Title: **Building an interactive Slack bot**

Summary:

Slack is a popular messaging app used by over 750,000 companies including 65% of the Fortune 100 companies. Did you know you can do more than just send text messages with the odd image or emoji? It's possible to create a rich interactive UI using Slack's Block Kit UI framework. In this session we'll build a simple hot-desk booking bot with Node.js where you'll learn how to respond to events from your controls and adjust the displayed message.

Note to organizers:

I use this all the time in my day-to-day work! I prototype messages in Slack's tool (https://app.slack.com/block-kit-builder/), I do up my weekly status update and send it directly from the tool, I've created a hot-desk booking system (this talk will be a simplified version of that), and a tool to turn dev servers in aws on/off from slack!

Level: **Beginner**

Duration needed: **35 minutes for the talk**, 10 minutes for Q&A

Presentation will be on: **Friday, February 24th at 10am** (2nd session of the day)

**Setting up for the talk:**

* In an incognito window, make sure I have Slack open to the workspace: <https://slack.com>
  + Signed in to the **ConFoo 2023** workspace
  + Make sure the general channel is empty and the bot is deleted from the Slack api
* Open the code snippet file
* Open the desk booking message file
* Open Visual Studio code to the basic code: D:\ConFoo\_Slack
* Open PowerPoint and run it
* Have my notes Printed and on the laptop/my phone as a backup
  + **Check things off as I go** to make sure I don’t miss anything
* Have a countdown timer from 35 and leave the phone on so that I can see where I’m at
* Have water and take a sip every once in a while.

**The Talk itself…**

* While in the PowerPoint
  + **Welcome slide**

Good morning everyone!

Welcome to my talk on building an interactive Slack bot!

***[next slide]***

* + **About me slide**

My name is Gerard Gallant.

I work at Dovico Software and currently wear a few hats.

CIO is a role that I’m expanding into but I’m also a senior software developer and architect.

Totally unrelated to this talk… I had wanted to write a book since I was a child and was recently given the opportunity by Manning Publications to write a book about WebAssembly called “WebAssembly in Action”.

If you want to reach me, you can find me on Twitter or LinkedIn.

***[next slide]***

* + **Slide outline**

When Covid hit we all went home for what we thought was going to be two weeks. We thought the pandemic would be over after that point and life would go back to normal.

Two weeks became many more weeks and, when restrictions were finally lifted, most of us were content with working from home. We had a setup that worked for us, we had our routines, saving money, time and less stress with no more commutes and so on.

As a company, it didn’t make sense to keep renting out a large section of an office building’s floor if very few people were coming into the office so we decided to buy our own building.

The new building was smaller so it didn’t have enough desks if everyone wanted to come in at the same time.

We needed a way to reserve a desk so that people didn’t make a drive in only to find out there was no room.

At the same time, I couldn’t justify paying a monthly fee for an app when all we needed to do was say that we wanted a particular desk so I started looking around at what we were already using for tools to see if there was an opportunity there.

I decided to try using Slack for our desk booking.

I started out with a poll and then, as we got things set up at the office and the number of desks available grew, I switched to a post with multiple messages were staff could add an emoji to the desk message they wanted to reserve.

In the end, I wanted to automate the posting of the message so that I would have to do it manually every day and that’s when I looked into creating a Slack bot which we’ll talk about today.

***[click for bullet 1]***

For this talk, the first thing we’ll do is take a quick look at the Slack message we’re going to create and interact with.

***[click for bullet 2]***

Then we’re going to jump into setting up an app in Slack and building the code.

***[click for bullets 3 & 4]***

For the code, we’re going to leverage Slack’s Bolt for JavaScript SDK

***[Switch to Slack and show 3rd-apprach channel]***

If you don’t already have a Slack account, you can sign up for a free one which is nice because it has access to the Slack API.

This is the desk booking message that we’re about to build.

You’ll be able to @ the bot’s name to trigger it to post this message, clicking a button will indicate that you reserved the desk, clicking the button again will remove your reservation, and the bot will be automated to post the message at the same time each workday.

Creating the bot itself is a combination of code and Slack API settings. I’ll show you the bot’s code in steps and switch over to Slack to see each step’s progress.

* + **Slack/Code**

When you’re in Slack, the first thing that you’ll need to do is to switch over to Slack’s API.

There are a couple of ways to reach this… one is by clicking on the workspace name, **Settings & administration, Manage apps** context menu, and then the **Build** link in the top-right corner.

Or… you could just navigate to **api.slack.com/apps** in your browser

While we’re here, I’ll quickly point out that Slack has lots of great documentation that you can get to by clicking the **Documentation** link at the top right ***[click the link]***.

If you scroll down, you’ll see a **Bolt** link. Click on that ***[click]*** and these are some SDKs created to help build Slack bots. Clicking on **Bolt for JavaScript** ***[click]***, there’s a **Getting started** link at the top which is pretty useful ***[click]***: <https://slack.dev/bolt-js/tutorial/getting-started#create-an-app>

The Bolt for JavaScript SDK will be what we’ll be using for today’s code.

Tones of documentation! We’ll be coming back to this in a moment.

**<STEP 1 code if need be>**

***[Switch to VS Code]***

I was going to walk you through installing the npm packages but, for time’s sake, I’ve already set up the basic bot code.

***[Show contents of package.json]*** I’ve installed the bolt SDK package and node-schedule which will be used to automate sending the message to slack every evening.

***[Show contents of app.js]*** Before we can run this code, we need the **Bot Token** and **Slack Signing Secret** values from the Slack API.

To get those values, we need to create an App in the Slack API

***[Switch to the API]***

* Click on the big green **Create an App** button if you don’t have any apps *(there’s a* ***Create New App*** *button if you already have apps and the list of existing apps below that you can click on to edit)*
* Choose **from scratch**
* Give it a name *(****Desk Bot*** *for example)* and select a workspace
* Click the **Create App** button

The first thing you’ll see are the features and functionality that you want your app to have, App credentials will be needed by your code in order to communicate to the API, Display information allows you to adjust what gets displayed for the app once it’s added to Slack.

The bot needs the Signing secret so scroll down to the ***App Credentials section*** and copy the app’s Signing Secret by clicking **Show** first. In the VS Code terminal, save this to an environment variable:

**$ENV:SLACK\_SIGNING\_SECRET=""**

I’m using a PowerShell terminal here so I’m prefixing the variable name with **$ENV:**

For Linux/macOS replace ‘$ENV:’ with ‘export ’

Go back to the Slack API and under the ***Add features and functionality section***, click on the **Permissions** item.

Scroll down to ***Scopes*** and click the Add an **OAuth Scope** button under **Bot Token Scopes**.

For this app, we’ll need **app\_mentions:read** so that the bot can respond to someone doing an @BotName.

We’ll need **chat:write** so that the bot can post messages.

**users:read**

Now, scroll up and click **Install to Workspace** and when prompted, click **Allow**.

Copy the Bot User OAuth Token and switch back to VS Code terminal and save this to an environment variable:

**$ENV:SLACK\_BOT\_TOKEN=""**

Run the app and you’ll see that bolt is running.

**node app.js**

It doesn’t do anything yet but, congratulations, you have a bot!

***[Floor Plan message]***

With the basic app started we now need to create the floor plan message.

The bot can send messages in plain text or it can use Block Kit messages which allow for rich messages compared to what can be created directly in slack.

***[switch back to the Documentation page]*** Back on the Documentation page, scroll down to the section about Block Kit and click the **Block Kit Builder tool** link ***[click]***

You can also just go to <https://app.slack.com/block-kit-builder/>

When you first open this page, you a default message to show some of what’s possible.

On the right are basically code snippet type links that will get added to the bottom of your message ***[click on some]***.

The next pane to the right shows a preview of the message. There’s also a Templates button that you can click on ***[click]*** to get a message that has already been set up for you ***[click on Use Message Template for one – Trip Agent perhaps]***.

You can drag things around or delete items in this pane too if you want. It’s sometimes easier to do that here than in the Payload pane on the right.

In the right pane, you can edit the text and fine tune things.

Aside from using this to create the message for your bot, I’ve occasionally used this for sending posts to slack for things like my weekly progress report because I could get a message with much nicer formatting than if I had typed it directly in Slack.

You don’t need a bot set up to do that. Just hit the **Send to Slack** button and choose the channel when you want to do this ***[show them by sending the current template]***.

One thing to note about sending messages this way is that you can’t edit it in Slack. You can only delete it and resend it if you notice an error.

Click the **Clear Blocks** button ***[click]***

I don’t have time to walk you through building the whole message but it’s pretty straightforward.

We’ll have a markdown section at the top for the title saying that it’s the desk booking for the next day ***[click mrkdwn and change the text to \*Desk booking for [date]...\*]***

Then we’ll want an image for the floor plan ***[scroll to Image and click ‘title’… set the text to Floor plan]***

Then we’ll need some buttons for each desk ***[scroll back up to Section, button and add a couple]***

Add a divider to make it easier for employees to tell which desk is in which room ***[add Divider, plain]***

We could then copy the two desk button sections and divider and paste to create all the desk buttons ***[copy/paste once]***.

That’s the gist of building the message so I’ll paste in the full message

***[Paste in the code from the desk booking message file]***

A couple of things to point out… You’ll want each desk with a unique action\_id value. The value attribute can hold anything. Here, we’ve prepopulated it with some JSON to indicate who has reserved that desk (by default, no one).

Images will give an error if they’re not publicly accessible.

Slack’s Block Kit documentation will tell you which properties are required and other properties you can include. Buttons for example can have a style of primary to display with green text and a green outline or danger for red text and outline.

With the message created, you can either click the Copy Payload button or Ctrl+A and Ctrl+C ***[copy the message]***.

**<STEP 2 code if need be>**

Switch back to your code ***[switch back]***.

There are different ways to keep this message in your code. You could create a variable right here in app.js for example.

Personally, I prefer keeping the code and message separate so I’ll show you how I’ve done it at work.

Create a file called **deskbooking.js**

Add the line **module.exports.messageToPost =** and then paste the payload.

Add a semicolon at the end.

In your app.js file, after the current require statement, add the following line of code:

**const { messageToPost } = require('./deskbooking.js');**

You don’t really need the app\_mention event for this bot but it’s useful if something happens and the scheduled post didn’t happen. Maybe you accidently misconfigured the cron job or maybe you notice an issue with the message, tweaked it and then wanted to repost it. ***[Paste both functions]***

In my production app, I also listen for certain keywords so that I can post to a test channel that only I see rather than the main one when I want to test a layout change. I also have two layout files. The main one and a testing one so that I don’t accidently break the main layout while trying something out.

Also, because I want to have things post for the next day, if I tried to repost the message, the date on the message would get the next day’s date.

If I had to delete the current day’s post and repost it, I’d want the current date used instead so it checks for the word “today” in the message.

We could post the message in this function but, because I want schedule the posts, and not just the result of a mention, I use a common function that both sets of code can call.

For posting the message, all we need to adjust is the title with the proper date (today or tomorrow depending on how this function gets called).

Then we specify which channel to send the message to, the message blocks to send and fallback text used in notifications.

Let’s test out what we have so far.

Right now, we can talk to the Slack API but it can’t talk to us. To get around this, we’ll use an npm package called **ngrok**. To install it, in your terminal you can do:

**npm install ngrok -g**

I already have it installed on my system. Once it’s installed, tell it to run and which port to use with the following command (3010 is the port number I’ve specified in the app.js file):

**ngrok http 3010**

A terminal opens with a couple links that point back to our local host port. Copy the **HTTPS** link.

Now that we have a public endpoint that Slack can call, we need to tell Slack what that endpoint is.

Note that every time you run ngrok, it will generate a new endpoint and you’ll have to update Slack again.

\*\* Start the app (**node app.js**)

Switch back to the API and go to **Event Subscriptions** on the left. Togggle Enable Events on and paste the ngrok URI followed by **/slack/events**

Expand Subscribe to bot events, click **Add Bot User Event**, and choose **app\_mention**.

Click **Save Changes**.

Now, if everything is done right so far we can go into slack and @ the bot name to trigger the message to be posted into Slack. Let’s try it ***[do the @Desk bot]***.

If the bot isn’t in the channel yet, you’ll be asked if you want to add it. Press the **Add to Channel** button. The message won’t be triggered if you just added the bot to the channel because it wasn’t there to get your message. You’ll need to @ it again ***[do the @Desk bot again]***.

If you look at the console where ngrok is running, you’ll see that the latest call returned a 200 OK but if you look at the code, you’ll see in the terminal window that there was an error.

The error is talking about invalid arguments and the issue is that the postMessage call specifies an environment variable that we haven’t defined yet.

Because we want to post messages to the General channel in this case, we need to specify the name. Stop the app (Ctlr+ C) and then run the following command:

**$ENV:POST\_TO\_CHANNEL="#general"**

Restart the app (**node app.js**) and then @ the bot again in the channel ***[do the @Desk bot again]***.

Awesome! We’ve made good progress! You see the message with tomorrow’s date, the image, and all the desk reservation buttons.

If you @ the bot again and include the text ‘**today’**, you’ll see the message post but using today’s date ***[do the @Desk bot again with the text ‘today’]***.

The next thing we want to implement is interactivity with the bot so that clicking the button on the message causes it to say who has reserved it or allow someone to un-reserving the desk if they’re no longer coming in or are leaving for the day ***[show what happens when you click a button right now – an exclamation point next to the button and, if you mouse over that, you’ll see why]***.

**<STEP 3 code if need be>**

Listening for a button click is similar to how you listen for a mention exception button clicks are actions rather than events.

I’ll paste in the function and then I’ll explain it ***[paste the code]***.

The first value is the action\_id you’re listening for. We could set up 12 event handlers, one for each button id, and then call a common function to handle the work of adjusting the button to indicate who clicked on it. Or, as we did here, we can use a regular expression to handle all button clicks who’s action\_id starts with **Reserve**.

At work, I started by saying Desk1, Desk2 and so on but then we also had a few tables in a common area and giving it an id Table1, Table2 would have meant that I would have 2 action event listeners. Instead I prefix both desks and tables with the word Reserve and they all use the same listener.

You need to acknowledge that you’ve received the message from the api within 3 seconds or the user will be shown an error message so do that first before doing any other code.

Next, get the button id from the action’s action\_id property

Get the id of the employee who clicked on the button and see if the button has a value. If there is no value then the button isn’t reserved yet so we can set it to this employee’s name

Set the button’s value to the current employee and ask Slack for their name.

Adjust the button’s text value to hold a checkmark emoji along with **Reserved by** and the employee’s name

If the button has a value it means someone has reserved the desk. If the person who reserved it is not the current employee, exit this event handler now. Only the person who reserves a desk should be able to unreserve it.

If it is them, clear the button’s value and set the text to **Available**.

Now the fun… we want the current message adjusted but with just the one button changed so that other buttons are still in the same state.

The whole message can be found in the body.message.blocks property and we need to loop through it looking for the button that was clicked on so that we can adjust its value and text.

One thing to be aware of is that when a message is posted to Slack, Slack will add properties to the images.

If you tried to resend the message to Slack with those properties, you’d get an error because they’re not valid through the API. I just wrote to the console to see what the properties were and then removed the ones not indicated in the documentation. I only had an issue once where I had deployed the bot and then Slack was adjusted to have additional image properties.

Keep looping until we find a section with a button and it’s the button that was clicked on. Adjust the button’s text, value, and style.

Note that because we have to loop through the entire message to make sure all images have their extra properties removed, if you gave two buttons the same action\_id, they will both toggle.

Finally, send the modified message to Slack.

Restart the app and switch back to Slack ***[switch back to Slack]***.

Clicking on the buttons will still give you an error because there’s a missing permission ***[click on a button]***

Switch back to Slack, go to Event Subscriptions and copy the URI you added there.

On the left click on **Interactively & Shortcuts** and **toggle Interactivity on** ***[toggle it on]***.

Paste in the URI and the click **Save Changes *[Save]***.

Now if you click on a button, it should toggle to show that you’ve reserved it and toggle back to Available if you click it a second time ***[click on a button a couple times]***.

Something to be aware of too is that if you make a change to the message, like changing an id, only new messages will have that id so older messages might give errors if you also adjust your code to no longer handle the button click for example.

***TIP:*** I ran across this when I wanted to change the layout image with a slightly different desk configuration. I updated the image and the old posts showed the new image when I only wanted the new posts to use that image. I ended up having to revert the image change and use a different image name for the new posts. Little thing to keep in mind.

We’re almost there. We can trigger the message to show and we can toggle the desk buttons. We don’t want to be doing this manually every day because we might forget, loose power, be on vacation and it’s just one more thing to do in a very busy day.

What would be ideal is if the bot could post a floor plan on its own.

**<STEP 4 code if need be>**

For the scheduling, I chose to go with node-schedule.

Add the require statement for the module at the beginning of the file after the App’s require statement:

**const schedule = require('node-schedule');**

It’s a pretty simple function but I’ll paste it in and then explain it ***[paste it just after the postFloorplan function]***.

We set up a job that will run at 6pm AST on Sunday to Thursday. When triggered, it will post the floorplan telling it to use tomorrow’s date.

To demonstrate this, I’ll delete all messages from Slack first ***[delete the messages]***

Now, I’ll add 5 to the list because it’s not set to post on Fridays by default but today is Friday and we want it to post for this demo ***[add 5 to the list]***.

We also don’t want to wait until 5pm local time (6pm Atlantic) so adjust this to be in two minutes from the current time ***[The talk that started at 10am Eastern which is 11am AST and then adjust the minutes]***.

Restart the app (**node app.js**)

And there you have it!

**TIP**: Point out collaborators for the app – apps go into a weird limbo if you’re removed from slack and there’s no collaborator set up. They disappear even if your account is re-enabled. You need to contact support to get them back.

If I need a bit more to talk about *(my timing seems right but just in case I do go a bit too fast)*… show the way to enable a uri that allows you to make curl with. I use this for sending messages to slack from server scripts about things like the deployment of code and our daily message every morning listing everyone who’s out for the day; can also show them the dev status bot that allows control of AWS servers via Slack (devs can turn servers on/off without needing to enter AWS).

Does anyone have any questions?

Thank you!