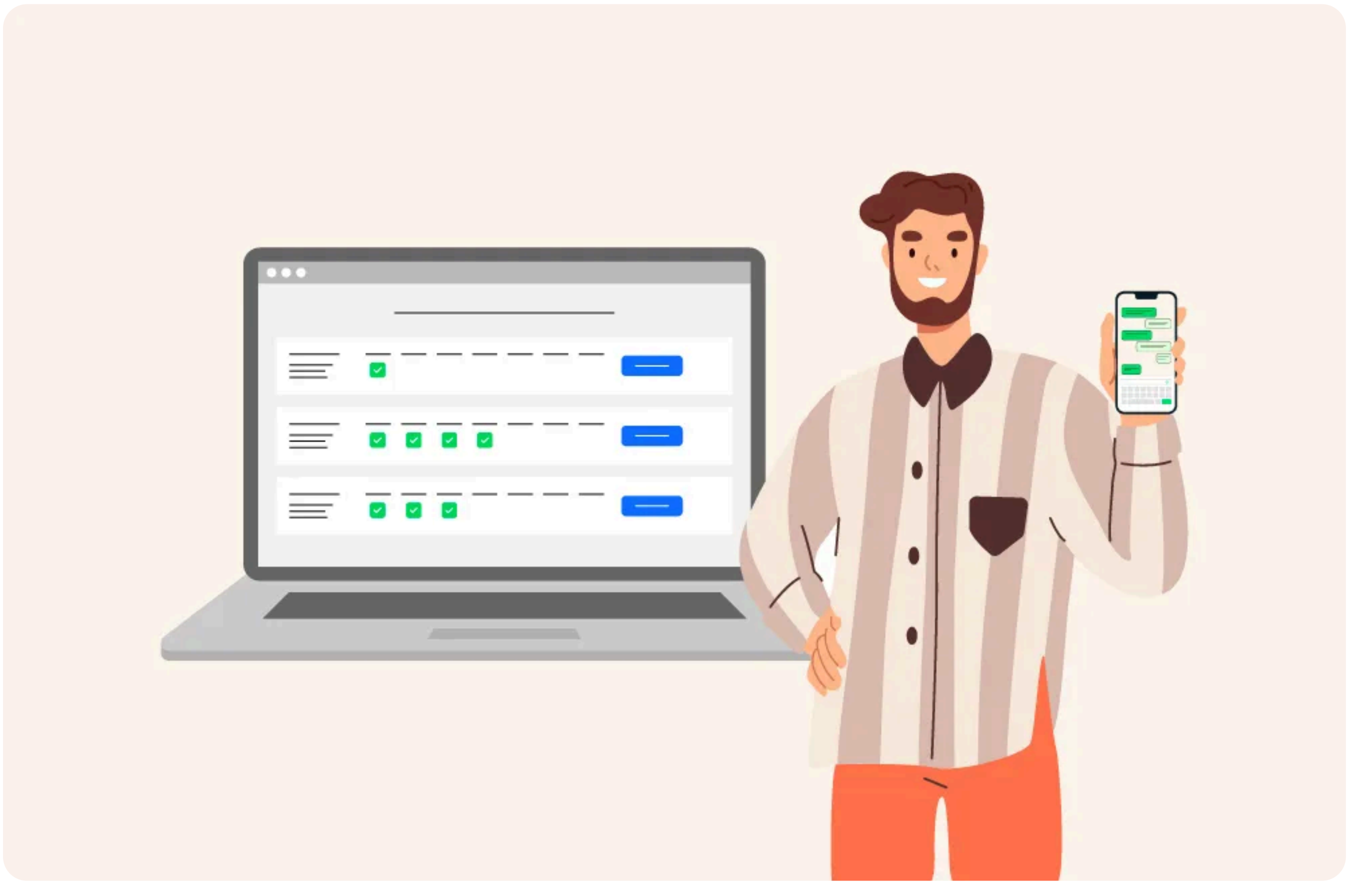
November 14, 2022

Integrating WhatsApp Into Your E-commerce Flow

By Rashed Talukder



The WhatsApp Business Platform helps businesses communicate directly with their customers using APIs to send standardized messages, receive customer responses, and reply accordingly.

This walkthrough demonstrates how a fictitious grocery store that sells products online would keep customers up-to-date on their order status. Your app will use the Cloud API, hosted by Meta, to create WhatsApp message

In the real world, order status changes according to customer actions. For instance, signing up, paying with a credit card, or finishing the purchase. You can accomplish this by integrating the WhatsApp Business Platform with your CRM or ecommerce systems, but that is outside the scope of this tutorial. Here we will be creating a manual update system so we can focus on creating and managing message templates.

Prerequisites

First, we'll go over creating a WhatsApp-powered Node.js web app. You can download the complete application code if you'd like a preview of where you'll end up.

Follow the Set up Developer Assets and Platform Access tutorial to send and receive messages using a test phone number. Make sure you include building the initial app on Meta for Developers, connecting it with WhatsApp, and associating it with a Business Manager.

You will also need an understanding of Node.js and all the required tools installed.

Building on a Minimal App with Node.js and Express

This article assumes you know how to build a basic app within Node.js. We'll provide the sample app code and show how to implement WhatsApp message template functionality.

When you download the source code, this is the folder structure you see in your code editor. We'll be working primarily in `bin, public, routes,` and `views` – as well as editing `.env, app.js,` and `package.json`:

- **bin folder:** Contains executable binaries from your node modules.
- **public folder:** Contains all the static files such as images, JavaScript, and CSS files.
- **routes folder:** Contains routes registered in the app.js file. Routes are JavaScript codes that Express.js uses to associate an HTTP verb (GET, POST, PUT, and so on) with a URL path and a function that handles HTTP requests.

- **.env file:** Contains key and value pairs defining environment variables required by the project. This file allows configuring the application without modifying the project code.
- **package.json file:** This is the heart of a Node.js project and contains the necessary metadata that the project needs to manage and install dependencies, run scripts, and identify the project's entry point.

To get started, open the package.json file located in the project root folder:

```
{
  "name": "ecommerce-node",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "axios": "^0.27.2",
    "body-parser": "^1.20.0",
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "dotenv": "^16.0.1",
    "ejs": "~3.1.8",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1"
  }
}
```

Note the requirements listed in the dependencies section. To run the application, you must install these dependencies first. Open your console and run `npm install` to install these dependencies.

This example uses a minimal app written with Node.js, Express, EJS. The app's homepage is built on Bootstrap and features a simple form you use later to create the message template that you send to customers.

Before we go any further, your Node.js application needs specific data from your developer account on Meta for Developers.

according to the tutorial mentioned in the Prerequisites section.

```
APP_ID=<<your WhatsApp business app id>>
APP_SECRET=<<your WhatsApp business app secret>>
RECIPIENT_WAID=<<your recipient test phone number>>
PHONE_NUMBER_ID=<<sender test phone number id>>
BUSINESS_ACCOUNT_ID=<<WhatsApp business account id>>
ACCESS_TOKEN=<<access token you generated for your System User>>
VERSION=v13.0
TEMPLATE_NAME_PREFIX=mkt
```

Creating Message Templates with Node.js and WhatsApp Business Platform

Now we're ready to start building your first message template.

Your form action in the
index.ejs
file located in the `views/` folder tells the app to POST to the `/createTemplates` route. Therefore, you need a new router to:

- Handle the createTemplates HTTP POST request.
- Verify if the templates already exist.
- Obtain the configuration needed to create each message template.
- Create the message templates using the Cloud API.
- Redirect the app to the homepage after creating the templates.

Create a new file called `createTemplates.js` in the `/routes` folder with the code below. This file contains the POST endpoint required to handle the request from the homepage and calls another function to generate the message templates to work with:

```
const express = require('express');
const router = express.Router();
const bodyParser = require('body-parser');
require('dotenv').config()
const { messageTemplates } = require("../public/javascripts/messageTemplates")
const { createMessageTemplate } = require("../messageHelper");
const { listTemplates } = require("../messageHelper");
```

```
  try {
      const templatesResponse = await listTemplates()
      const remoteApprovedTemplates =
        templatesResponse.data.data.filter(t => t.status === 'APPROVED');

      const localTemplates = messageTemplates

      for (let index = 0; index < localTemplates.length; index++) {
        const localTemplate = localTemplates[index];

        const queryResult = remoteApprovedTemplates
        .filter(t => t.name == process.env.TEMPLATE_NAME_PREFIX
          + '_' + localTemplate.name)

        console.log(`Creating template: ${localTemplate.name}.`)

        if (queryResult.length > 0) {
          console.log(`Template ${queryResult[0].name} already exists.`)
          continue;
        }

        try {
          await createMessageTemplate(localTemplate);
          console.log(`Template ${localTemplate.name} created successfully.`)
        } catch (error) {
          console.log(`Failed creating template ${localTemplate.name}.`)
          console.log(error.response.data);
        }

      }
    } catch (error) {
      console.log(“Failed obtaining remote template list.”)
      console.log(error);
    }

    console.log(“Redirecting to the backoffice.”)
    res.redirect('/backoffice');
});

module.exports = router;
```

Next, you need a function to encapsulate the Cloud API code to list and create message templates. Create a new file called `messageHelper.js` in the project’s root folder with the following code:

```
  return await axios({
    method: 'get',
    url: `https://graph.facebook.com/${process.env.VERSION}/${process.env.BUSI
      + '?limit=1000'
      + `&access_token=${process.env.ACCESS_TOKEN}`
  })
}
```

The code above sends a GET request to the `message_templates` endpoint to determine which templates already exist. The reply tells you whether you must create a new template.

Open the `messageHelper.js` file and add the following code to it:

```
async function createMessageTemplate(template) {

  const config = {
    method: 'post',
    url: `https://graph.facebook.com/${process.env.VERSION}/${process.env.BUSI
    headers: {
      'Authorization': `Bearer ${process.env.ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    data: {
      name:  process.env.TEMPLATE_NAME_PREFIX + '_' + template.name,
      category: "TRANSACTIONAL",
      components: template.components,
      language: template.language
    }
  };

  return await axios(config)
}

module.exports = {
  listTemplates: listTemplates,
  createMessageTemplate: createMessageTemplate
};
```

The code above accesses the Meta Graph API and makes an HTTP request to the `/message_templates` endpoint to list (HTTP GET) and create (HTTP

- Your WhatsApp Business account ID
- The access token you generated for your system user

Note that the `createMessageTemplate` function used here creates a template with a specific data structure whose parameters fit the needs of the example e-commerce app. You can learn more and experiment with creating different parameters by visiting the Message Template Guidelines page.

The `app.js` file defines the entry point for the Node.js app. In the sample application source code, the `app.js` file starts like this:

```
const createError = require('http-errors');
const express = require('express');
const path = require('path');
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const app = express();

const indexRouter = require('./routes/index');
const usersRouter = require('./routes/users');


// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};
});

module.exports = app;
```

You can see how the `app.js` file above declares the required modules. Then it registers the routes necessary for the Express.js framework to decide which JavaScript functions handle each HTTP request the web application processes.

Modify the `app.js` file to create a router variable for the `/createTemplate` route:

```
const createTemplatesRouter = require('./routes/createTemplates');
```

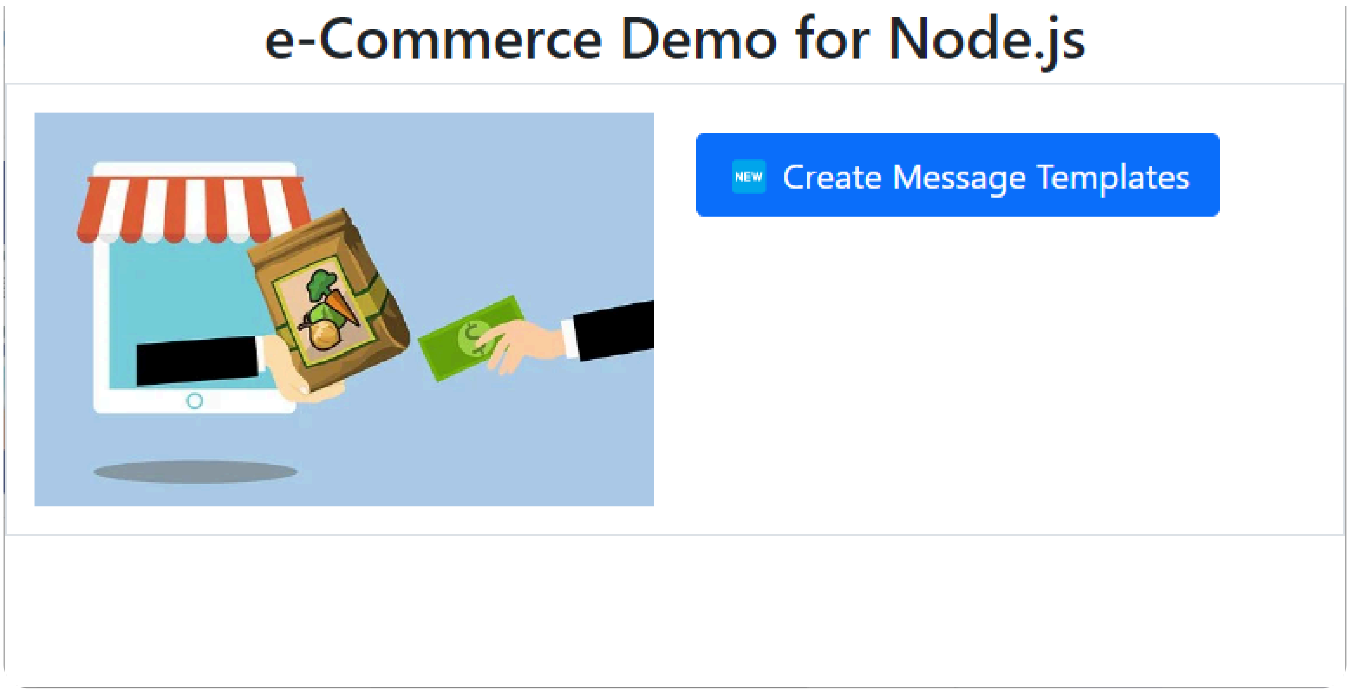
Then, allow the app to use the new `createTemplatesRouter` variable:

```
app.use('/createTemplates', createTemplatesRouter);
```

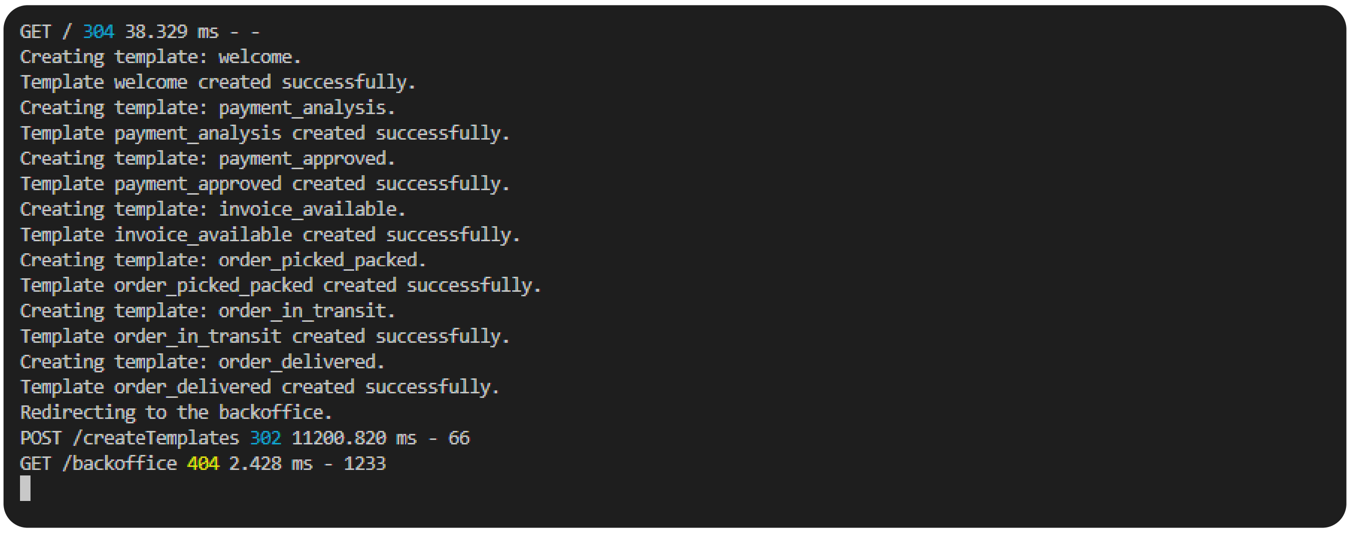
Now, to create the test message templates, create a new file called `messageTemplates.js` in the `public/javascripts/` folder using the contents of the
messageTemplates.js
file in the sample repository.

Finally, rerun the app:

```
> npm start
```



Click Create Message Templates. You see the creation of the message templates in the terminal window:



```
GET / 304 38.329 ms - -
Creating template: welcome.
Template welcome created successfully.
Creating template: payment_analysis.
Template payment_analysis created successfully.
Creating template: payment_approved.
Template payment_approved created successfully.
Creating template: invoice_available.
Template invoice_available created successfully.
Creating template: order_picked_packed.
Template order_picked_packed created successfully.
Creating template: order_in_transit.
Template order_in_transit created successfully.
Creating template: order_delivered.
Template order_delivered created successfully.
Redirecting to the backoffice.
POST /createTemplates 302 11200.820 ms - 66
GET /backoffice 404 2.428 ms - 1233
```

Now that you've created them all, you use them to send preformatted messages to WhatsApp users.

Creating the Back Office Page

The next step is creating mock-up data for the product catalog and the orders. You store the data in files separated from your JavaScript code.

and the
orders.js
files in the sample repository.

You now need a new route for users to access the Back Office page:

Create a new file called `backoffice.js` in the `/routes` folder with the following content to render the Back Office page with the orders data:

```
const express = require('express');
const { products } = require("../public/javascripts/products");
const { orders, orderActions, statuses } = require("../public/javascripts/orde
const { getMessageData, sendWhatsAppMessage } = require("../messageHelper")
const router = express.Router();

const viewModel = products;

router.post('/', async function (req, res, next) {
  const orderId = req.body.orderId;
  const order = orders.filter(o => o.id == orderId)[0]
  order.statusId++;

  const data = getMessageData(process.env.RECIPIENT_WAID, order);
  try {
    const response = await sendWhatsAppMessage(data)
    console.log(response);
  } catch (error) {
    console.log(error);
  }

  renderBackoffice(res);
});

router.get('/', function (req, res, next) {
  renderBackoffice(res);
});

module.exports = router;
function renderBackoffice(res) {
  res.render('backoffice',
  {
    title: 'e-Commerce Demo for Node.js',
    products: viewModel,
    statuses: statuses,
    orders: orders,
```

Create a new file called `backoffice.ejs` in the `/views` folder with the contents of the
backoffice.js
file in the sample repository to render the order data using the EJS template syntax.

Next, open the `app.js` file and create a router variable for the `/backoffice` route:

```
const backofficeRouter = require('./routes/backoffice');
```

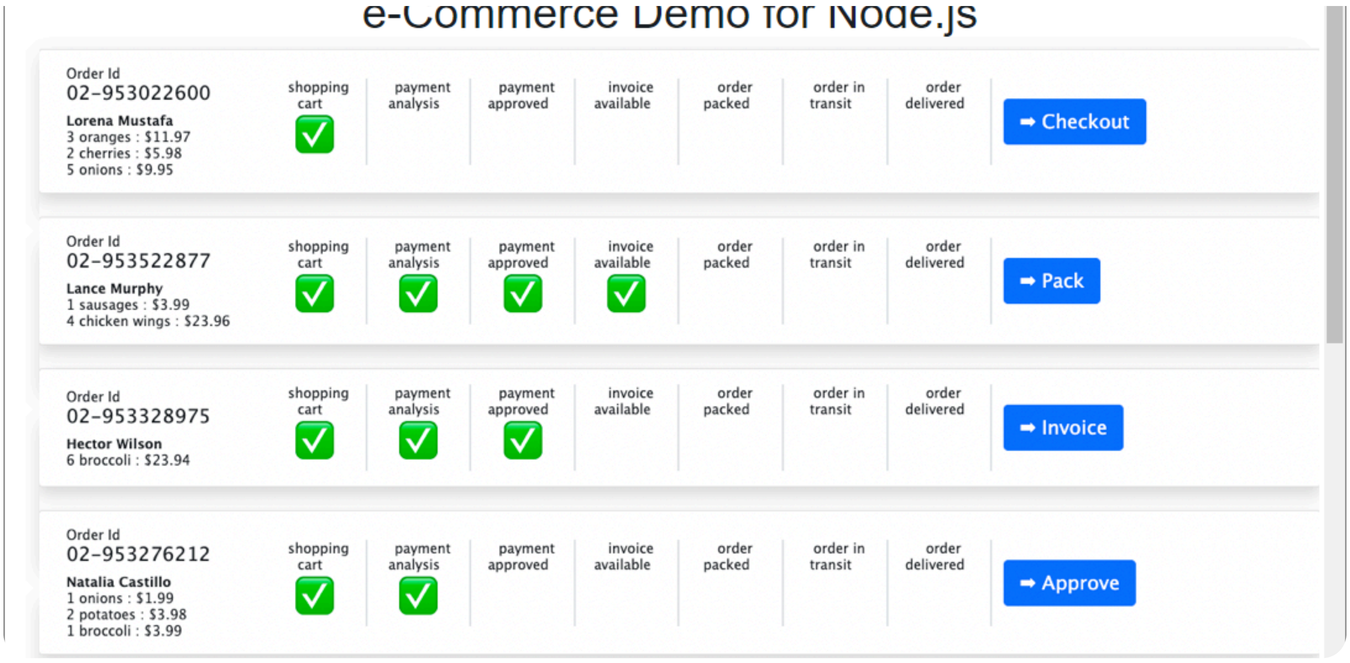
Allow the app to use the new `backofficeRouter` variable:

```
app.use('/backoffice', backofficeRouter);
```

Finally, rerun the app:

```
> npm start
```

Now, click Create Message Templates to create all the message templates with the Cloud API and redirect you to the Back Office view:



e-Commerce Demo for Node.js

Order Id	shopping cart	payment analysis	payment approved	invoice available	order packed	order in transit	order delivered	
Order Id 02-953022600 **Lorena Mustafa** 3 oranges : $11.97 2 cherries : $5.98 5 onions : $9.95	✓							➡ Checkout
Order Id 02-953522877 **Lance Murphy** 1 sausages : $3.99 4 chicken wings : $23.96	✓	✓	✓	✓				➡ Pack
Order Id 02-953328975 **Hector Wilson** 6 broccoli : $23.94	✓	✓	✓					➡ Invoice
Order Id 02-953276212 **Natalia Castillo** 1 onions : $1.99 2 potatoes : $3.98 1 broccoli : $3.99	✓	✓						➡ Approve

Sending Templated Messages

A business-initiated conversation requires an approved message template during production and at minimum, be marked active during development. These conversations could include customer care messages, appointment reminders, payment or shipping updates, and alerts.

You must send the message using the parameters required by each template.

Open the
messageHelper.js
file and delete the `module.exports` block. Next, add the `sendWhatsAppMessage` and the `getMessageData` functions. Then, import the required javascript files for the products and message templates. Finally, add the new `module.exports` block using the code below:

```
const { messageTemplates } = require('./public/javascripts/messageTemplates')
const { products } = require('./public/javascripts/products')

async function sendWhatsAppMessage(data) {
  const config = {
    method: 'post',
    url: `https://graph.facebook.com/${process.env.VERSION}/${process.env.PHON
    headers: {
      'Authorization': `Bearer ${process.env.ACCESS_TOKEN}`,
  };

  return await axios(config)
}

function getMessageData(recipient, order) {

  const messageTemplate = messageTemplates[order.statusId - 1]

  let messageParameters

  switch (messageTemplate.name) {
    case 'welcome':
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
      ];
      break;
    case 'payment_analysis':
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
        { type: "text", text: products[order.items[0].productId - 1].name },
      ];
      break;
    case 'payment_approved':
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
        { type: "text", text: order.id },
        { type: "text", text: order.deliveryDate },
      ];
      break;
    case 'invoice_available':
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
        { type: "text", text: products[order.items[0].productId - 1].name },
        { type: "text", text: `https://customer.your-awesome-grocery-store-dem
      ];
      break;
    case 'order_picked_packed':
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
        { type: "text", text: order.id },
        { type: "text", text: `https://customer.your-awesome-grocery-store-dem
      ];
      break;
    case 'order_in_transit':
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
        { type: "text", text: order.id },
        { type: "text", text: order.deliveryDate },
      messageParameters = [
        { type: "text", text: order.customer.split(' ')[0] },
        { type: "text", text: order.id },
        { type: "text", text: order.deadlineDays },
      ];
      break;
  }

  const messageData = {
    messaging_product: "whatsapp",
    to: recipient,
    type: "template",
    template: {
      name: process.env.TEMPLATE_NAME_PREFIX + '_' + messageTemplate.name,
      language: { "code": "en_US" },
      components: [{
          type: "body",
          parameters: messageParameters
        }]}}

  return JSON.stringify(messageData);
}

module.exports = {
  sendWhatsAppMessage: sendWhatsAppMessage,
  listTemplates: listTemplates,
  createMessageTemplate: createMessageTemplate,
  getMessageData: getMessageData
};
```

Note that the `getMessageData` function returns a data structure required for sending messages formatted for your back-office templates. You can experiment with other templates or create new ones by visiting the Message Templates page.

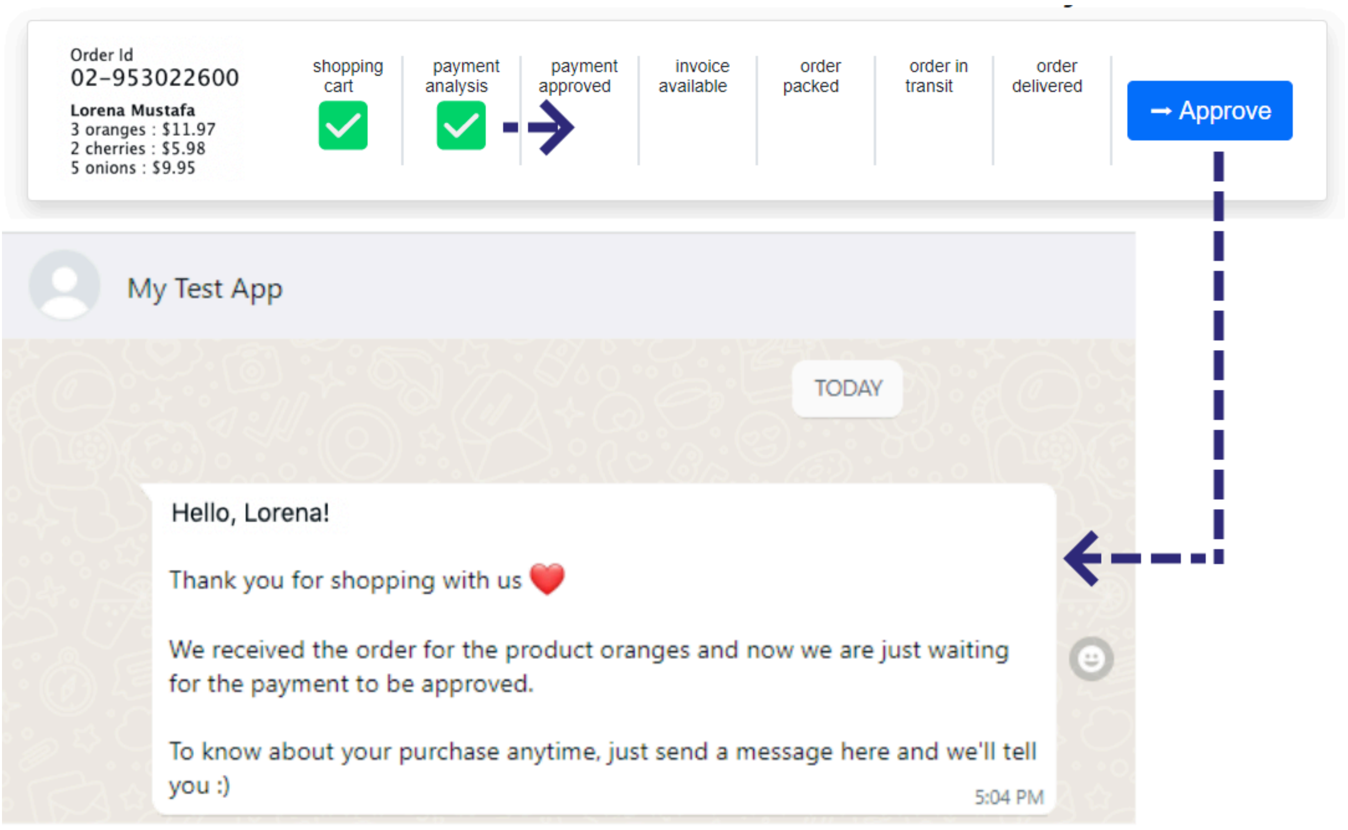
Finally, run the app:

```
> npm start
```

On the Back Office page, click each Action button to move the orders along the sales funnel. Your app sends the appropriate templated WhatsApp

Pick the first order and click the Approve button. Your app makes a POST request to the `/messages` endpoint of the Cloud API. It then sends a message to the customer according to the `payment_analysis template` created by your app. The app then moves the order status to Payment Approved.

Now open WhatsApp to visualize the templated message:



Order Id
02-953022600
Lorena Mustafa
3 oranges : $11.97
2 cherries : $5.98
5 onions : $9.95

Hello, Lorena!

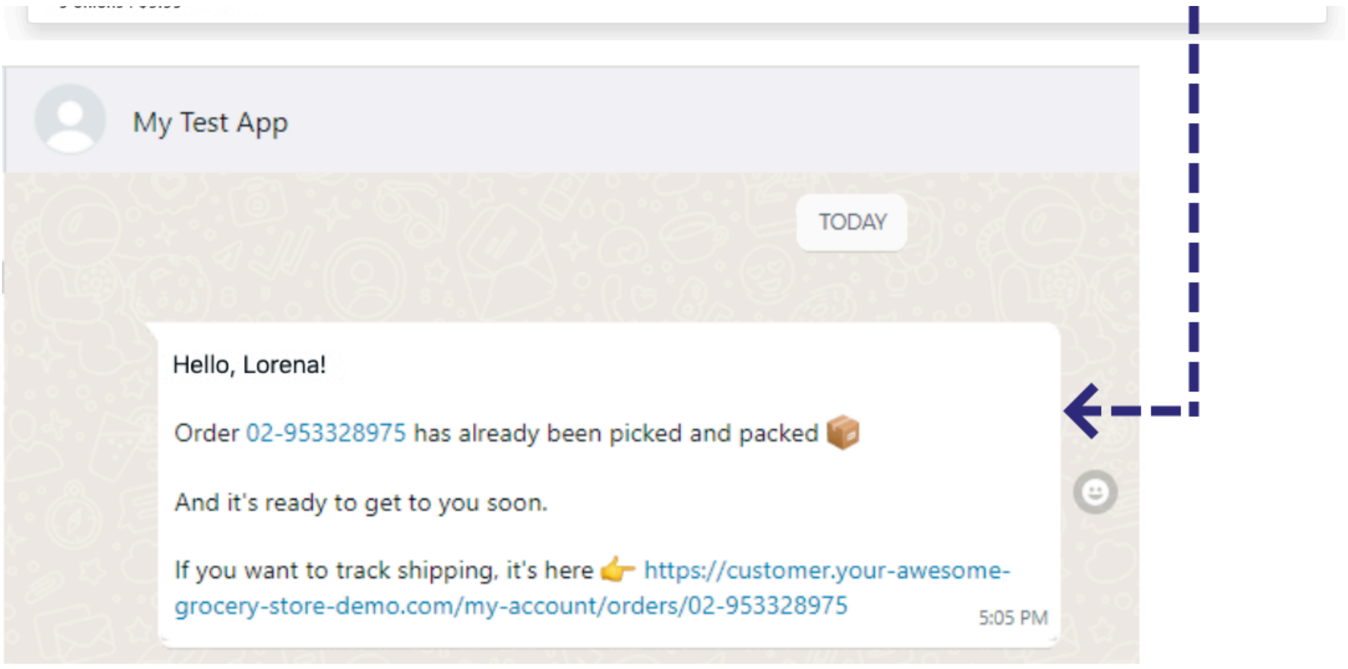
Thank you for shopping with us ❤️

We received the order for the product oranges and now we are just waiting for the payment to be approved.

To know about your purchase anytime, just send a message here and we'll tell you :)

5:04 PM

Next, pick the second order, click Ship, and repeat the process above:


My Test App

TODAY

Hello, Lorena!

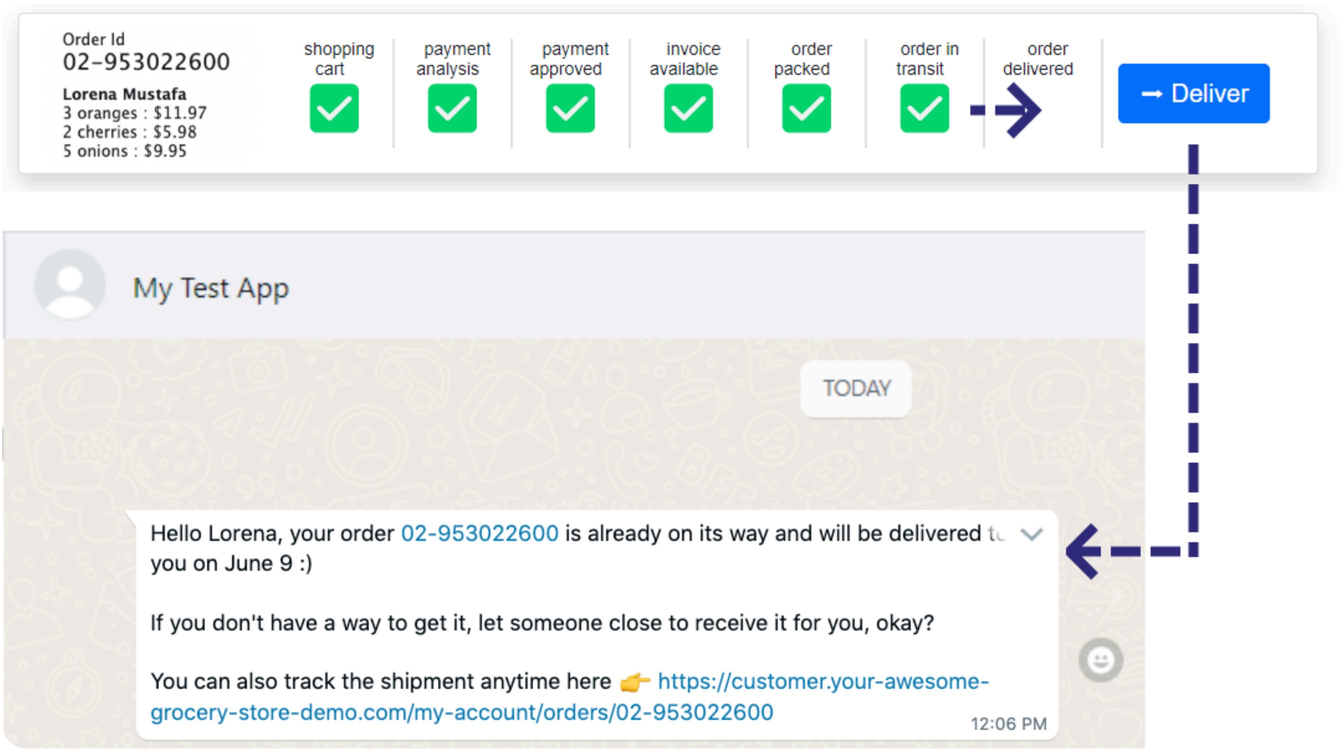
Order 02-953328975 has already been picked and packed 📦

And it's ready to get to you soon.

If you want to track shipping, it's here 👉 https://customer.your-awesome-grocery-store-demo.com/my-account/orders/02-953328975

5:05 PM

Finally, pick the fifth order and click Deliver:


Order Id
02-953022600
Lorena Mustafa
3 oranges : $11.97
2 cherries : $5.98
5 onions : $9.95

shopping cart	payment analysis	payment approved	invoice available	order packed	order in transit	order delivered
✓	✓	✓	✓	✓	✓	→

My Test App

TODAY

Hello Lorena, your order 02-953022600 is already on its way and will be delivered to you on June 9 :)

If you don't have a way to get it, let someone close to receive it for you, okay?

You can also track the shipment anytime here 👉 https://customer.your-awesome-grocery-store-demo.com/my-account/orders/02-953022600

12:06 PM

The WhatsApp messages above have two essential elements: the message templates representing the current purchase order state in the e-commerce funnel and the raw data containing information about the customer or products.

integration would also allow you to automate this process based on customer updates in those systems.

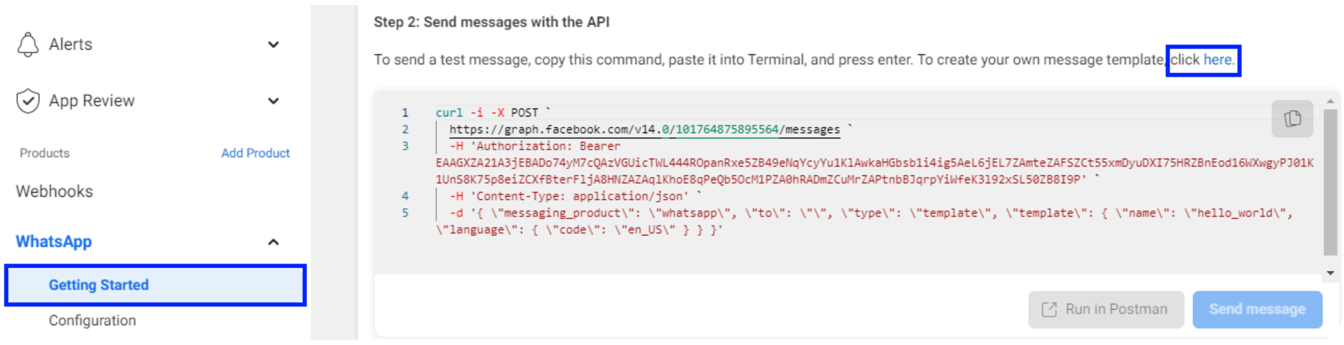
Creating and Managing Message Templates Using the Meta Business Manager UI

You have seen how the Cloud API allows your application to create message templates programmatically. In addition to this powerful feature, you can count on the Meta Business Manager UI to create and modify templates directly through the web without needing a program.

To reach this UI, login to the Meta for Developers Platform, then navigate to your app. On the left navigation panel, under Products, click WhatsApp and then click Getting Started.

Under Step 2: Send messages with the API, you see this paragraph with a link:

"To send a test message, copy this command, paste it into Terminal, and press enter. To create your own message template, click here."

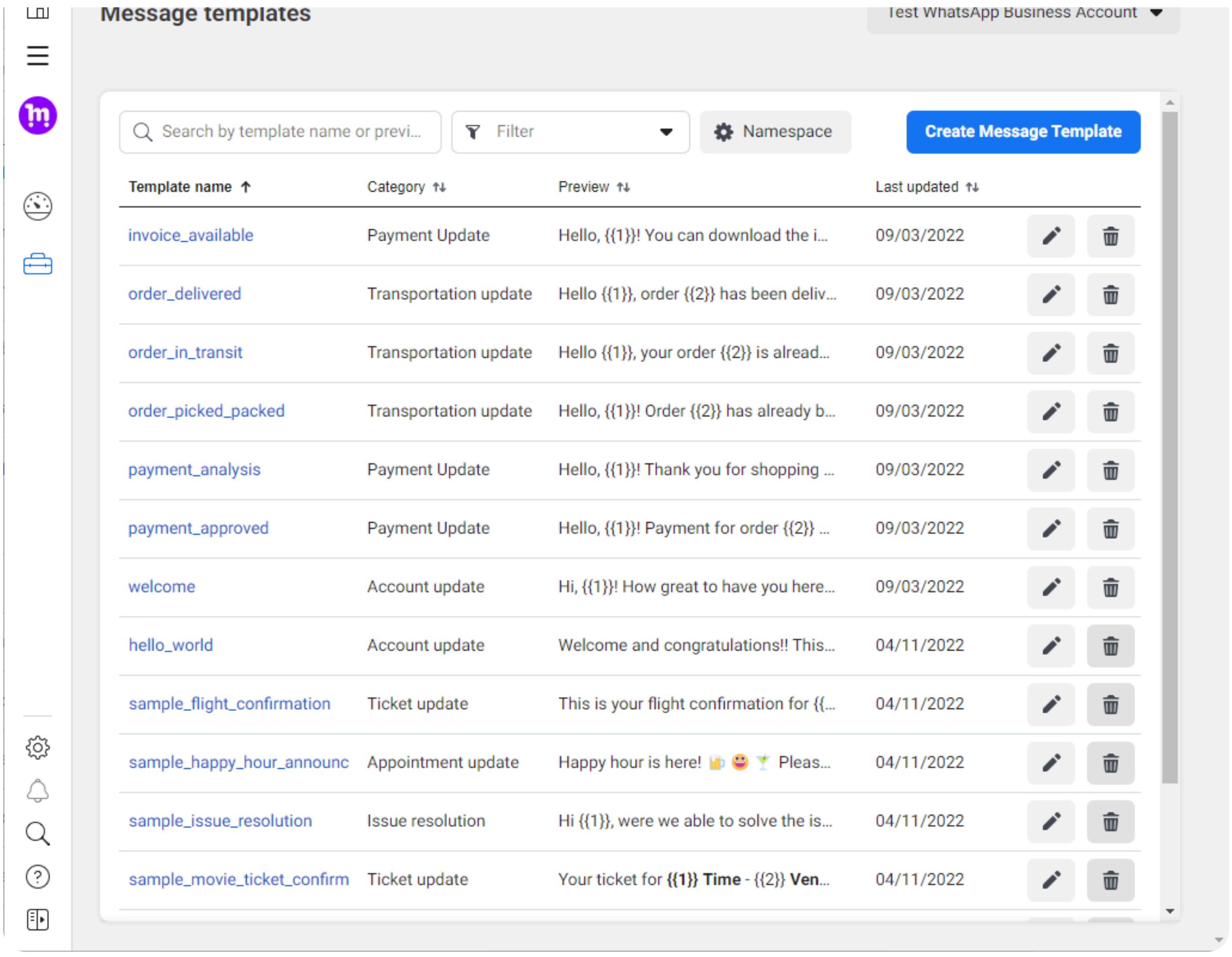


Step 2: Send messages with the API

To send a test message, copy this command, paste it into Terminal, and press enter. To create your own message template, click here.

```
curl -i -X POST `
  https://graph.facebook.com/v14.0/101764875895564/messages `
  -H 'Authorization: Bearer
EAAGXZA21A3jEBADo74yM7cQAzVGUicTWL444ROpanRxe5ZB49eNqYcyYu1KlAwkaHGbsb1i4ig5AeL6jEL7ZAmteZAFSZCt55xmDyuDXI75HRZBnEod16WXwgyPJ01K
1UnS8K75p8eiZCXfBterFljA8HNZAZAqlKhoE8qPeQb5OcM1PZA0hRADmZCuMrZAPtnbBJqrpYiWfeK3192xSL50ZB8I9P' `
  -H 'Content-Type: application/json' `
  -d '{ \"messaging_product\": \"whatsapp\", \"to\": \"\", \"type\": \"template\", \"template\": { \"name\": \"hello_world\",
\"language\": { \"code\": \"en_US\" } } }'
```

Click the link. Alternatively, you can go directly to the Message Templates page.



Message templates

Test WhatsApp Business Account

Search by template name or previ...

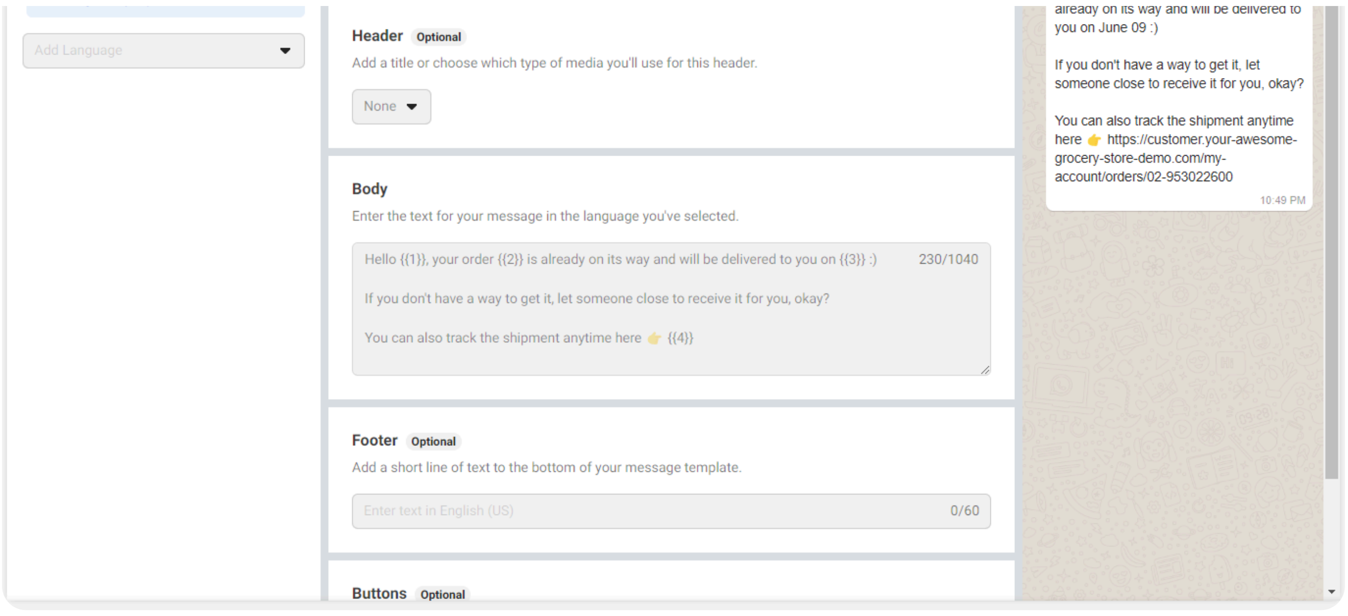
Filter

Namespace

Create Message Template

Template name	Category	Preview	Last updated
invoice_available	Payment Update	Hello, {{1}}! You can download the i...	09/03/2022
order_delivered	Transportation update	Hello {{1}}, order {{2}} has been deliv...	09/03/2022
order_in_transit	Transportation update	Hello {{1}}, your order {{2}} is alread...	09/03/2022
order_picked_packed	Transportation update	Hello, {{1}}! Order {{2}} has already b...	09/03/2022
payment_analysis	Payment Update	Hello, {{1}}! Thank you for shopping ...	09/03/2022
payment_approved	Payment Update	Hello, {{1}}! Payment for order {{2}} ...	09/03/2022
welcome	Account update	Hi, {{1}}! How great to have you here...	09/03/2022
hello_world	Account update	Welcome and congratulations!! This...	04/11/2022
sample_flight_confirmation	Ticket update	This is your flight confirmation for {{...	04/11/2022
sample_happy_hour_announc	Appointment update	Happy hour is here! 🍺 😀 🍸 Pleas...	04/11/2022
sample_issue_resolution	Issue resolution	Hi {{1}}, were we able to solve the is...	04/11/2022
sample_movie_ticket_confirm	Ticket update	Your ticket for **{{1}} Time** - {{2}} **Ven**...	04/11/2022

To view one of the messages you created with the Node.js application, click the pencil icon, and you will see:


Add Language

Header Optional

Add a title or choose which type of media you'll use for this header.

None

Body

Enter the text for your message in the language you've selected.

Hello {{1}}, your order {{2}} is already on its way and will be delivered to you on {{3}} :) 230/1040

If you don't have a way to get it, let someone close to receive it for you, okay?

You can also track the shipment anytime here 👉 {{4}}

Footer Optional

Add a short line of text to the bottom of your message template.

Enter text in English (US) 0/60

Buttons Optional

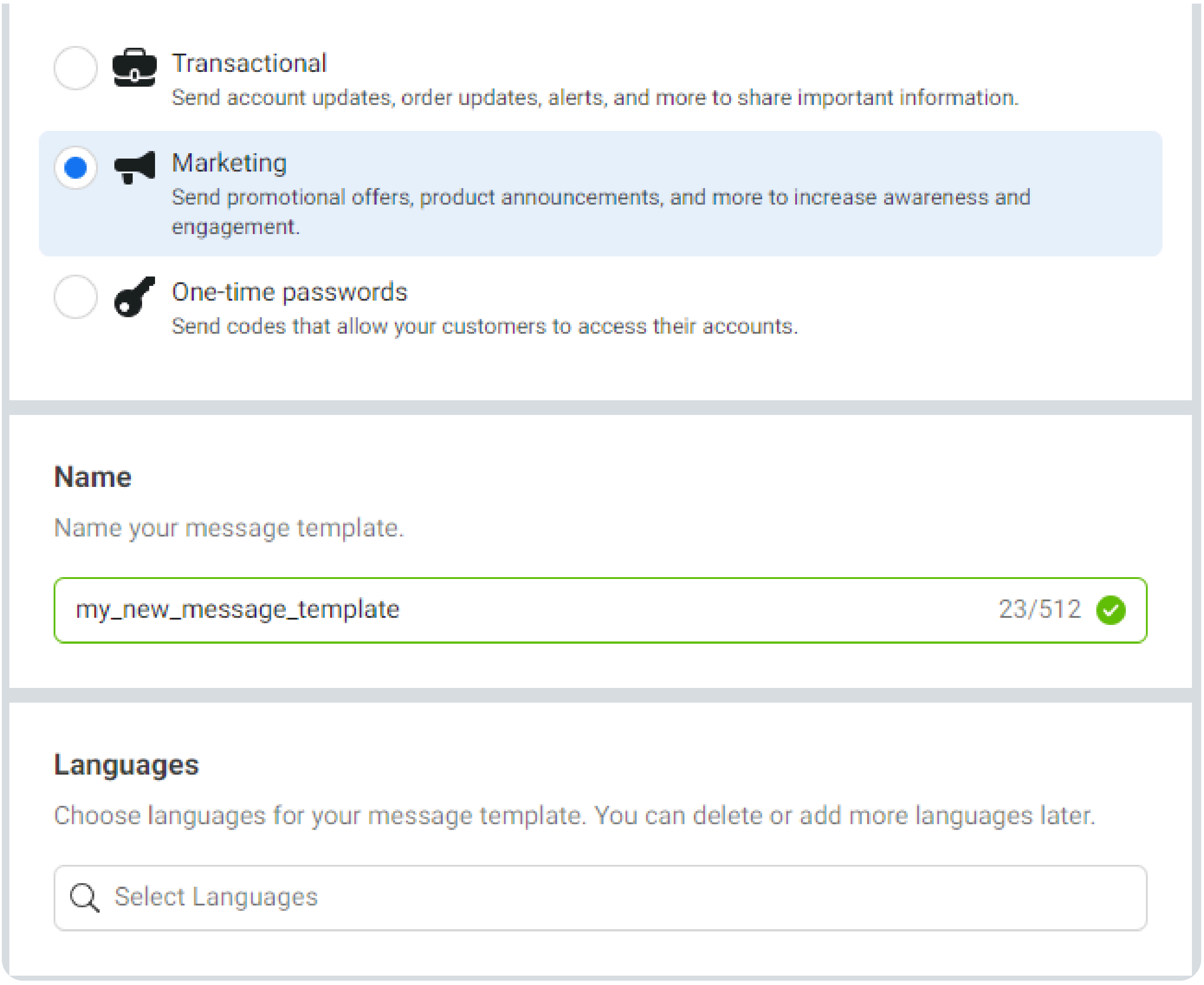
already on its way and will be delivered to you on June 09 :)

If you don't have a way to get it, let someone close to receive it for you, okay?

You can also track the shipment anytime here 👉 https://customer.your-awesome-grocery-store-demo.com/my-account/orders/02-953022600

10:49 PM

To create a new template, click the Create Message Template button and provide a category, a name, and one or more languages for your template:



Transactional
Send account updates, order updates, alerts, and more to share important information.

[✓ Marketing]
Send promotional offers, product announcements, and more to increase awareness and engagement.

One-time passwords
Send codes that allow your customers to access their accounts.

Name
Name your message template.

my_new_message_template 23/512 ✓

Languages
Choose languages for your message template. You can delete or add more languages later.

Finally, type some text for your message template. The message body may contain variables, which are placeholders for data your app must provide later when it sends messages to WhatsApp users, just as in this article. To customize your templates further, you can include a message footer and a header with an image, video, or document.

Conclusion

The WhatsApp Business Platform is a simple tool to facilitate communication between companies and their customers. Using the Cloud API, hosted by Meta, you can easily connect your app, communicate efficiently with your customers, and promote your business.

the Cloud API.

Ready for more? See our tutorial on building and sending media-rich messages with your app and the Cloud API.

TAGS Business Tools Developer Tools Platforms 2022 WhatsApp

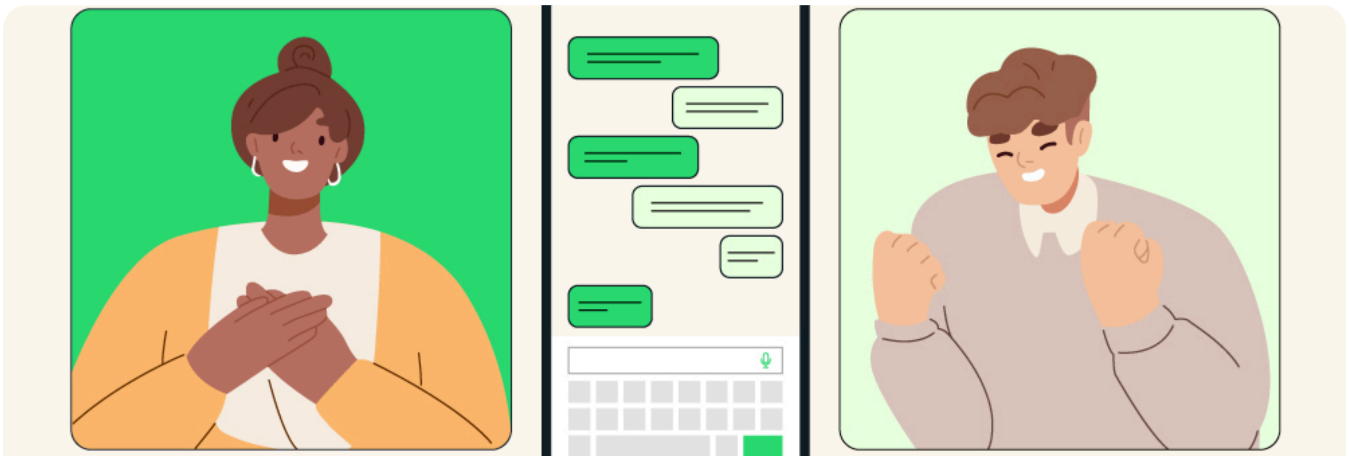
Explore more

December 19, 2022

How to Visualize WhatsApp Account Metrics in Your App

December 12, 2022

Using QR Codes and Short Links in WhatsApp



November 21, 2022

Send Interactive Messages on the WhatsApp Business Platform

Get our newsletter

Sign up for monthly updates
from Meta for Developers.

Meta