Build

The art of crafting a Slack app your team will love

Contents

Finding Workflows

Slack is where you communicate with your team, make decisions, get updates, and build company culture. This happens not just with your colleagues but with apps and services you use to get your work done. From to-do lists to spreadsheets to service monitoring, your apps communicate with each other, and Slack is at the center of that communication

This guide will help you build a great Slack app for your team. We'll start with a primer on how to find the right workflow: which features to use and how to successfully roll it out to your team. What you're reading is also meant for the entire team, not just engineers or project managers — the best apps, the ones that end up shaping how your company works, require input from across the organization.

We'll focus on the types of apps that teams tend to build for themselves, not necessarily something you'd build as a service for other teams to install or to be listed in Slack's app directory. These internal apps are built primarily to help your team with a specific business process or a workflow. Throughout the book, we'll use the term "workflow" to mean any set of steps that helps you get your work done more simply.

Assembling your team

Once you're ready to build an app, the first thing to do is assemble your team. This team will determine what the app does and how it will be built. Some of these team members will be obvious: the engineers who will write the code and the PM who will oversee the project, for instance. If you already have a workflow in mind — for example, letting team members file a helpdesk ticket from within Slack — be sure to include someone from those corresponding team as well. In our helpdesk ticket example, pull in someone from IT who will receive these tickets as well as a colleague who will be filing requests.

If you don't yet have a workflow in mind, but you know you want to integrate Slack into more processes, that's great! You have a unique opportunity to help make your teammates' working lives simpler and more productive. This means you might need to cast a wider net when recruiting team members for your project. Someone from a team that's already heavily invested in Slack would be a good fit. If you know someone whose team is using Slack just for chat, or colleagues from disparate teams, such as operations and marketing, these are also great stakeholders to bring in.

Ideally, this team should be big enough to gather a diverse set of opinions but small enough to keep things manageable. Most projects should have about six to ten stakeholders — any fewer and the project is probably too specific to a department, any larger and it'll be challenging to find consensus on what to work on and how to build it.

With your team set up, it can be helpful to establish a few roles. The first is the facilitator, the person who will be guiding the process (maybe this person is you!) The facilitator gathers everyone together, schedules time to meet, assigns pre-work and roles, takes notes, and generally keeps things moving on track. This is similar to a Project Manager role, and it's important for this person to be organized and have goals in mind, but not have a pre-set destination.

Another helpful role is the decision-maker, someone who can definitively say “yes” or “no” when the team is locked at an impasse. It’s important for the decision-maker to be open-minded about what the team is hoping to accomplish, but also to have the authority to make sure decisions stick when it comes time to launch the app and bring everyone on board.

It’s almost certainly best if the facilitator and decision-maker are separate people who understand the importance of their roles. It can sometimes be challenging, especially on small teams, to maintain so many different roles, but try to keep these two distinct.

With the facilitator and decision-maker picked out, the rest of the team is empowered to bring their expertise and experience to the task of building your app. Some of these folks will be technical and will view the project through the lens of what is (and often isn’t!) feasible. Some will be more process-oriented and will see things via the people and tasks to be done. It’s a good thing. These diverse perspectives will help you decide what’s right to build and how to build it.

Finding your flow

Picking the project to work on can be the trickiest part of building an app. You might have an engineering team that’s already integrated Slack with other services as part of a larger DevOps pipeline. There may be other teams that use Slack primarily as a chat application and want to know more about these apps that they’ve heard about, but don’t know where to begin or find the process of building an app intimidating.

In general, we try not to be too prescriptive about how to build an app — every organization is different, after all. There is, however, one activity that we’ve found helpful to identify those critical workflows for your team.

Step One

Identify your team, which you may have already done (or had done for you, perhaps by the decision-maker!). With the team in place, figure out who will be the facilitator and the decision-maker.

Step Two

Book a room for a ninety-minute meeting. Preferably, everyone will be onsite and will be able to assemble in a single room, but videoconferencing can work as well, just make sure the A/V setup is in working order.

Step Three

Invite your attendees at least one week ahead of time so they've got plenty of notice. You'll also be giving them a bit of pre-work and an assignment they'll need to bring along with them. There's a template for the invitation below.

About the pre-work assignment: it's pretty quick, doesn't require any technical knowledge of the Slack platform or even ever having used Slack before. Everyone on the team should spend a few minutes in their email archive and find one or two automated messages that they've received in the past week or so that are critical to their job. These might be simple notifications, analytics reports, responses to an action they've taken, or timed requests to complete a task.

Everyone should print out this email (it helps enormously to actually print it out!) and highlight the bits that are most relevant to their work. It could be a terse bit of text, a call to action link, or a chart or graph. With those highlighted bits in mind, flip the sheet of paper over and diagram an ideal path for this message. It doesn't need to be beautiful, boxes and arrows with a few labels are great. Feel free to include links to other systems in the diagram — for example, maybe receiving this message means you need to log into a separate, unrelated app. The idea is to understand where this message comes from and what steps you need to take to get your work done.

Here's an example invitation you can use to get the team ready.

> *Hello! Thanks so much for agreeing to take part in our project building a new Slack app. We'll be hosting a kickoff meeting [details here]*
>
> *To get ready for our meeting, we have one request. This should be pretty simple, please don't spend more than half an hour on it (it'll probably only take 5-10 minutes).*
>
> *Open your email and find an automated message you've received in the past week or so. For this exercise, try to pick something that comes from one of the tools you use in your day-to-day work (so, not marketing emails). It could be a regular update from a sales system, a request to take action on something from HR, or a notification from one of your ops systems. Next, go ahead and print out that message then highlight the parts of the email that are most relevant to your job.*
>
> *Finally, flip the page over and quickly diagram how this automated message should work best for you — it doesn't need to be beautiful, boxes and arrows are great. Just outline where the message comes from, other additional data or actions you might want and what the ideal end result would be.*

Meeting your needs

When it's time for the team to get together, everyone should have a process in mind they'd like to discuss based on the pre-work exercise with their email. After everyone gets assembled, the facilitator can give people time to introduce themselves, especially if this is the first time the group has all met.

Next, everyone should introduce the process they chose and be given an opportunity to really explain how it impacts their day and their work. There's plenty of time at this stage and everyone should feel free to really dig in with some specifics. Why does this email matter? How often does it show up? What's the next step? When do

you consider the task complete? This part will likely take up about a third of the meeting time, so 20-30 minutes depending on the size of the group. The facilitator should take some brief notes on the whiteboard, just enough so everyone will remember what each process is about.

After everyone has presented their workflow and ideas, it's time to discuss. The goal here is for everyone in the room to have a high-level understanding of the needs of the people in the room. This is a great opportunity to understand different functions of your team or company! It will also help everyone get an idea of which workflows are most useful across their organization, not just a particular pet project.

With a bit of shared understanding in place, it's now time to decide what to build. This could be as simple as the decider saying "we're going with this option" or a longer discussion amongst the group about where to allocate resources. Finish up the meeting with a one-line statement that will define what the team will actually build. It should be specific and with little-to-no jargon. Some examples you might consider are:

"Anyone in the company will be able to file a helpdesk ticket from Slack that the IT team can then respond to from a shared triage channel."

"A daily roundup of ongoing sales figures, posted in each region's sales channel by 10am local time."

With a little bit of prep work and a single meeting, you'll have a clear idea of how your custom Slack app will improve existing processes, reduce context switching between tools, or bring a bit of delight to your team. Now that you have a mission in mind, it's time to start building. Next we'll explore the capabilities of the Slack platform and take a look at what the APIs can do.

Platform Features and APIs

You've now decided what app you want to build. You've used Slack before, and even tried a few apps — you have a pretty good idea of what's possible so you sit down and start figuring out the features of your app.

But, wait. What is possible with a Slack app? And how will your app actually communicate back to your Slack workspace? Which APIs will you use to build your app? This section is all about the features of the Slack platform and an overview of the APIs available to use. Even if you've built a Slack app before and are pretty familiar with the ins-and-outs, we're constantly adding and updating features, you might discover something new.

Choosing the right API

The Slack platform is composed of several APIs that let you build apps of all shapes and sizes. Generally, these APIs work together but there's also a bit of overlap that can sometimes be confusing. Some of the APIs have similar feature sets but require different ways of connecting to them. This section will help you find the right API for your project.

The RTM API

The Real Time Messaging (RTM) API is one of the original ways of connecting another application to Slack. Your app connects to Slack's infrastructure via a websocket, which creates an always-on connection between Slack and your app. All activity that happens in your workspace — every message in a public channel, every new channel created, every emoji reaction — is pushed through this channel. The traffic can be overwhelming and maintaining those websocket connections can get onerous. For these reasons, we often recommend using other APIs for building your app.

The RTM API is mostly a legacy API. It made sense when teams using Slack were small or for apps that were transitioning over from using something like Hubot or IRC. These days, however, Slack workspaces have hundreds, thousands, or even tens of thousands of team members. (There's one notable use case where using the RTM API can make sense, we'll cover that a bit later.)

The Events API

It's helpful to think of the Events API as an evolution of RTM — instead of creating an always-on connection that gets a firehose of Slack activity, you pick the events you're interested in receiving and Slack will send them via regular HTTP to an endpoint you specify. No more managing websockets or filtering for a specific message, the needle in a massive haystack of Slack activity. If your app only needs to take action when someone joins a channel (or uses an emoji reaction), just subscribe to that event and wait for it to come to you.

The Web API

If the Events API is how Slack pings your app, the Web API is how your app pongs back to Slack. Often, this will be because you want to take some action on a Slack workspace, such as composing a message or creating a new channel. Other times, it's because you want to request something directly, such as a list of all the people in a workspace.

The Conversations API

The Conversations API is technically a subset of the Web API, meaning it's a set of functions you use to call Slack directly. These functions deal specifically with all of the various channel-like objects that can exist in a Slack workspace, specifically public and private channels, shared channels, external shared channels, DMs, and multiparty DMs. For the most part, these channel-like objects all behave the same (you can list the users in them, for example), however they will have unique attributes around things like how private it is.

APIs make your app work, and help it communicate with Slack. They're important to understand at a high level, even though they're built for engineers. So you know what your app is capable of, here's a look at the features you can use.

Slack App Features

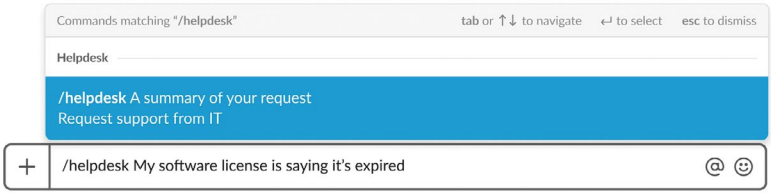
Your app will use some subset of the following features. Simple apps might just post notifications, where more complex workflows might require starting with a slash command, gathering more information via a dialog, then updating the ongoing status.

Use this brief overview of features to get an idea of what your app should accomplish. The rest of the book will cover these more in-depth.

Initiate an action with a slash command

Slack has a number of built-in slash commands that act as short cuts to get things done — for example, you can type `/mute` in any channel to reduce notifications for that channel. Apps can define slash commands, too, as a way of kicking off a process.

When thinking about the types of slash commands you'd want to add to your app, think about actions your users will want to instigate on their own, rather than responding to a notification or a message. Filing a helpdesk ticket would be a good example.



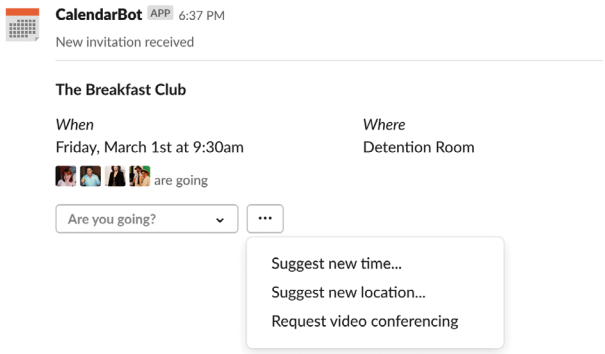
Post simple notifications and rich messages

Posting notifications into Slack is a simple way to connect your workspace to the other tools and services you use. You might, for example, want to receive a notification in Slack rather than via email.

Messages don't have to be just simple text notifications, though. They can include rich formatting and even complex user interfaces. To create these messages, you'll use a UI framework we call Block Kit to build rich app interfaces right inside Slack. These interfaces work just as well on a mobile device as they do in a browser or the desktop client.

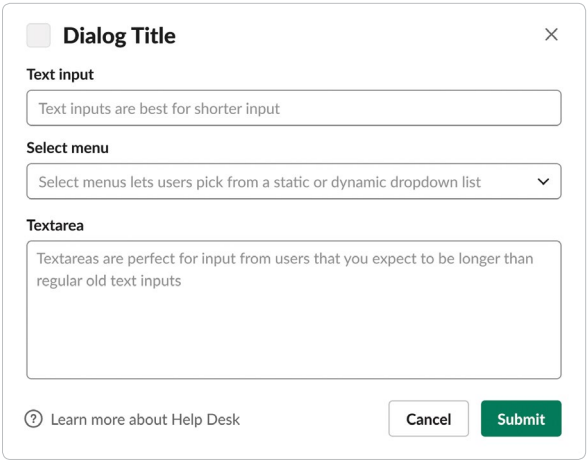
Actionable messages with interactive components

Interactive Components help make those notifications actionable by letting you attach buttons, dropdowns, date-pickers, and overflow menus. This means you can immediately turn a notification into the start of a workflow. For example, a message arrives from your helpdesk app with a dropdown attached to let you assign it to a fellow team member.



Gather data using dialogs

Dialogs are modal windows that contain forms for gathering more information from people. This helps you collect structured information in a concise, easy-to-use interface, without needing to build out a full back and forth conversation.



Message actions extend Slack's UI

Message Actions create a new surface area within the Slack user interface specifically for you app to use. Send messages from a channel to your app and then assign a bug report, create a task, follow up on a ticket, and more.

Add context via app unfurls

Posting links to websites in Slack will result in what we call an unfurl if we can grab the metadata about the page (we unfurl this data to show more context). But what if the link is to an internal system that requires authentication credentials? App unfurls let your app add crucial context to links from other systems so your message contains all the right information inline without forcing team members to context switch into another app.

Sketching Your App

Now that you have an idea about the capabilities of the Slack platform, a team ready to build a great app, and a solid foundation for what you want to build, there's another tool we'd like to put in your hands to help you think through how your app will work.

Included with this book is a stencil that includes all of the major interface elements you can use with your app — everything from buttons to dropdowns to emoji reactions. You can use the stencil to sketch out and storyboard the basic flow of an app before committing any design or development resources to get an idea of how your app will feel and discover pieces that might be missing.

At first glance, it might look like a template for just one kind of app, but the idea is you can mix and match the pieces to suit your interface. If your app just needs a button to add some action to a notification, feel free to keep it simple. If you're building a more robust back-and-forth workflow with inputs, inline images, and a dialog, use them all.

And if you don't have access to the stencil or if you want to see the code that composes your prototyped message, you can use the Block Kit Builder online at api.slack.com/tools/block-kit-builder.

Posting notifications

Pushing to Slack

The most common point to start building a Slack app is also one of the most straightforward — pushing information or notifications into a workspace. These can take the form of a simple heads up from another app or system; or can be richly formatted messages that include features like images, glanceable contexts, and components like buttons or select menus. These also make an ideal starting point for your app because they're easily discoverable to everyone on your team. Once your app is installed, configured, and built, notifications will simply start to appear for your users — they won't have to set anything up to start receiving them.

Because notifications are so simple to start posting in a workspace, it's important to make sure you're crafting the right notifications and sending them to the right place. Overwhelming your colleagues with alerts is a sure-fire way to get your app uninstalled. We've written a guide at https://api.slack.com/start/ that includes guidelines on crafting effective notifications that will delight, rather than annoy, your team.

You have two options for posting messages via your app — incoming webhooks and the `chat.postMessage` API method — each with a different level of complexity. The folowing section will help you choose the right option for your app.

Incoming webhooks

The first method — incoming webhooks — is the simplest to set up, but also the least flexible. Webhooks are a standard way to exchange bits of information between systems via single-use URLs. A Slack app's incoming webhooks let you designate a channel to receive incoming messages. The app will then generate a unique URL that you can use to post messages into Slack. This is great for sending notifications from another server, when you just want a mechanism for posting without needing to manage state or handle authentication credentials. In fact, it's possible to post via a webhook using nothing more than the standard Unix curl command, like this:

```
curl https://hooks.slack.com/services/T12345678/
B98765432/ABC123DEF456GHI789000XXX

Content-type: application/json
{
    "text": "Hi there!"
}
```

The URL is completely unique to your workspace and to the channel you specified to receive messages at install time, so you don't need to store any OAuth credentials. The webhook URL is only able to post into the single channel you set up and you can't read data from a Slack workspace using the generated hash. It's important to keep this URL secure since anyone with access to it can post a message into your Slack workspace.

Incoming webhooks include the ability to add message formatting, so it is possible to generate fairly sophisticated messages with nothing more than a webhook. One of the most common use cases for incoming webhooks is to set up systems monitoring on remote servers — a process might run periodically to check that its overall system health is good, for example. If not, it could post a notification into Slack detailing the problem. The message could even include a chart generated by a graphics library showing what the issue might be.

Incoming webhooks do have a few key limitations. First, they can only post to a single channel. The URL itself is limited to a single channel that's specified when you install the app. It's not possible to override this with a separate channel name or ID, either, it's just the one channel.

On their own, incoming webhooks can't receive responses to the messages they post or process interactive buttons. For interactive notifications, you'll need to use the appropriate Web API methods. It's probably best to think of incoming webhooks as one-way only channels that are simple to set up and use for posting notifications into your workspace.

Incoming webhooks also can't receive responses to the messages they post — it's not possible to make them interactive with buttons, for example, or to know if someone posted a reacji to a message. They're one-way only channels, which has the advantage of being simple to set up and get started with.

chat.postMessage

For complete control over posting messages across an entire workspace, there's the `chat.postMessage` Web API method. Using `chat.postMessage`, your app can:

- Post into any channel it has access to
- Compose sophisticated app-like interfaces using Slack's Block Kit UI framework (more on this a bit later), and
- Receive responses to build true interactivity into your app

These extra powers come with a bit more upfront complexity, however. Rather than having a single, simple URL that your app can just post messages to, your app will call the Slack API with an authentication token you receive when the app is installed. These tokens are powerful and should be safeguarded within your app's code.

For more of the technical nitty-gritty on how to manage OAuth tokens, including safe and secure storage, please visit https://api.slack.com/docs/oauth-safety

---

Similar to incoming webhooks, using `chat.postMessage` is a matter of making an HTTP call to Slack's API. You can use something like the Unix curl command again but will most likely be using an HTTP library alongside your programming language of choice. Calling the API looks something like this:

```
curl https://slack.com/api/chat.postMessage —token
xoxp-abc... —body ...
```

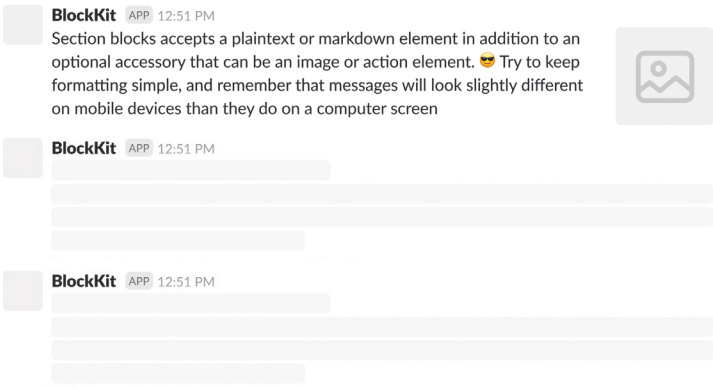
All of the formatting available to incoming webhooks also works with `chat.postMessage`

Formatting messages with block kit

You've probably used basic message formatting in your day-to-day Slack usage, such as surrounding a word with asterisks to make it bold, ***like this***. Apps can also format text but they also have the power to build richer user interfaces with system we call Block Kit.

Block Kit is Slack's framework for composing more app-like user interfaces. As the name implies, you build messages out of individual (or groups of) pre-defined blocks such as text, thumbnail images, dividers, and interactive elements like buttons, dropdown menus, and date pickers. Even with a relatively small number of starting blocks, it's possible to customize interfaces unique to the needs of your workflow.

Block Kit UI's are defined as JSON, which makes them fairly straightforward to build without having to learn a new markup language. We also have a visual tool available for building out sample messages in a visual interface and then have the corresponding JSON generated automatically. This is great for letting anyone on your team sketch out some ideas for messages and then copy and paste the resulting JSON.



There are no hard and fast rules on when to choose incoming webhooks vs. `chat.postMessage`. It really does depend on the needs of your workflow. If you just want a lightweight way to send simple notifications from another system into a Slack channel, without having to worry about managing auth tokens, incoming webhooks are an obvious and easy way to get started.

If you already know you're going to want these messages to go beyond just notifications and make them something people can respond to, or if you want to build a more sophisticated UI, it's worth going through a few extra steps to set your app up using `chat.postMessage`.

It's also possible to use both options, either in separate contexts or as a way to iteratively build your app. As you and your team are getting started, incoming webhooks are great for getting going quickly. These can be replaced or augmented with more sophisticated messages via `chat.postMessage`.

Once your Slack app can post notifications and messages into Slack, let's explore how to add context to your team's conversations.

Adding Context

One of the more delightful bits of magic an app can perform is to automatically add context to the conversations your colleagues are having in Slack. You're likely already familiar with a built-in version of this: when you post a link to just about any public web page, Slack will grab some metadata about that page and attach a preview of the page to your message. Your app can perform similar actions using data that requires authentication.

It can send messages directly to a specific user in-channel using an ephemeral message that only they can see, and can receive events when a message mentions your app, at which point it will respond accordingly.

Ephemeral messages

Sometimes you want your app to send a message to someone in the channel they're in (instead of a direct message), but you only want that specific user to see it, rather than the entire channel. A common example is if you need to show an error message that alerts the user to a problem, but doesn't break them out of the workflow they were in by forcing a context-switch over to a DM space.

We call these *ephemeral messages* and they work almost identically to public messages except they are scoped to a specific user. Ephemeral messages have all of the same formatting options and are able to include interactive elements, they're just only visible to one person.

Most often, an ephemeral message is posted in response to an action someone has taken — the simplest use case would be something like an error message letting the user know that something went wrong. Ephemeral messages can also be used for further clarification, for example, maybe your app is used to display some data in a channel and it needs a bit more information from the person who posted the initial command. For the most part, we encourage apps not to send ephemeral messages out of the blue and rather use them in response to activity, however there might be use cases where it makes sense to post a reminder or something in the context of the channel. Use your discretion here.

Ephemeral messages are sent using the `chat.postEphemeral` command, which works almost identically to `chat.postMessage` except for the additional, required `user` parameter. Ephemeral messages are also not guaranteed to be delivered if the person isn't logged in at the time and they won't persist across clients or if an existing client is reloaded. Treat them as ethereal and as fleeting as the air we breathe.

App mentions

Mentioning a coworker is a standard way to bring someone's attention to a conversation, for approval or just to make sure they've got `:eyes:` on something. Apps can also listen for a mention and respond, albeit in a more automated fashion. App mentions work much in the same way mentions do for people — typing `@yourapp` will notify your app. If your app isn't already part of a channel, the mentioner will have the option to invite you to that channel. If you app gets invited, or is already a member of the channel, your app will be able to respond to the mention.

From here, it's pretty open ended what your app should do. Most likely you'll take the contents of the message and do something with it, like search a knowledge base. Giving your response a little personality is great, desirable even, but keep in mind a little personality goes a long way. Overly chatty personalities can get annoying pretty quickly, so it's best to keep the back and forth to a minimum and use interactive features like buttons or dialogs as much as possible. Be sure to check out the App UI Guidelines book for more about communicating clearly via Slack apps.

App unfurls

Building out a custom response to links posted in a workspace is a powerful way to customize the Slack experience for your team. Custom unfurls are most often used to perform some task when someone posts a link to a website that isn't on the public internet, such as a site that requires authentication. For example, you might post a link to a sales forecast from your CRM system in the middle of a discussion about upcoming goals because it's handy to see the data from that link presented right inline. No need to leave the discussion and context-switch to your CRM in a different app.

App unfurls are also useful for building out custom workflows from any link. A publishing company or marketing department might build a workflow where any time someone posts a link to the publication or blog, the app adds a custom attachment that includes buttons for opening the CMS to edit the post, or sending it to an editor from a pre-selected list.

Apps create custom unfurls by subscribing to any valid, fully-qualified domain name. It doesn't necessarily need to be a domain you control, which means you can override the default unfurling behavior if you'd like. You can choose up to five domains to listen for and don't need to worry about specifying subdomains. Once your app is set up to listen for a domain, anytime a link matching it gets posted in a Slack workspace, your app will receive an event that it can respond to using the `chat.unfurl` method.

Making Messages Interactive

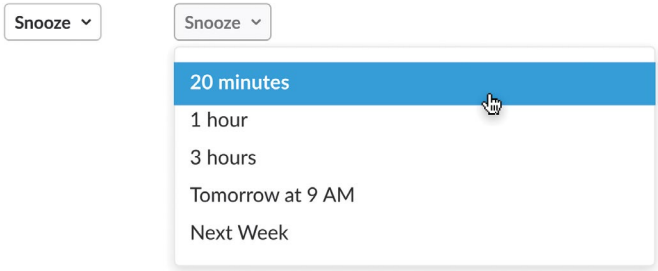
Slack messages can do so much more than inform and clarify information, they are opportunities for action as well. Actionable messages are at the heart of the workflows that let your team get real work done from Slack.

Taking action

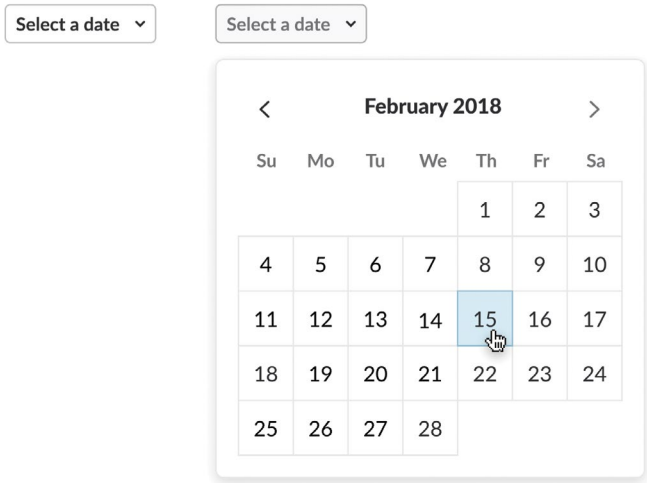
The action block is the most immediate way to make your app interactive. This block currently features four elements: the button, the select menu, the date picker, and the overflow menu. Use these separately or combined in any combination to build out rich workflows right from Slack, reducing the need to context switch over to other apps.

When someone interacts with any of these elements, Slack will send a small payload to your app's backend for processing. What happens next is up to you.

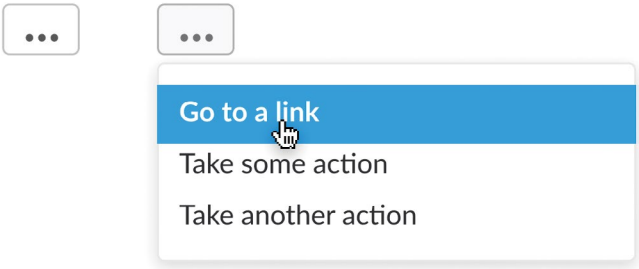
Buttons are the most basic way of capturing some command from a user.



Select menus are great for offering a compact, pre-built set of options. In addition to options you might decide to hard code, select menus can also include a list of the people in a workspace, a list of channels, or a type-ahead style menu that interacts with your backend.



Datepickers show a standard calendar dropdown for picking a day, month, and year. Combine this with a select menu if you need to add time, too.



The overflow menu is a way to keep your interface tidy while also providing additional options. If you need a quick and easy way for your user to access part of your app that's not front and center — such as updating the frequency of a notification, for example — the overflow menu is a great place to stash it.

Message actions embed your app in the Slack UI

These UI elements are great for building an array of different kinds of apps and workflows, using Slack messages as the container for your interface. It's also possible to extend the Slack interface for your workflow with message actions.

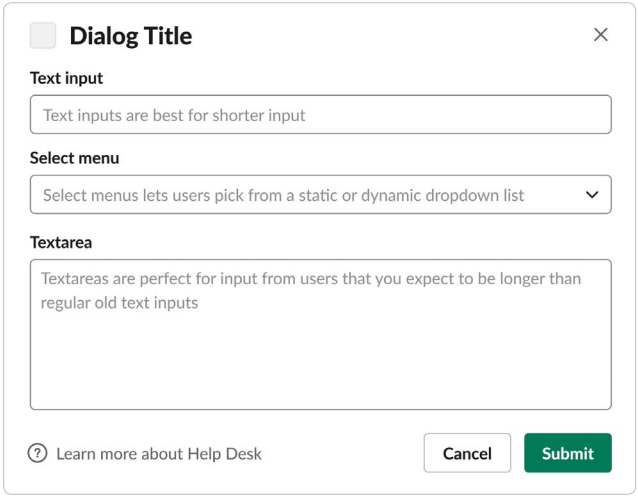
Message actions let your app hook into a message's native overflow element — those three dots that you use for activities like copying a link or setting a reminder about a specific message. Your app defines custom actions that let you do things like create a bug or attach a message to a to-do list managed by your favorite third party app.

Message actions are another way of using apps to enhance the conversational context of Slack. Consider a pretty common use case — filing an internal helpdesk ticket. The issue might get diagnosed in a support channel where someone asks for help and receives follow-up from your support staff in the thread of that original message. Once the support team gets to the bottom of the actual problem, a message action would make it simple to file a ticket in the

support system for follow up. Slack remains a great place to triage the problem while the ticketing system remains the system of record for managing the team's helpdesk. The team member reporting the problem can get back to work without having to context switch between apps.

Use dialogs to gather information

We've been using forms to collect feedback and structured data on the web for 25 years now — they're pretty well understood! Despite the prevalence of chatbots and AI assistants, they remain one of the easiest and clearest ways to gather input, and they're a feature your apps can use too. Dialogs offer another surface area for your app that go beyond the messaging pane to create modal windows for quickly gathering bits of information. A dialog can only be created in response to a direct user action, such as clicking a button, so you don't have to worry about pop-up spam. Currently, dialogs support a simple subset of forms: single line text entries, drop down select menus, and longer text area elements that can contain up to three thousand characters.



Bringing it all together

There's no reason any of these features needs to exist in isolation — Slack apps work best when features are used together. A notification might arrive with a set of buttons attached that, when clicked, spawn a dialog for collecting more information. This could be a simple back-and-forth, or a more complex series of steps involving multiple messages (some of them could even be ephemeral!)

Your app's job is to help orchestrate this dance, to pick and choose the features that will make your colleagues' jobs simpler, to automate repetitive tasks, or just add a bit of contextual delight to the otherwise everyday conversations happening in-channel.

Notes:

Notes:

Notes: