TUTORIAL

Sending Scheduled and Recurring Email Notifications with PHP

Sarah Barber

July 12, 2023

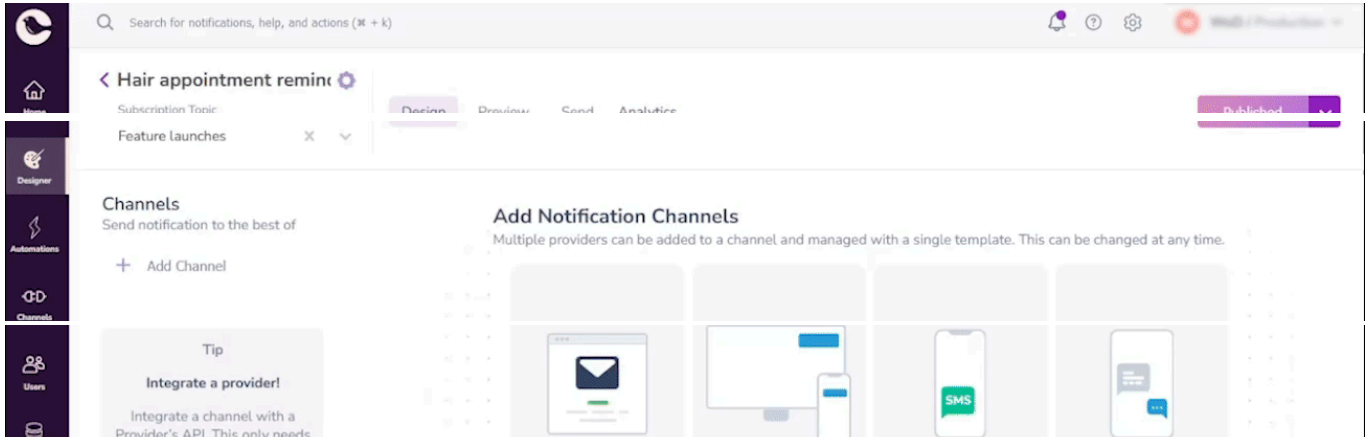
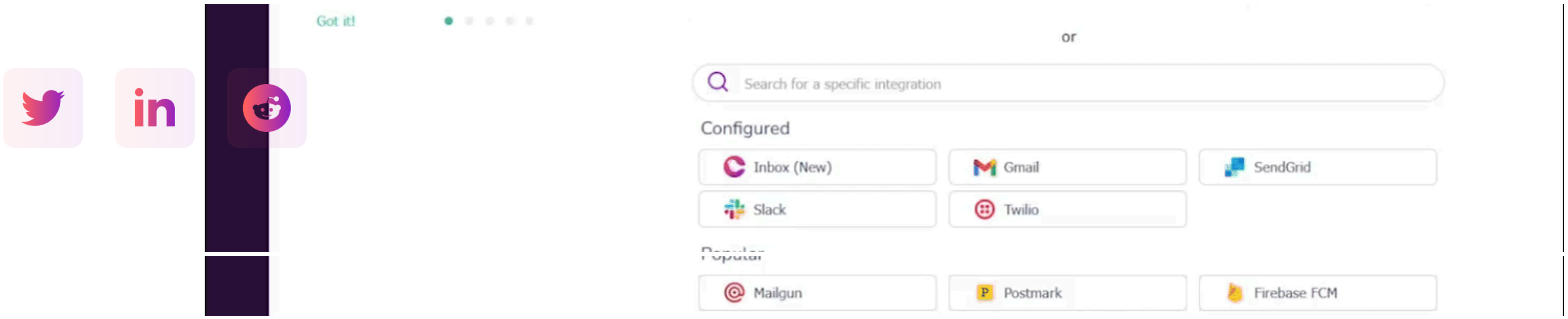


Table of contents



Sending timely targeted email notifications greatly affects how your audience engages with your product. For these notifications to be effective at notifying your users of key events, you need to schedule when they are sent so that they are delivered at the right time.

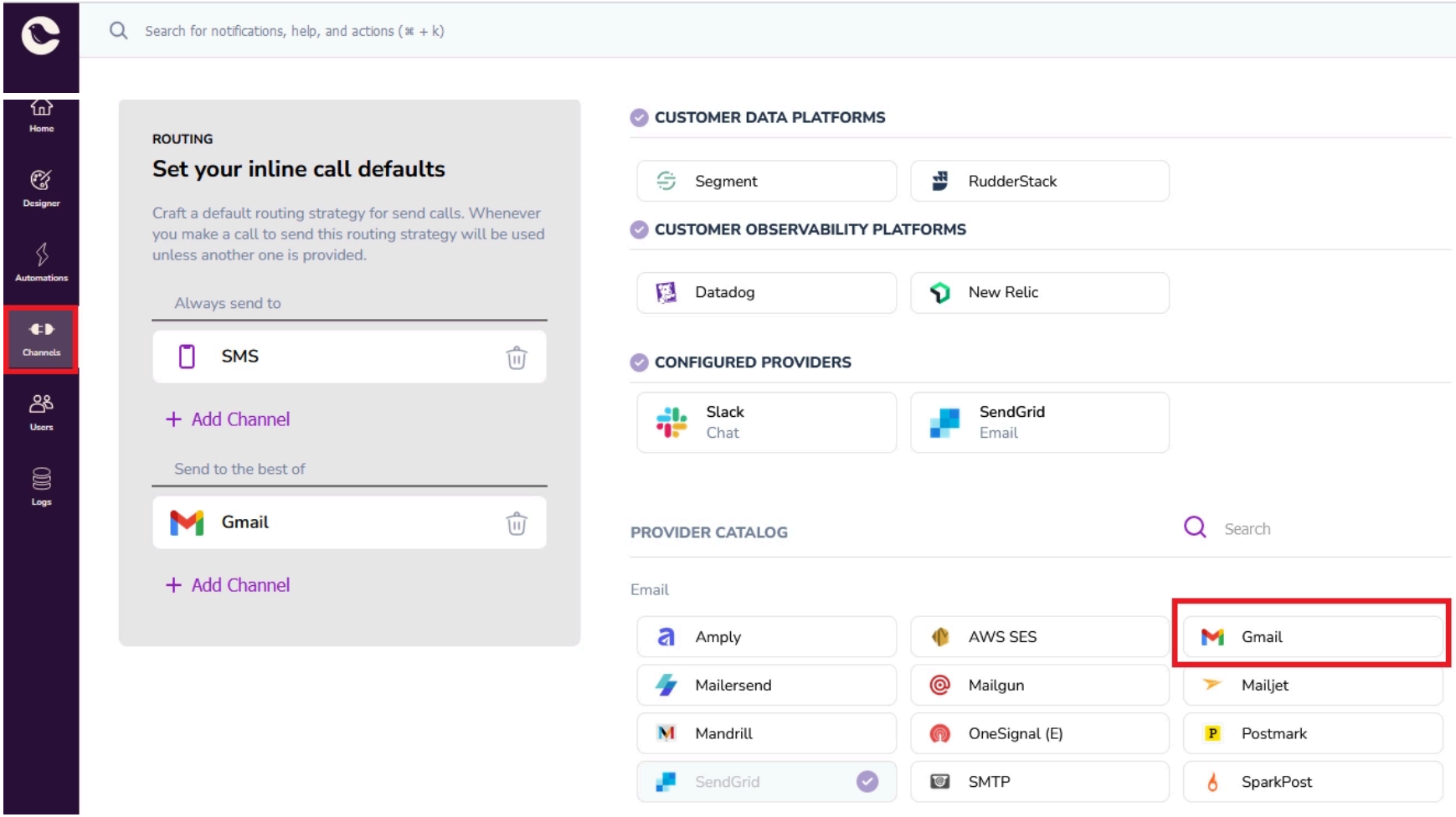
Scheduled and recurring notifications are in use everywhere — for example, online stores use scheduled notifications to inform users about sale events (like Black Friday), and doctors, dentists, and tradespeople have systems that send appointment reminders. Recurring emails are commonly used for subscription services to email monthly bills to customers.

This tutorial covers two different ways for PHP developers to send scheduled and recurring email notifications through the Courier notification platform using its PHP SDK. It also offers a low-code solution for sending scheduled emails using just the Courier UI. Courier is a multi-channel notification service with a robust API, which you can use to build a production-ready email notification system in a few minutes.

Configure an email service provider in Courier

Create a Courier account if you don't already have one, so that you can configure an email service provider in Courier, allowing it to send emails on your behalf. \

In the Courier app, navigate to **Channels** and choose your email service provider. For this tutorial, we will use Gmail.

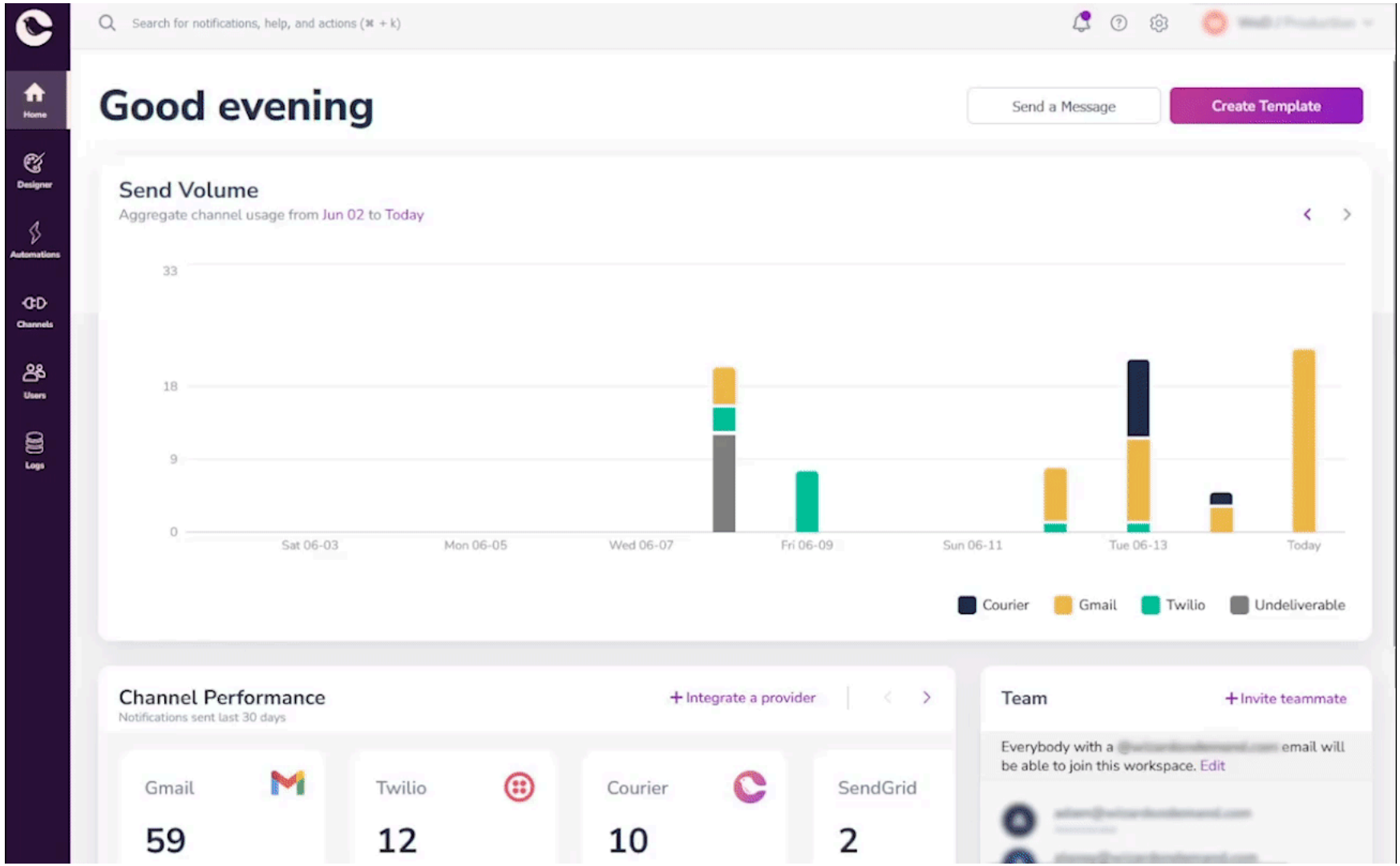


On the next screen, select **Sign in with Google** to give Courier permission to access your Gmail account.

Create a notification template in Courier

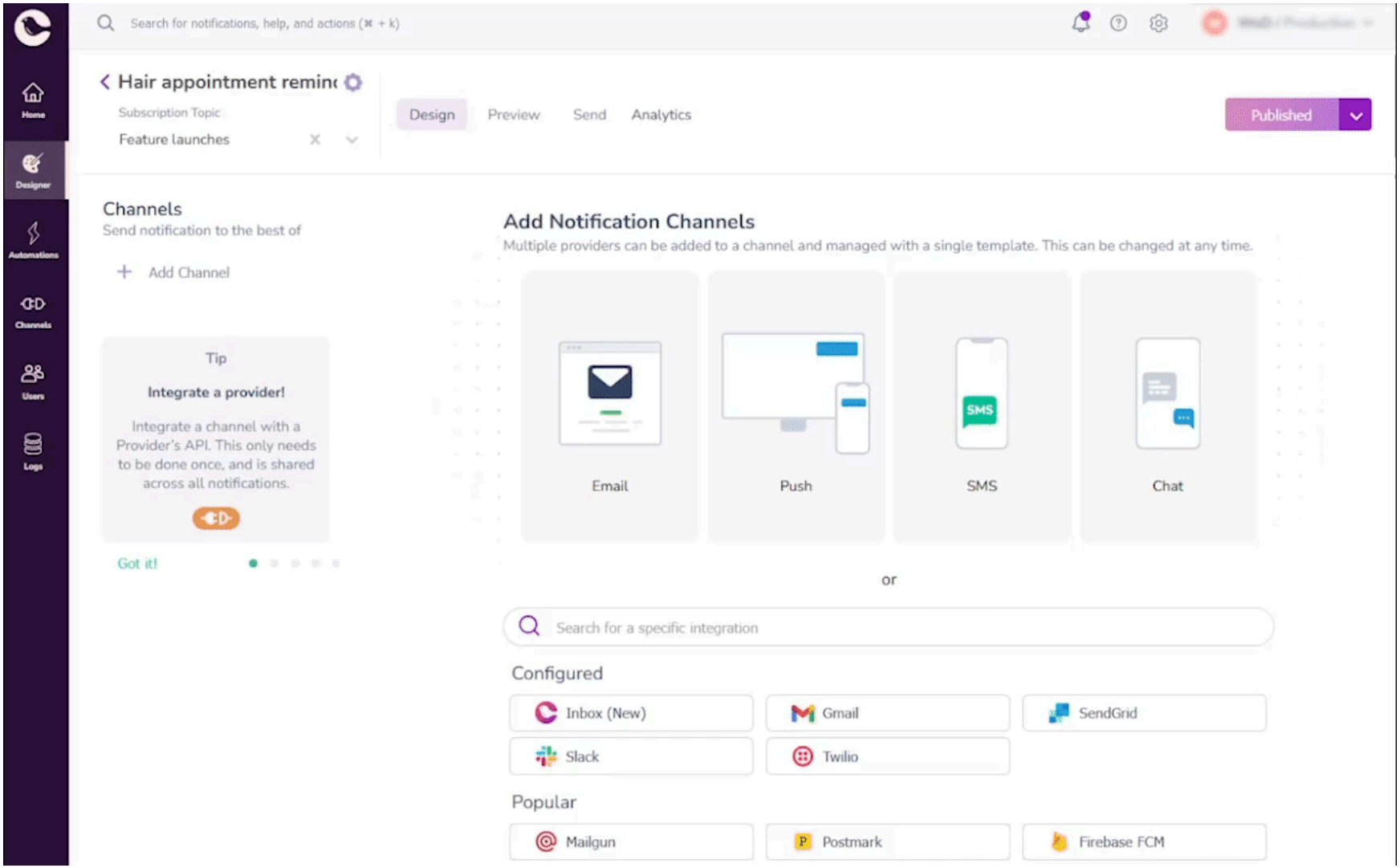
Courier uses email templates to make it easy to reuse your emails. In this tutorial, we will use an example of a hairdressing business that sends emails to its clients before (and sometimes after) appointments.

To create your first template, start by navigating to the Designer. Click **Create Template**, give it the name **Hair appointment reminder**, and click **Create Template** again.



Next, select **Email** as the channel for this notification (choosing Gmail as the specific provider in the drop-down box). Now, click on your new email channel on the left side to see the no-code editor for designing

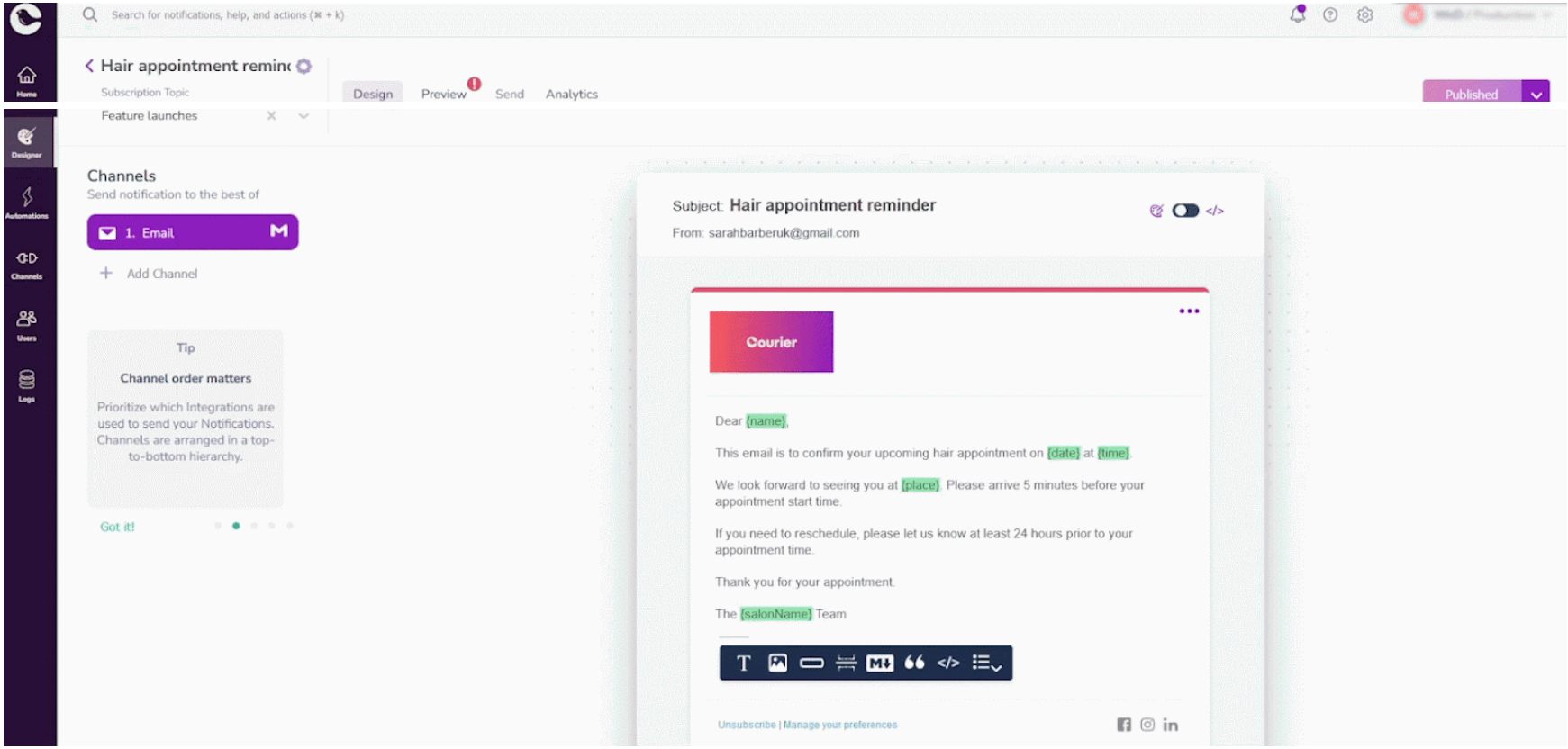
your notification.



Give your notification the subject “Hair appointment reminder”, and paste the following content into the message body:

```
Dear {name},

This email is to confirm your upcoming hair appointment tomorrow.

We look forward to seeing you at {place}. Please arrive five minutes before your appointment start time.

If you need to reschedule, please let us know at least 24 hours prior to your appointment time.

Thank you for your appointment.

The {salonName} Team
```

The curly braces in this template are variables that will later be passed to the template as data. Now that your template is complete, click **Publish Changes**. Later on, you will need your notification template ID and Courier authorization token when you want to call the Courier API. These are both available in your notification template settings. Navigate there now and copy both so that you can use them later.



Now you have three options:

1. **Directly send scheduled or recurring emails using Courier:** Call the `sendEnhancedNotification()` function from the Courier PHP SDK, and use a third party task scheduling library called Crunz to deal with the scheduling side of things. This works using cron syntax, so the same principle can be used for scheduled or recurring emails.
2. **Use Courier's automations to add send logic to your scheduled emails:** An automation in Courier is a way of chaining together different steps such as the sending of emails (or other notification-related logic) so that the steps happen in a particular order. An automation can be run by calling the `invokeAutomation()` function, and as with option 1, you can use Crunz to deal with the scheduling.

3. **Using Courier's no-code automations designer:** This is a no-code GUI tool in the Courier UI that uses a drag-and-drop canvas to build up your notification logic. It contains some more advanced logic than option 2 (such as the ability to create email digests or batching).

Option 1: directly send a scheduled or recurring email using Courier

Preparing your PHP environment

For both of the PHP examples shown below, you will need to prepare a PHP environment and create a notification template in the Courier app.

1. Install the latest versions of PHP and Composer.
2. Install the following packages using Composer:
   - trycourier/courier — the Courier PHP SDK
   - guzzlehttp/guzzle — required by the Courier PHP SDK
   - crunzphp/crunz — for scheduling one-time and recurring tasks
   - vlucas/phpdotenv — for handling environment variables

You can install these by running the following command:

```
composer require crunzphp/crunz vlucas/phpdotenv trycourier/courier guzzlehttp/guzzle
```

You can also find the code shown in this tutorial in our working example repository on GitHub. If you are cloning the repository, you will need to run the `composer install` command to install the dependencies.

Sending a scheduled email

To manage the scheduling side of things, we will use Crunz — a PHP package that allows you to run scheduled tasks within your PHP code without having to create a cron job for each of them. We will wrap this around Courier's `sendEnhancedNotification()` function, which is used for sending emails.

For this example, we have specified that the hair appointment reminder should be sent at a specific time (13:30 2023-07-01). However, note that Crunz's `on()` function accepts any date format parsed by PHP's `strtotime()` function, so it's easy to pass in a time value derived from one of your variables.

As per the example in our GitHub repository, you must provide your configuration in an `.env` file at the project root. You can create this file by copying the provided example file into your project directory:

```
cp .env.example .env
```

Then, fill out the contents of the newly copied `.env` file:

COURIER_AUTHORIZATION_TOKEN	COURIER API KEY, FOUND IN SETTINGS
TEST_EMAIL_TO	Use your own email address as the value

COURIER_AUTHORIZATION_TOKEN	COURIER API KEY, FOUND IN SETTINGS
TEST_DIRECT_SEND_NOTIFICATION_TEMPLATE_ID	The notification template ID for the "Hair appointment reminder" email template

Crunz requires you to add a single crontab entry that runs every minute. Whenever this job runs, any tasks scheduled using Crunz for that time will be executed. You only need to do this once for your project. Append the following line to your own user's crontab on your PHP server (you can do this by typing `crontab -e` in your terminal):

```
* * * * * cd /path/to/project && vendor/bin/crunz schedule:run
```

Replace `/path/to/project` with the absolute path to your PHP project. For security reasons, you should not add this line to your root user's crontab but instead create a user who has execute permissions for the project directory or is a member of the group that your web server runs under (by default, `www-data` on most Linux systems).

Crunz requires the presence of a configuration file, which contains a configured timezone. Create this by running this command in your project directory:

```
vendor/bin/crunz publish:config
```

Note that the default configured timezone is UTC, but you can change this in the config if needed.

All Crunz tasks must be contained in a directory called `tasks` at the root level of your project. The file containing each task should end with `Tasks.php`. Create this directory, and inside it, create a file called `scheduledSendTasks.php` and paste the following code into it:

```php
<?php

use Crunz\Schedule;
use Courier\CourierClient;
use Dotenv\Dotenv;

// Configure environment variables - set the .env directory to the parent of the Tasks directory
// Environment variables are stored in a variable so that they can be passed to the scheduled task, which will no
$dotenv = Dotenv::createArrayBacked(__DIR__ . "/..")->load();

// Configure scheduler
$schedule = new Schedule();

// Configure the Courier PHP SDK - note the first null is the API path, of which we will use the default
$courier = new CourierClient(null, $dotenv['COURIER_AUTHORIZATION_TOKEN']);

// Create a new scheduled task
$task = $schedule->run(function () use ($courier, $dotenv) {

    echo "Running " . __FILE__ . "\n";

    // Send notification using the Courier PHP SDK
    $notification = (object) [
```

```
            "to" => [
                "email" => $dotenv['TEST_EMAIL_TO']
            ],
            "template" => $dotenv['TEST_DIRECT_SEND_NOTIFICATION_TEMPLATE_ID'],
            "routing" => [
                "method" => "single",
                "channels" => ["email"]
            ],
            "data" => [
                "name" => "John Doe",
                "place" => "123 High Street",
               "salonName" => "Cutting Edge"
            ]
        ];
        $result = $courier->sendEnhancedNotification($notification);
    });

    // Schedule a single notification for a specific time
    $task->on('13:30 2023-07-01')
        ->description('Sending scheduled email');

    return $schedule;
```

Now that you've created a scheduled task, it will run as soon as the scheduled time is reached. Remember, as mentioned above, the default timezone is UTC, but you can change this.

If you want to test your scheduled task, you can force it to run immediately using this command:

```
vendor/bin/crunz schedule:run --force
```

The above command will run all scheduled or recurring tasks that you've created. If you want to be sure which tasks will run with this command, you can run another command to check how many scheduled tasks you have:

```
vendor/bin/crunz schedule:list
```

This will output a table containing your scheduled tasks:

```
+---+------------------------------+-------------+-----------------+
| # | Task                         | Expression  | Command to Run  |
+---+------------------------------+-------------+-----------------+
| 1 | Sending scheduled email      | 30 13 1 7 * | object(Closure) |
| 2 | Sending recurring email      | 30 13 * * * | object(Closure) |
| 3 | Sending scheduled automation | 30 13 1 7 * | object(Closure) |
+---+------------------------------+-------------+-----------------+
```

Sending a recurring email

You can also use Crunz to create recurring tasks on a schedule.

Inside your `tasks` directory, create a PHP file called `recurringSendTasks.php` and paste this code into it:

```php
<?php

use Crunz\Schedule;
use Courier\CourierClient;
use Dotenv\Dotenv;

// Configure environment variables - set the .env directory to the parent of the Tasks directory
// Environment variables are stored in a variable so that they can be passed to the scheduled task, which will no
$dotenv = Dotenv::createArrayBacked(__DIR__ . "/..")->load();

// Configure scheduler
$schedule = new Schedule();

// Configure the Courier PHP SDK - note the first null is the API path, of which we will use the default
$courier = new CourierClient(null, $dotenv['COURIER_AUTHORIZATION_TOKEN']);

// Create a new scheduled task
$task = $schedule->run(function () use ($courier, $dotenv) {

    echo "Running " . __FILE__ . "\n";

    // Send notification using the Courier PHP SDK
    $notification = (object) [
        "to" => [
            "email" => $dotenv['TEST_EMAIL_TO']
        ],
        "template" => $dotenv['TEST_DIRECT_SEND_NOTIFICATION_TEMPLATE_ID'],
        "routing" => [
            "method" => "single",
            "channels" => ["email"]
```

```
            ],
            "data" => [
                "name" => "John Doe",
                "place" => "123 High Street",
                "salonName" => "Cutting Edge"
            ]
        ];
        $result = $courier->sendEnhancedNotification($notification);
    });

    // Set up a recurring task
    $task
        ->daily()
        ->at('13:30')
        ->description('Sending recurring email');

    return $schedule;
```

To test this, again run `vendor/bin/crunz schedule:run --force`. However, you'll only receive one email, as you are using the `--force` option, which forces a single run of each task. When your recurring task is being invoked by Crunz as a scheduled task, it will be called at the specified interval, and if you use the above code example, you will receive one email per day at 13:30.

Using Laravel? It's already got scheduling baked in

If you're using Laravel, you don't need to worry about setting up your own scheduling solution, as it already has its own(https://laravel.com/docs/10.x/scheduling) (and queueing!) built in — one of the many advantages of using a PHP framework.

Option 2: use Courier's automations to add send logic to your scheduled emails

Sometimes you need to add some logic around the sending of your scheduled or recurring emails, and this is where Courier's automations can be useful. Automations allow you to chain together a series of different steps, including the sending of emails based on different notification templates.

Examples of when to use automations

Imagine a gym that sends its customers workout tips — it may want to sometimes send out different workout plans based on customers' age or other groupings. Using automation in Courier, it could implement some basic branching logic such as "if the customer's age is greater than 50, send an email using the over 50s email template; otherwise, send using the under 50s email template."

Another feature that Courier automations offer is the ability to add a delay between two different actions. To reuse our hairdresser example, imagine that in addition to reminding the customer of their appointment the day before, they also want to send an email the day after their appointment to thank them and offer them 10% off their next appointment. Here's how to implement this using Courier's automations:

Sending two scheduled emails in one automation

As this involves sending two different emails, you will need to create a second email template for the 10% off offer. Create a new template with the following body:

```
Dear {name},

We hope you were satisfied with your hair appointment yesterday, and
we would like to offer you 10% off your next booking.

Thanks,

The {salonName} Team
```

Now that you have both email templates ready to go, you can create your automation in Courier. The automation sends the first email, then there is a delay (in the example below, there is a two-minute delay, but for real-world use, you would probably set it to two days), and then it sends a follow-up email. We will continue to use the PHP Crunz package to kick off the automation at the right moment (one day before the customer's scheduled appointment), meaning that the second follow-up email will be sent one day after their appointment.

To follow along with this example, you will need to have followed the steps in the "Prepare your PHP environment" step explained earlier.

In your `tasks` directory in your PHP project, create a file called `scheduleAutomationTasks.php` and paste in the following code:

```php
<?php

use Crunz\Schedule;
use Courier\CourierClient;
use Dotenv\Dotenv;

// Configure environment variables - set the .env directory to the parent of the Tasks directory
// Environment variables are stored in a variable so that they can be passed to the scheduled task, which will no
$dotenv = Dotenv::createArrayBacked(__DIR__ . "/..")->load();

// Configure scheduler
$schedule = new Schedule();

// Configure the Courier PHP SDK - note the first null is the API path, of which we will use the default
$courier = new CourierClient(null, $dotenv['COURIER_AUTHORIZATION_TOKEN']);

// Create a new scheduled task
$task = $schedule->run(function () use ($courier, $dotenv) {

    echo "Running " . __FILE__ . "\n";

    // Invoke an automation using the Courier PHP SDK
    $automation = (object) [
        "steps" => [
            [
                "action" => "send",
                "recipient" => $dotenv['TEST_AUTOMATION_RECIPIENT_USER_ID'],
                "template" => $dotenv['TEST_AUTOMATION_NOTIFICATION_TEMPLATE_ID_1'], // Reminder email
                "brand" => $dotenv['YOUR_COURIER_BRAND_ID'],
                "data" => [
```

```
                        "name" => "John Doe",
                        "salonName" => "Cutting Edge"
                    ]
                ],
                [
                    "action" => "delay",
                    "duration" => "2 minutes" // You will probably want to delay by days or hours, but minutes are ea
                ],
                [
                    "action" => "send",
                    "recipient" => $dotenv['TEST_AUTOMATION_RECIPIENT_USER_ID'],
                    "template" => $dotenv['TEST_AUTOMATION_NOTIFICATION_TEMPLATE_ID_2'], // Follow-up email
                    "brand" => $dotenv['YOUR_COURIER_BRAND_ID'],
                    "data" => [
                        "name" => "John Doe",
                        "salonName" => "Cutting Edge"
                    ]
                ]
            ]
        ];
        $result = $courier->invokeAutomation($automation);
    });

    // Schedule the automation for a specific time
    $task->on('13:30 2023-07-01')
        ->description('Sending scheduled automation');

    return $schedule;
```

Ensure you've updated your `.env` file with any configuration you need to run this automation:

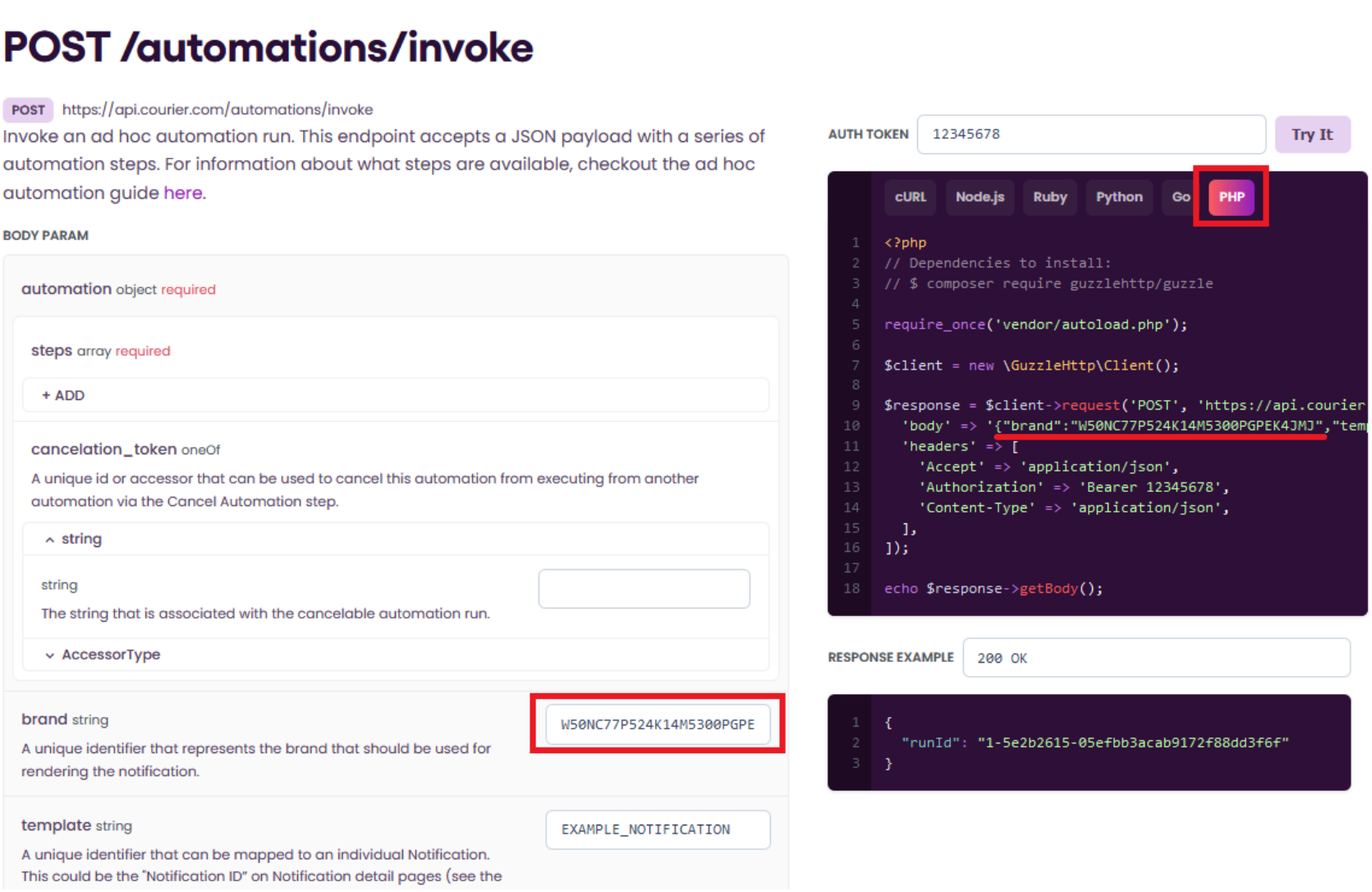
COURIER_AUTHORIZATION_TOKEN	COURIER API KEY, FOUND IN SETTINGS
TEST_AUTOMATION_RECIPIENT_USER_ID	Find your user ID in Courier's list of users
TEST_AUTOMATION_NOTIFICATION_TEMPLATE_ID_1	The notification template ID for the "Hair appointment reminder" email template
TEST_AUTOMATION_NOTIFICATION_TEMPLATE_ID_2	The notification template ID for the "Hair appointment — 10% off" email template
YOUR_COURIER_BRAND_ID	Choose your brand and find its ID in its "brand settings" or URL

Now run `vendor/bin/crunz schedule:run --force`, and you will receive the two different emails with the specified delay in between.

Dynamic automations API documentation

To understand all the features that automations can offer, you can play around with our dynamic request builder in Courier's [automation API documentation][20]. This allows you to build up your PHP automation request dynamically by adding request parameters. For example, if you add your preferred Courier brand ID in the "brand" box, your brand ID will be automatically added to a PHP request on the box on the right. You will need to select the "PHP" button to get a PHP request; however, other languages are available at the click of a button.

One of the key features of a Courier automation is the series of “steps” that make it up. To understand the different steps that can be part of an automation, see Courier’s extensive documentation.



Option 3: use Courier's no-code automations designer to build complex logic around scheduled emails

The automations designer is a UI tool for building automation templates in Courier. An automation template offers a way to reuse Courier automations, and because they can be created in the Courier UI, they are super easy to create. Even your non-developer colleagues will be able to create automation templates using Courier's simple drag-and-drop canvas.

For this example, we will use a "remember to pay your taxes" email, as this email could be sent on a schedule (April 1 of this year) or as a recurrence (April 1 every year).

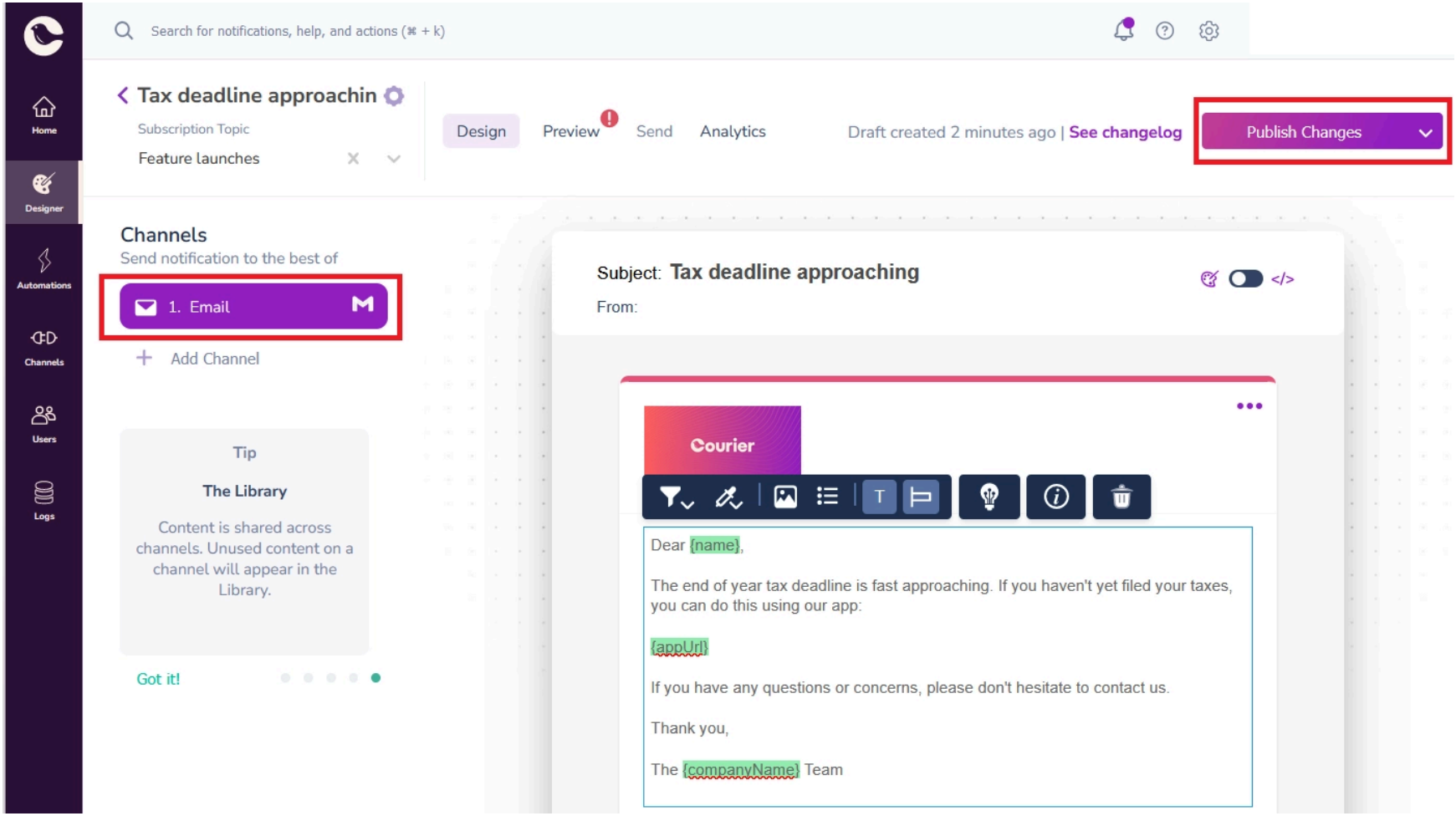
Create a notification template

Create a new notification template with the subject "Tax deadline approaching," and use the AI content generator to create some text for the body of your email. This may looks something like this:

```
Dear {name},

The end-of-year tax deadline is fast approaching. If you haven't yet filed your taxes, you can do this using our

{appUrl}

If you have any questions or concerns, please don't hesitate to contact us.
```

```
8
9    Thank you,
10
11   The {companyName} Team
```

Now that your template is complete, click **Publish Changes**.

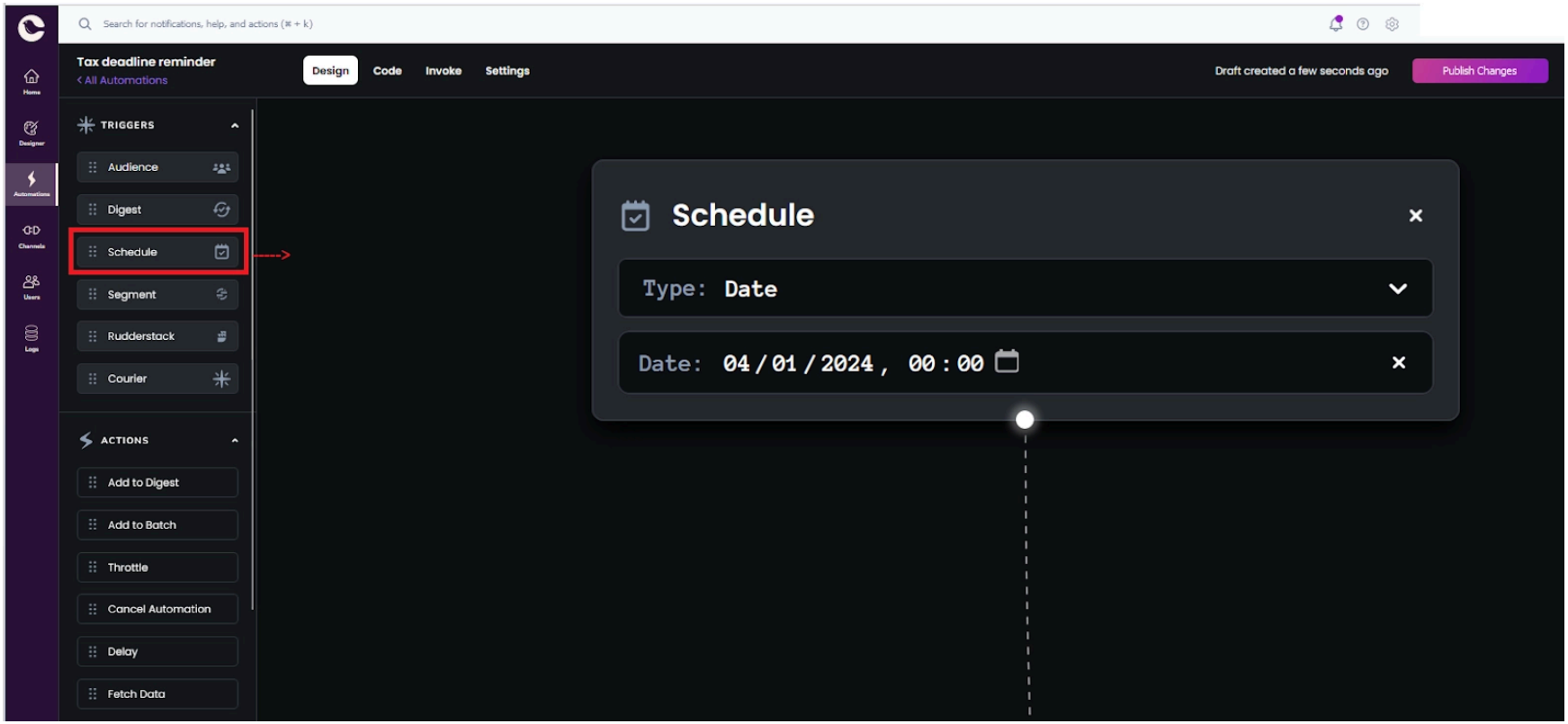


Create an automation template

Navigate to automations, and click **New Automation**. Rename your automation from “Untitled Automation” to “Tax Deadline Reminder.”

To define the trigger for your automation, drag the **Schedule** trigger onto the canvas.

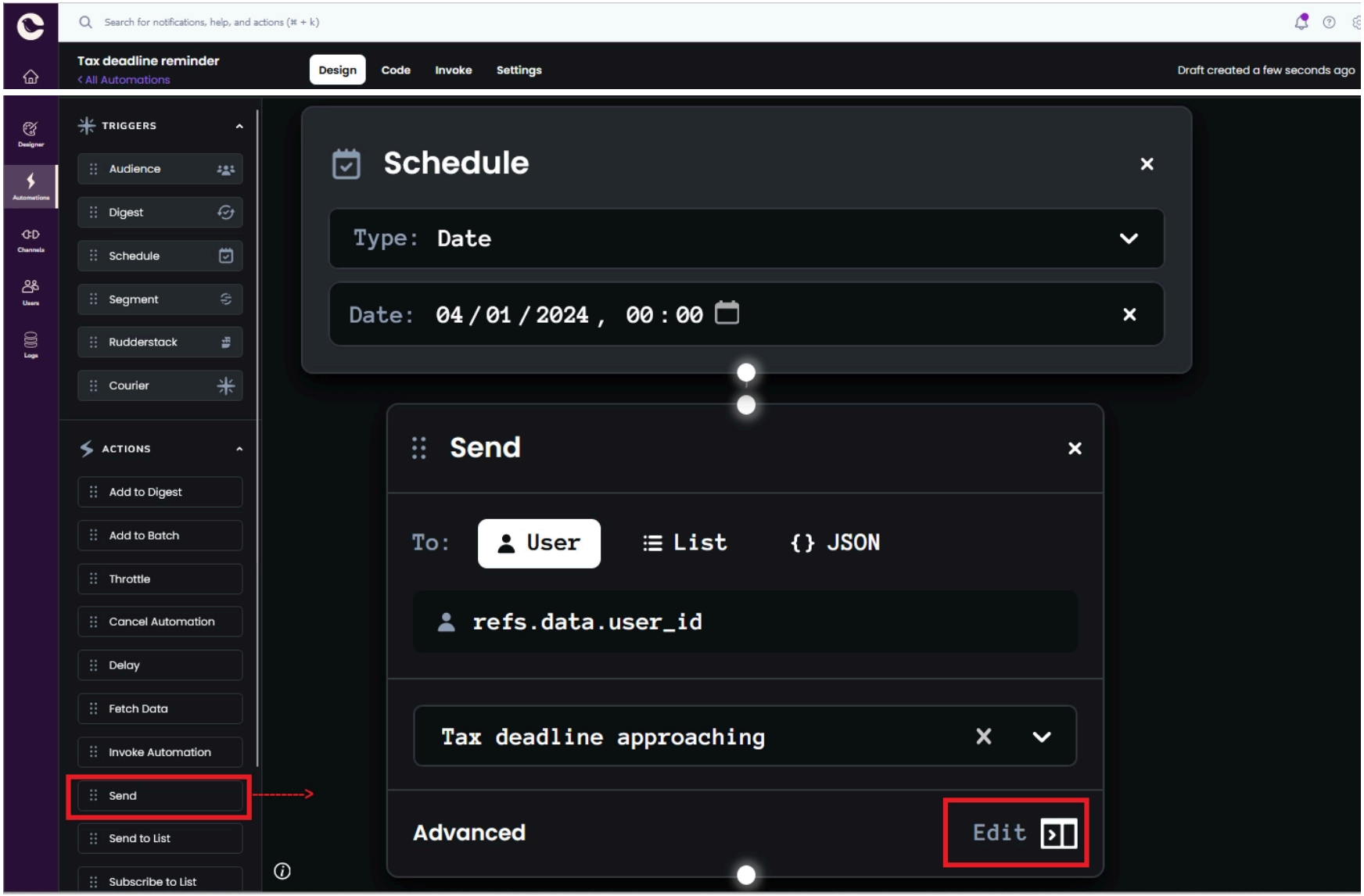
For a one-off scheduled email, change the **Type** of the **Schedule** node to **Date**, and enter the date and time you want your notification to be sent — in this case, we will choose midnight on April 1, 2024.



For a recurring email, change the **Type** to **Recurrence**. Then set a start date of April 1, 2024, 00:00, an end date of April 1, 2028, 00:00, and a frequency of **Yearly**. This will ensure the reminder email is sent for the next five years.

Next, drag a **Send** action onto the canvas, and ensure a line connects the bottom of the **Schedule** node to the top of the **Send** node so that it's clear that the send action follows the schedule trigger.

Enter `refs.data.user_id` as the user that the email should be sent to, and select your "Tax deadline approaching" notification template from the drop-down box. Now, click on **Edit** next to **Advanced** to edit some advanced properties.

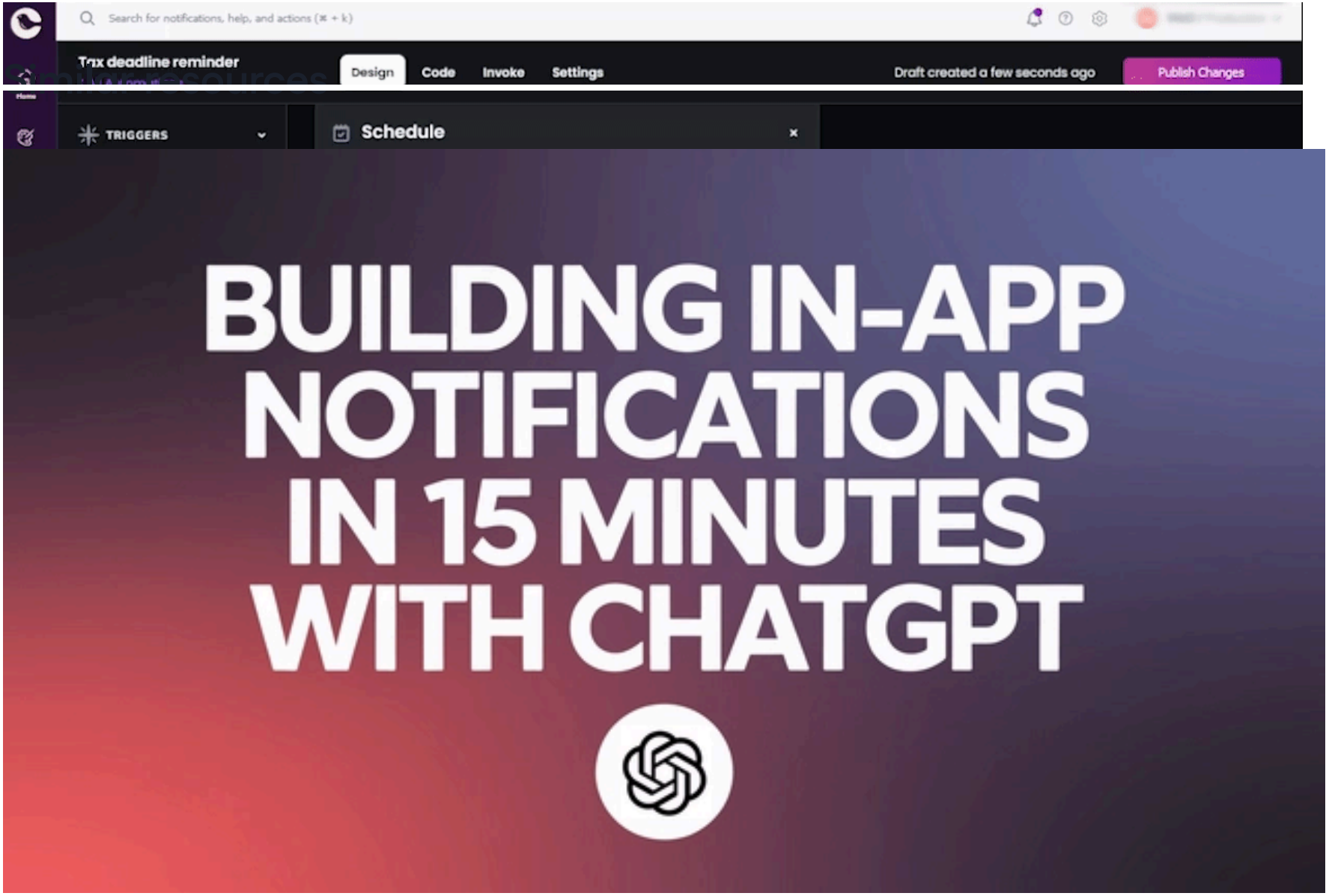


We will use the **Advanced** area to add some data to send to your automation template. This includes the `user_id` referred to in the previous paragraph plus any variables that your **Tax deadline approaching** notification template may be expecting.

Add this JSON to the **Data** section of the **Advanced** area:

```
{
  "name": "John Doe",
  "user_id": "courier-user-id",
  "appUrl": "example.com/file-taxes",
  "companyName": "Acme Ltd"
}
```

Back to top



Sending digests on a recurring schedule

Tutorial

Build Custom In-App Notifications in 15 Minutes with Courier and ChatGPT

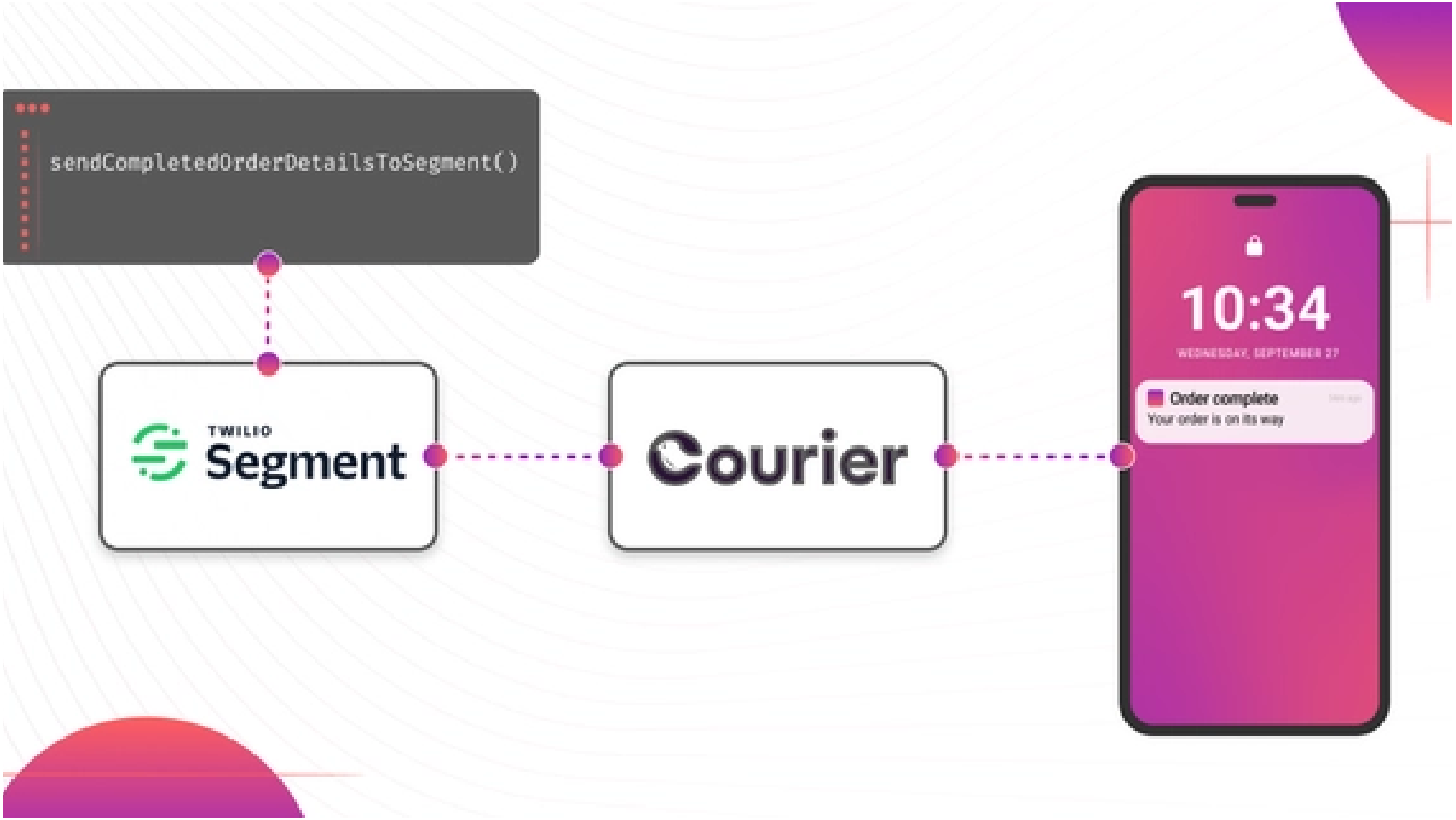
If your reason for sending recurring emails is that your users are subscribed to regular news or updates from your app, you may want to set up an email digest. A digest allows you to avoid overwhelming your users

When a customer asked for a fully custom in-app inbox that looked like an email client, Courier's sales engineer Jon turned to ChatGPT. Using Courier's React Hooks SDK and a simple prompt, he built a working prototype in just 15 minutes. This post walks through how AI and flexible tooling helped solve a real implementation request—fast.

With too many emails by condensing a large number of emails into one regular email update. Courier now offers the ability to send email digests. For more information, see the documentation.

By Thomas Schiavone March 26, 2025

Conclusion



If you haven't yet joined Courier, you can sign up today and get started immediately. If you're an existing user, you can easily try scheduling one of your existing notification templates in our automations

Tutorial

How to Set Up Automatic Push Notifications Based on Segment Events

designer, or use our PHP SDK to get started with scheduled and recurring emails.

Push notifications have carved their own niche as a powerful tool for continuous user engagement. Regardless of whether an app is actively in use, they deliver your messages straight to your user's device. Two key players that can combine to enhance your push notification strategy are Segment and Courier. In this tutorial, we show you how to set up Courier to…

By Sarah Barber November 17, 2023

Tutorial

How to Send Firebase Notifications to iOS Devices Using Courier

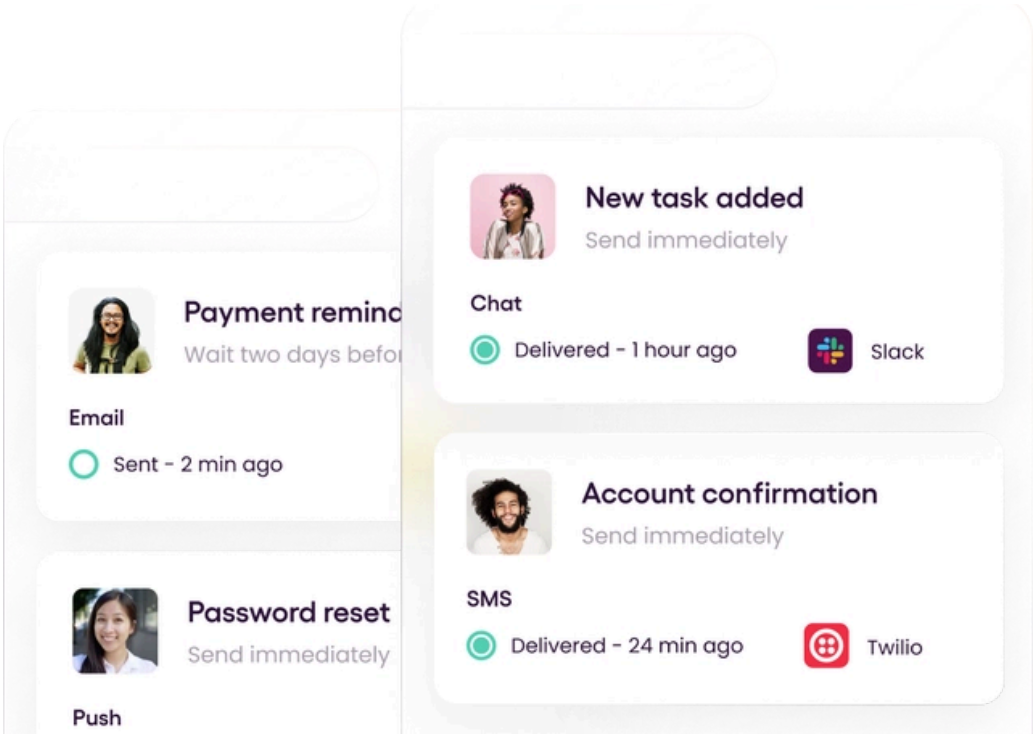
This tutorial explains how to send push notifications to iOS devices from your iOS application code using Firebase FCM and Courier's iOS SDK.

By Martina Caccamo November 01, 2023

Build your first notification in minutes

Send up to 10,000 notifications every month, for free.

→



Platform

Users

Content

Channels

Use Cases

Transactional

Action Required

Activity Center

Sending
Workflows
Preferences
Inbox
Workspaces
Observability
API Status
Changelog

Alerts
User Preferences
Digests & Batching

Explore

Docs
Pricing
Integrations
Discord Community
Demo

Company

About
Blog
Careers
Customers
Security

© 2025 Courier. All rights reserved.