
85    };
86
```

```
89    await axios(config)
90      .then(function (response) {
91        answer = response.data;
92        console.log(answer);
93      })
94      .catch(function (error) {
95        console.log(error);
96      });
97
```

Parsen der Antwort des Nordi Bots

In der “Index.js” wird eine helper-function, namens “sendMessage” aufgerufen.

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 anderes helper-functions aufgerufen, die die Antwort des Bots verarbeitet.

```
18  // Function to send message to WhatsApp
19  const sendMessage = async (answer, senderID) => {
20    //Filter out the messageType from Nordi Bot message
21    for (var i = 0; i < answer.messages.length; i++) {
22      var messageType = JSON.stringify(answer.messages[i].content[0].type).slice(1, -1);
23
24      if (messageType == "markdown" || messageType == "html") {
25        await html.sendMessage(answer, senderID, i)
26      } else if (messageType == "button") {
27        messageContext = "buttons";
28        button.sendMessage(answer, senderID, i)
29        buttonAnswers = await button.getOptions();
30      } else if (messageType == "link" || messageType == "youtube") {
31        await link.sendLink(answer, senderID, i)
32      } else if (messageType == "slider") {
33        await slider.sendSlider(answer, senderID, i)
34      } else if (messageType == "video" || messageType == "audio" || messageType == "image") {
35        await media.sendMedia(answer, senderID, i)
36      } else if (messageType == "disambiguation") {
37        messageContext = "buttons";
38        disambiguation.sendDisambiguation(answer, senderID, i)
39        buttonAnswers = await disambiguation.getOptions();
40      }
41    }
42  };
```

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 in einer normalen Textnachricht versendet werden und wird von WhatsApp selbst formatiert.

```
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;

const client = require("twilio")(accountSid, authToken, {
  lazyLoading: true,
});

const sendAddress = async (answer, senderID, i) => {
    let mapstring= answer.messages[i].content[1].address
    let mapsquery= mapstring.replace(/ /g,"+")

    let messagestring = "https://www.google.com/maps/place/" + mapsquery

    try {
        await client.messages.create({
            to: senderID,
            body: messagestring,
            from: `whatsapp:+14155238886`,
        });
        } catch (error) {
        console.log(`Error at sendMessage --> ${error}`);
        }
};

module.exports = {
    sendAddress
}
```

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 anworten. 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.
In “send-button.js” werden dazu 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.

```
const { get } = require("express/lib/request");

const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;

const client = require("twilio")(accountSid, authToken, {
  lazyLoading: true,
});

var buttonAnswers = []

const sendMessage = async (answer, senderID, i) => {
  let adress = "Du kannst folgende Stichworte antworten, um mehr zu erfahren: ";
  let buttonAmount = answer.messages[i].content[0].buttons.length;
  let bullets = [];
  let options = []

  for (let j = 0; j < buttonAmount; j++) {
    options.push(answer.messages[i].content[0].buttons[j].content[0].text);
    bullets.push("\n" + (j + 1) + ". "*' + options[j] + '*"');
  }
  buttonAnswers = options;
  messageContext = "buttons";
  try {
    await client.messages.create({
      to: senderID,
      body: adress + bullets.join(" "),
      from: `whatsapp:+14155238886`,
    });
  } catch (error) {
    console.log(`Error at sendMessage --> ${error}`);
  }
};

const getOptions = async () => {
  return buttonAnswers;
};

module.exports = {
  sendMessage,
  getOptions,
};
```

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

```
28        } else if (messageType == "button") {
29          messageContext = "buttons";
30          button.sendMessage(answer, senderID, i)
31          buttonAnswers = await button.getOptions();
```

```
48    const getmessageContext = async () => {
49      return messageContext;
50    };
51
52    const getButtonAnswer = async (i) => {
53      return buttonAnswers[i - 1];
54    };
55
56    const containsNumber = async (str) => {
57      return /\d/.test(str);
58    };
```

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.

```
59    if ((await WA.getmessageContext()) == "buttons" && (await WA.containsNumber(message))) {
60      message = await WA.getButtonAnswer(message);
61    }
```

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.

```
12    const sendDisambiguation = async (answer, senderID, i) => {
13      let adress = "Du kannst folgende Stichworte antworten, um mehr zu erfahren: ";
14      let buttonAmount = answer.messages[i].content[0].options.length;
15      let bullets = [];
16      let options = []
17
18      for (let j = 0; j < buttonAmount; j++) {
19        options.push(answer.messages[i].content[0].options[j].label);
20        bullets.push("\n" + (j + 1) + ". "*' + options[j] + '*"');
21      }
22      buttonAnswers = options;
23      messageContext = "buttons";
24      try {
25        await client.messages.create({
26          to: senderID,
27          body: adress + bullets.join(" "),
28          from: `whatsapp:+14155238886`,
29        });
30      } catch (error) {
31        console.log(`Error at sendMessage --> ${error}`);
32      }
33    };
```

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 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.

```
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;

const client = require("twilio")(accountSid, authToken, {
  lazyLoading: true,
});

const sendMessage = async (answer, senderID, i) => {
    //strip html
    let string = answer.messages[i].content[0].text;
    let strippedstring = string.replace(/<[^>]*>?/gm, '');

    if (string.includes("href")) {
    //extract link from message and create a new message with the link
    let urlRE= new RegExp("([a-zA-Z0-9]+://)?([a-zA-Z0-9_]+:[a-zA-Z0-9_]+@)?([a-zA-Z0-9.-]+\\.[A-Za-z]{2,4})(:[0-9]+)?([^ ])+");
    let linkRaw= string.match(urlRE);
    var link= linkRaw.slice(0,1)
    console.log("raw:"+ linkRaw)
    console.log("link:"+ link)
    }


  try {
    await client.messages.create({
      to: senderID,
      body: strippedstring,
      from: `whatsapp:+14155238886`,
    });

    if (link != null) {
    try {
      await client.messages.create({
        to: senderID,
        body: link,
        from: `whatsapp:+14155238886`,
      });

      //catch for inner try statement
    } catch (error) {
      console.log(`Error at sendMessage --> ${error}`);
    }
    };

    //catch for outer try statement
  } catch (error) {
    console.log(`Error at sendMessage --> ${error}`);
  }
};

module.exports = {
    sendMessage
}
```

send-link.js

URLs können in WhatsApp nur als normale Nachricht versendet werden, die Kennzeichnung des Links übernimmt WhatsApp selbst. Daher wird, sobald der MessageType “link” erkannt wird, nur die direkte URL aus der Antwort des Bots als Nachricht versendet.

```
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;

const client = require("twilio")(accountSid, authToken, {
  lazyLoading: true,
});

const sendLink = async (answer, senderID, i) => {
    try {
        await client.messages.create({
          to: senderID,
          body: answer.messages[i].content[0].url,
          from: `whatsapp:+14155238886`,
        });
      } catch (error) {
        console.log(`Error at sendMessage --> ${error}`);
      }
};

module.exports = {
    sendLink
}
```

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 übergegeben.
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.

```
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;

const client = require("twilio")(accountSid, authToken, {
  lazyLoading: true,
});

const sendMedia = async (answer, senderID, i) => {
    try {
        await client.messages.create({
          to: senderID,
          mediaUrl: [answer.messages[i].content[0].url],
          from: `whatsapp:+14155238886`,
        });
      } catch (error) {
        console.log(`Error at sendMessage --> ${error}`);
      }
};

module.exports = {
    sendMedia
}
```

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.

```
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;

const client = require("twilio")(accountSid, authToken, {
  lazyLoading: true,
});

const sendSlider = async (answer, senderID, i) => {
    for (let j = 0; j < answer.messages[i].content[0].slides.length; j++) {
      try {
        await client.messages.create({
          to: senderID,
          mediaUrl: answer.messages[i].content[0].slides[j].image.url,
          body:
            answer.messages[i].content[0].slides[j].image.name +
            "\n \n" +
            answer.messages[i].content[0].slides[j].image.url,
          from: `whatsapp:+14155238886`,
        });
      } catch (error) {
        console.log(`Error at sendMessage --> ${error}`);
      }
    }
};

module.exports = {
    sendSlider
}
```

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.

```
8   const sendMedia = async (answer, senderID, i) => {
9       try {
10          await client.messages.create({
11            to: senderID,
12            mediaUrl: [answer.messages[i].content[0].url],
13            from: `whatsapp:+14155238886`,
14          });
15        } catch (error) {
16          console.log(`Error at sendMessage --> ${error}`);
17        }
18  };
```