



Workshop

Build Apps for Slack

MLH localhost



Welcome to MLH Localhost: Build Apps for Slack



Wifi Network:
[TAMU - WPA]

Wifi Password:
[↖(ツ)↖]



Event Hashtag:
#MLHLocalhost

Twitter Handle:
@MLHacks



Welcome! My name is Derek Li!

- 1** I'm here to lead this session & help you learn something new today!
- 2** I'm a Junior Computer Engineering major.
- 3** My favorite programming language / tool is Scratch.

1

*Using your Web Browser,
Open this URL & Fill out the Form:*

<http://mlhlocal.host/checkin>

2

Afterwards, Check your Email to Find:

- Setup Instructions
- An Invite to the MLH Slack
- The Code Samples
- A Workshop FAQ
- These Workshop Slides
- More Learning Resources



Our Mission is to Empower Hackers.

65,000+
HACKERS

12,000+
PROJECTS CREATED

3,000+
SCHOOLS

We hope you learn something awesome today!
Find more resources: <http://mlh.io/>

What will you **learn today?**

- 1 How to create an app for a Slack workspace using the Slack API
- 2 How to configure a Slack app
- 3 How to add a super cool game to your Slack workspace!

Table of Contents

-  **1.** Introduction to Apps for Slack
- 2.** Configure and Try the App
- 3.** Review and Update the Code
- 4.** Review & Quiz
- 5.** Next Steps

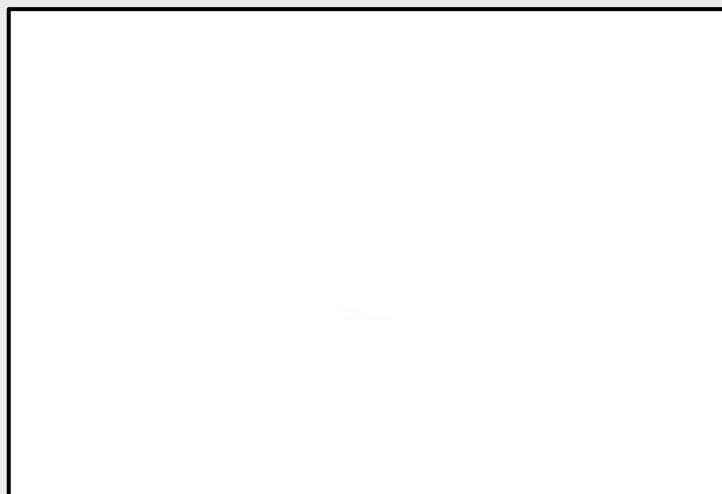
What is slack ?

Slack is a set of cloud based tools to make communication at work better.

- 8,000 customers signed up for the service within 24 hours of its launch in August 2013.
- As of May 2018, Slack has 8 million daily active users.
- Over 94% of paid Slack teams use apps.

What can slack Apps do?

- Send rich notifications into Slack from other services
- Use buttons, drop down menus, and dialog modals to send data back out of Slack to other tools
- Use conversational bots to have your app participate in channels



#eventplanning

You created this channel today. This is the very beginning of the #eventplanning channel. Purpose: Planning the event (edit)

+ Add an app 8 Invite others to this channel

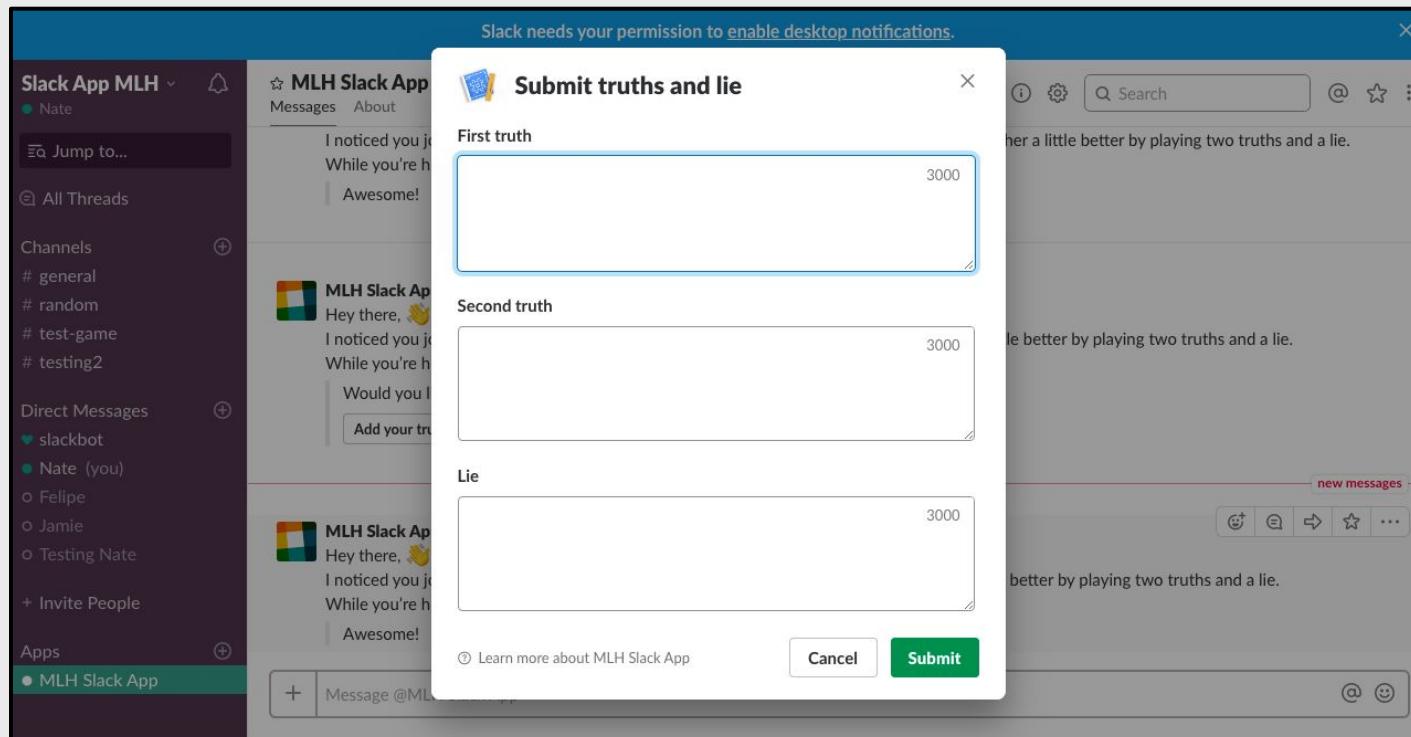
Today

samir 2:46 PM set the channel purpose: Planning the event

+ Message #eventplanning

What We'll Do with the Slack API

We're going to write a Two Truths and a Lie game that you can add to a Slack workspace! When someone joins your workspace, you can play this game as a way to get to know each other.

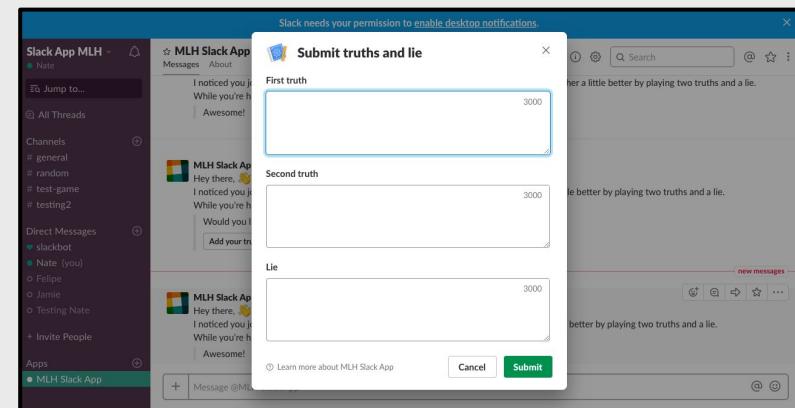


Let's Test it Out!

A few people in the room should go to the URL below, join the Slack organization, and try out the game! Not everyone has to play immediately – you can look on each other's screens.

mlhlocal.host/localhostslack

1. Go the URL above and join!
2. Open the workspace.
3. Join the #twotruths channel
4. Play!



What We Need To Do

There are several steps to getting an app up and running with Slack Apps!

What's coming?

1. Remix the code on Glitch. Use the Glitch URL to send requests to the Slack API.
2. Configure events, interactions, and permissions for the app.
3. Fix some bugs in the code.
4. Test it out!

Table of Contents

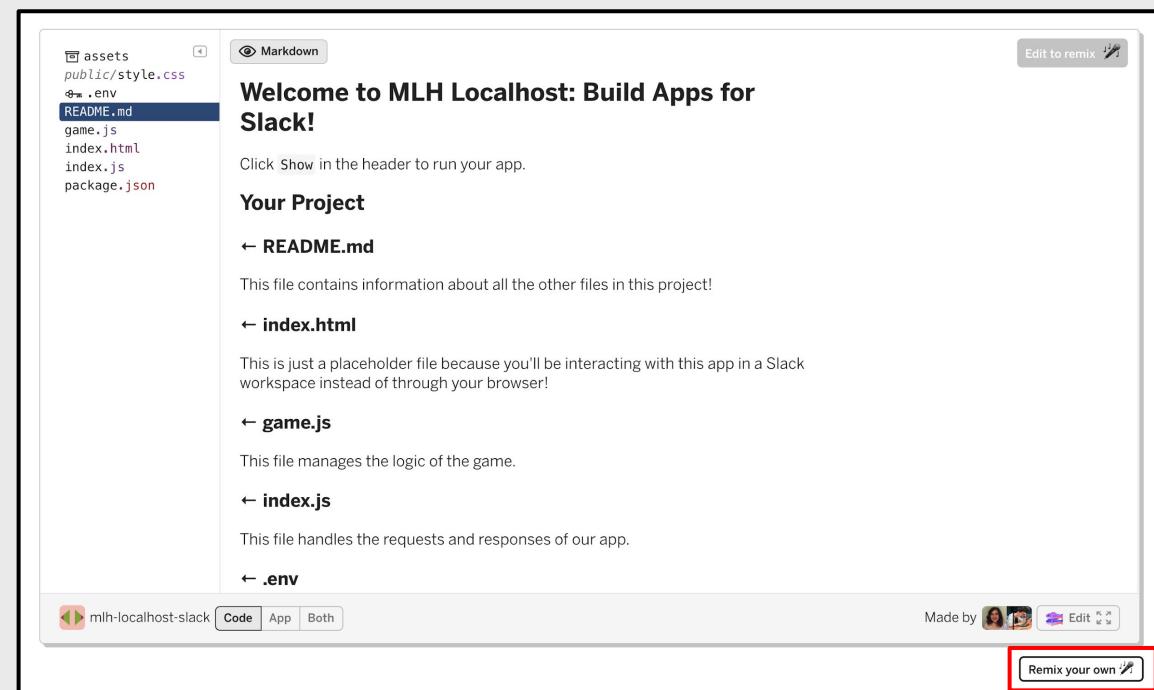
- 1.** Introduction to Apps for Slack
-  **2.** Configure and Try the App
- 3.** Review and Update the Code
- 4.** Review & Quiz
- 5.** Next Steps

Get the Source Code

Head to the URL below to see the code on Glitch!

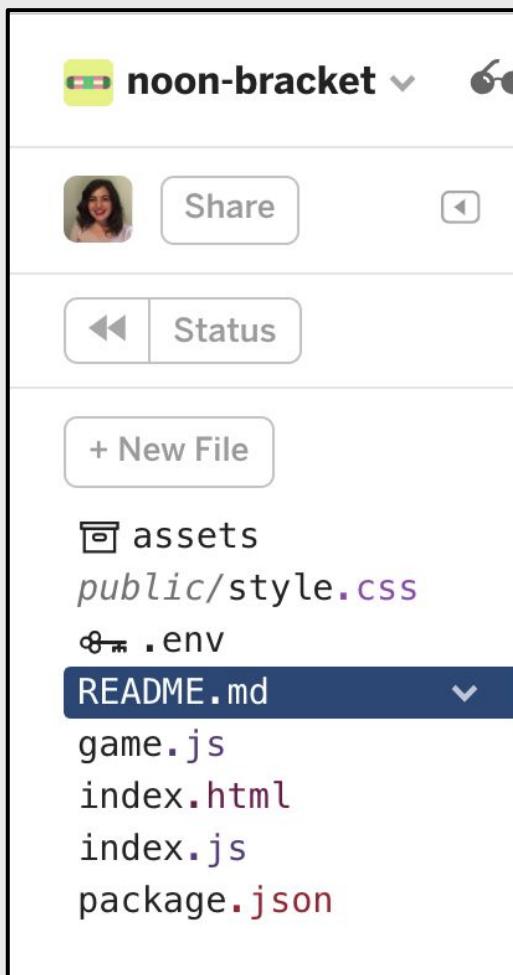
mlhlocal.host/slack-code

1. Click Remix your own
2. Sign In or Sign Up
3. Check out the code!



What's In This Project?

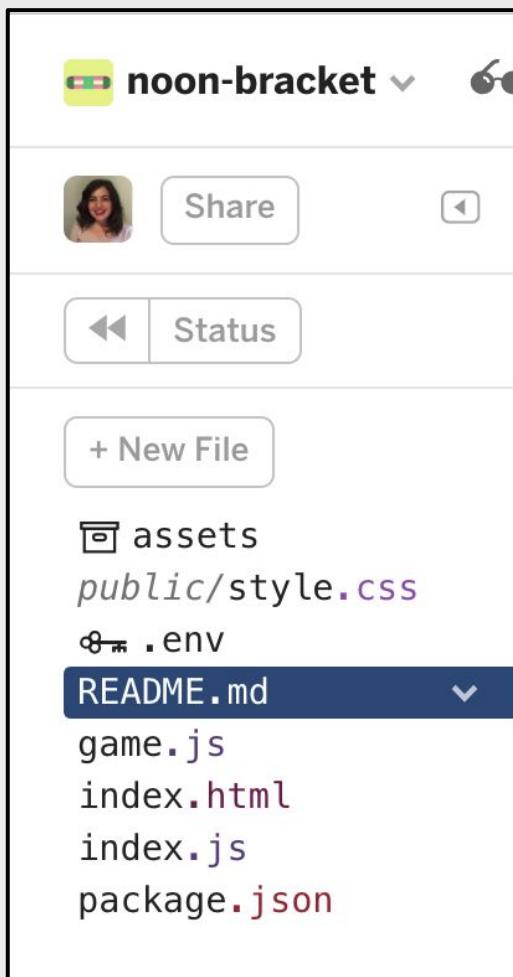
Open the project in Glitch and let's review!



- **package.json** - lists the node packages necessary for the app to work.
- **game.js** - this file contains the logic of the game.
- **index.js** - this file contains the express server to run the game and calls to the Slack APIs that run the game
- **.env** - this file will contain your private Slack API credentials.

What's In This Project?

Open the project in Glitch and let's review!



- **index.html** - This file provides a landing page for the Glitch app. This has no impact on the behavior of your app. We included this so that when you click "Show Live," you don't get a confusing error message.
- **public/style.css** - this file styles index.html.
- **assets/** - holds the image you see when you click Show Live
- **README.md** - A description of the app

These files control how the game is played. We need to create an App using the Slack API, configure it, and add it to our workspace to be able to play this game. Let's get started!

Create A New Slack App

Now that we have the code for our app and have joined a workspace, navigate to the URL below in a new browser tab.

<http://mlhlocal.host/slack-api>

1. You might have to log in. The workspace name is
mlh-localhost.slack.com
2. Enter your email address and password.

Sign in to your workspace

Enter your workspace's Slack URL.

Continue →

Sign in to Localhost

mlh-localhost.slack.com

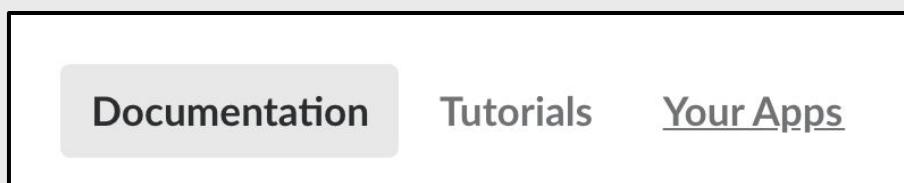
Enter your email address and password.

Sign in

Create A New Slack App

You might be redirected to the Slack workspace, but you need to be in the API console.

3. Navigate to **mlhlocal.host/slack-api** again.
4. Click **Your Apps**.
5. Click **Create an App** on the next screen.



A screenshot of the "Your Apps" page. At the top, it says "Your Apps". Below that is a section with the heading "Build something amazing." and a paragraph about using Slack APIs to build apps. At the bottom is a prominent green button labeled "Create an App".

Create A New Slack App

We created our app and assigned it to a workspace. Now, let's install it to that workspace and get started developing!

Create a Slack App

App Name

Don't worry; you'll be able to change this later.

Development Slack Workspace



Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the [Slack API Terms of Service](#).

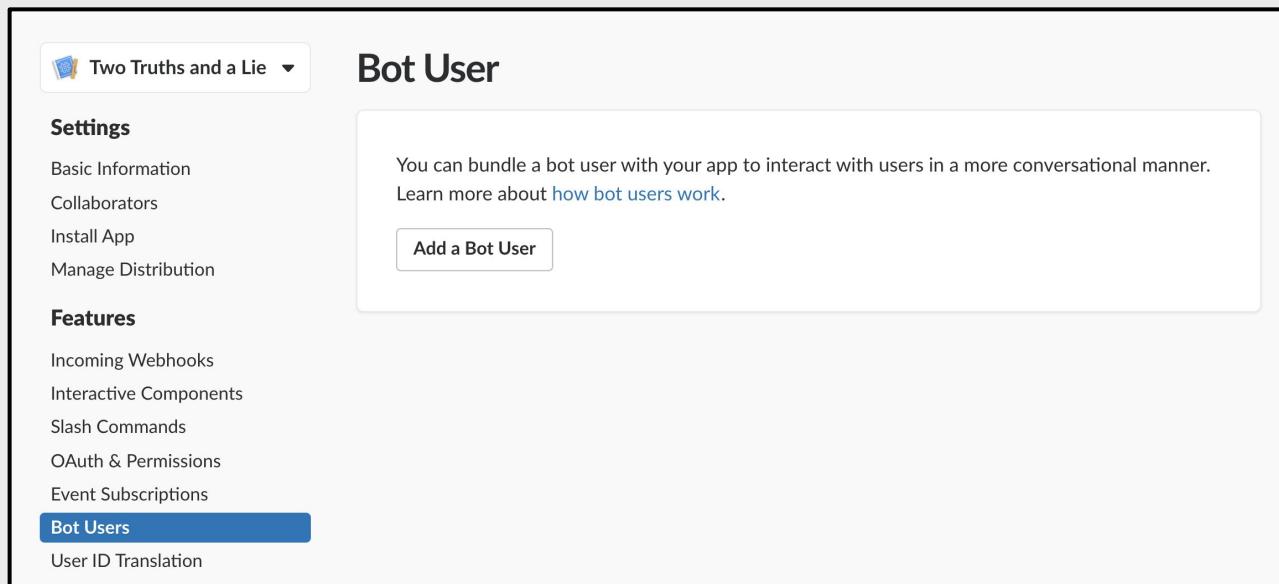
[Cancel](#) [Create App](#)

6. Click **Create App**. Now, let's configure it!

**The game is played by interacting with
a bot. Let's create the bot user!**

Add A Bot User

Players will interact with a Bot. Let's add the Bot to our app!



1. Return to browser where you have the Slack API dashboard open.
2. Under Features, Select **Bot Users**.
3. Select **Add a Bot User**.

Add A Bot User

Players will interact with a Bot. Let's add the Bot to our app!

Bot User

You can bundle a bot user with your app to interact with users in a more conversational manner. Learn more about [how bot users work](#).

Display name

Names must be shorter than 80 characters, and can't use punctuation (other than apostrophes and periods).

Default username

If this username isn't available on any workspace that tries to install it, we will slightly change it to make it work. Usernames must be all lowercase. They cannot be longer than 21 characters and can only contain letters, numbers, periods, hyphens, and underscores.

Always Show My Bot as Online Off

When this is off, Slack automatically displays whether your bot is online based on usage of the RTM API.

Add Bot User

4. You can keep the default settings. Click **Add Bot User**.

**Let's install the app to our
workspace so that we generate
the API credentials we need.**

Install the App to Your Workspace

Now let's add the App to your Workspace.

The screenshot shows a web-based application interface for managing a Slack app. On the left, there's a sidebar with a logo for 'Two Truths and a Lie' and a dropdown menu. Below it are sections for 'Settings' (with 'Basic Information', 'Collaborators', and 'Install App' buttons), 'Features', and 'Manage Distribution'. The main content area is titled 'Install App to Your Team' and contains instructions about installing the app to a workspace to test it and generate tokens. A prominent green button at the bottom of this section is labeled 'Install App to Workspace'.

Two Truths and a Lie ▾

Settings

- Basic Information
- Collaborators
- Install App**
- Manage Distribution

Features

Install App to Your Team

Install your app to your Slack workspace to test your app and generate the tokens you need to interact with the Slack API. You will be asked to authorize this app after clicking **Install App to Workspace**.

Install App to Workspace

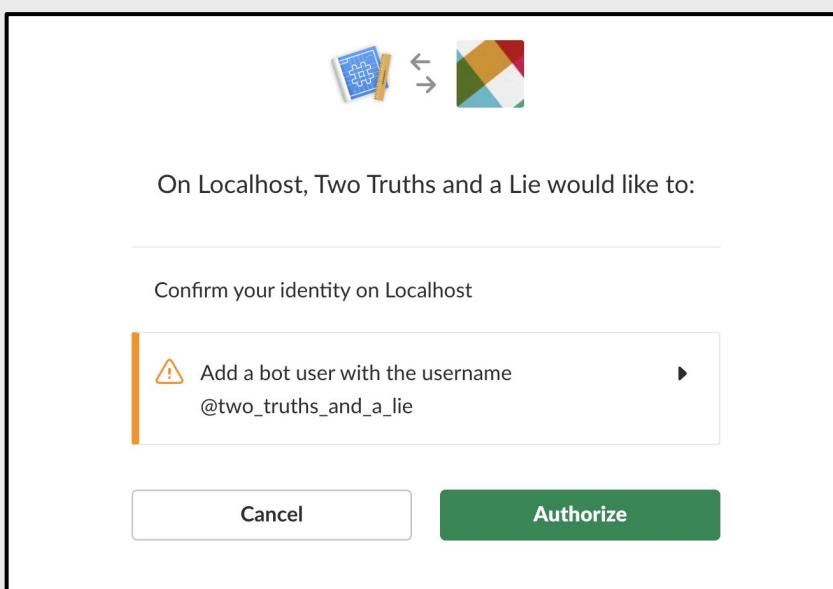
1. Select the **Install App** panel.
2. Click **Install App to Workspace**.

Install the App to Your Workspace

Now let's add the App to your Workspace.

3. Select Authorize.

4. Copy your Bot User Token.



Installed App Settings

OAuth Tokens for Your Team

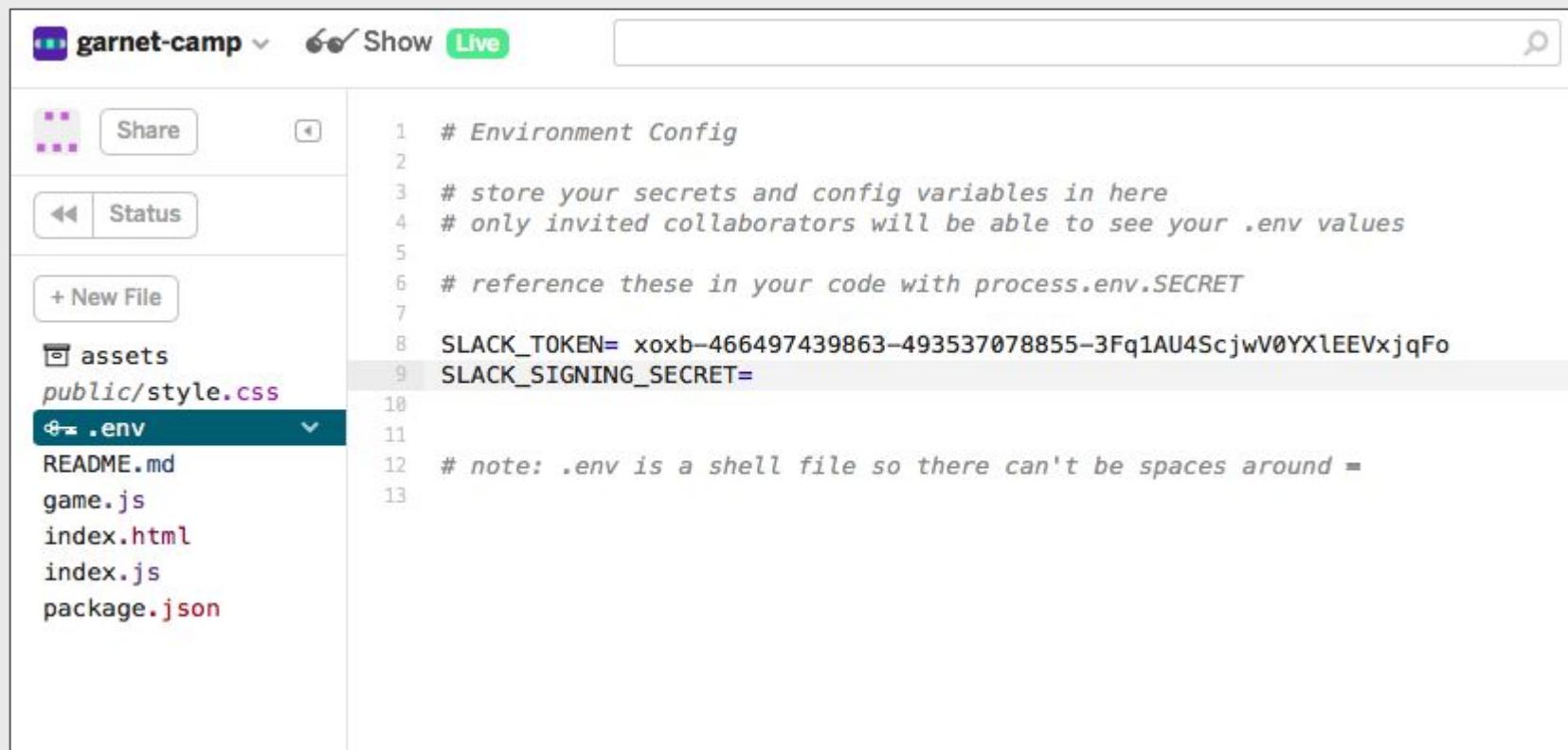
These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. [Learn more.](#)

OAuth Access Token
xoxp-466497439863-468623673399-471446132337-8b7fa428202c9a1457a8d4218 [Copy](#)

Bot User OAuth Access Token
xoxb-466497439863-472051158738-0Aq3b8hnuDLHC9jzm3ZvsYnC [Copy](#)

[Reinstall App](#)

Install the App to Your Workspace



The screenshot shows a Glitch project interface. The top bar includes the project name "garnet-camp", a "Share" button, and a "Live" indicator. On the left, there's a sidebar with icons for files and folders, and buttons for "Status", "+ New File", and ".env". The ".env" file is currently selected. The main area displays the contents of the .env file:

```
1 # Environment Config
2
3 # store your secrets and config variables in here
4 # only invited collaborators will be able to see your .env values
5
6 # reference these in your code with process.env.SECRET
7
8 SLACK_TOKEN= xoxb-466497439863-493537078855-3Fq1AU4ScjwV0YX1EEVxjqFo
9 SLACK_SIGNING_SECRET=
10
11
12 # note: .env is a shell file so there can't be spaces around =
13
```

5. In your Glitch project, open .env.
6. Paste the TOKEN you just copied on Line 8.

Now, let's connect our Slack app to our code. We'll have to verify our identity.

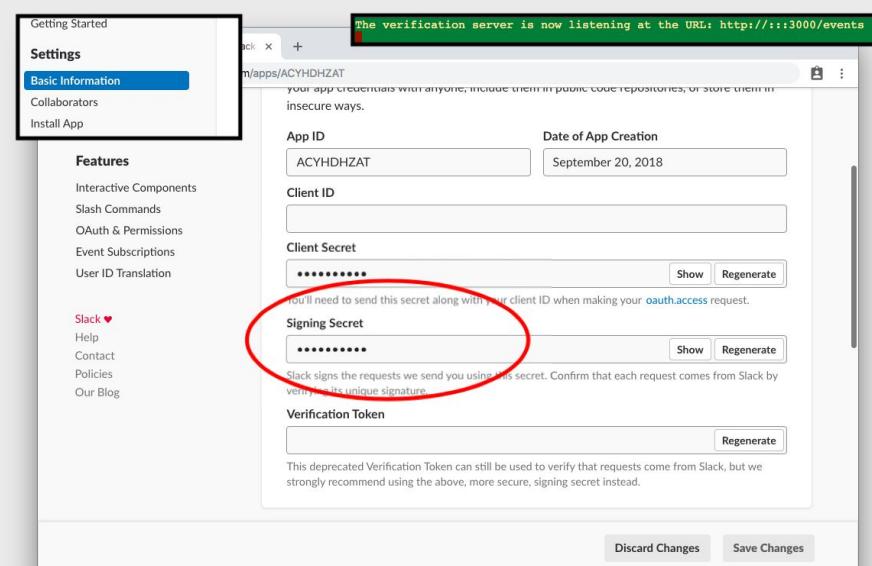
Verifying Your Identity to the Slack API

We need to get credentials from the Slack API to verify our identity.

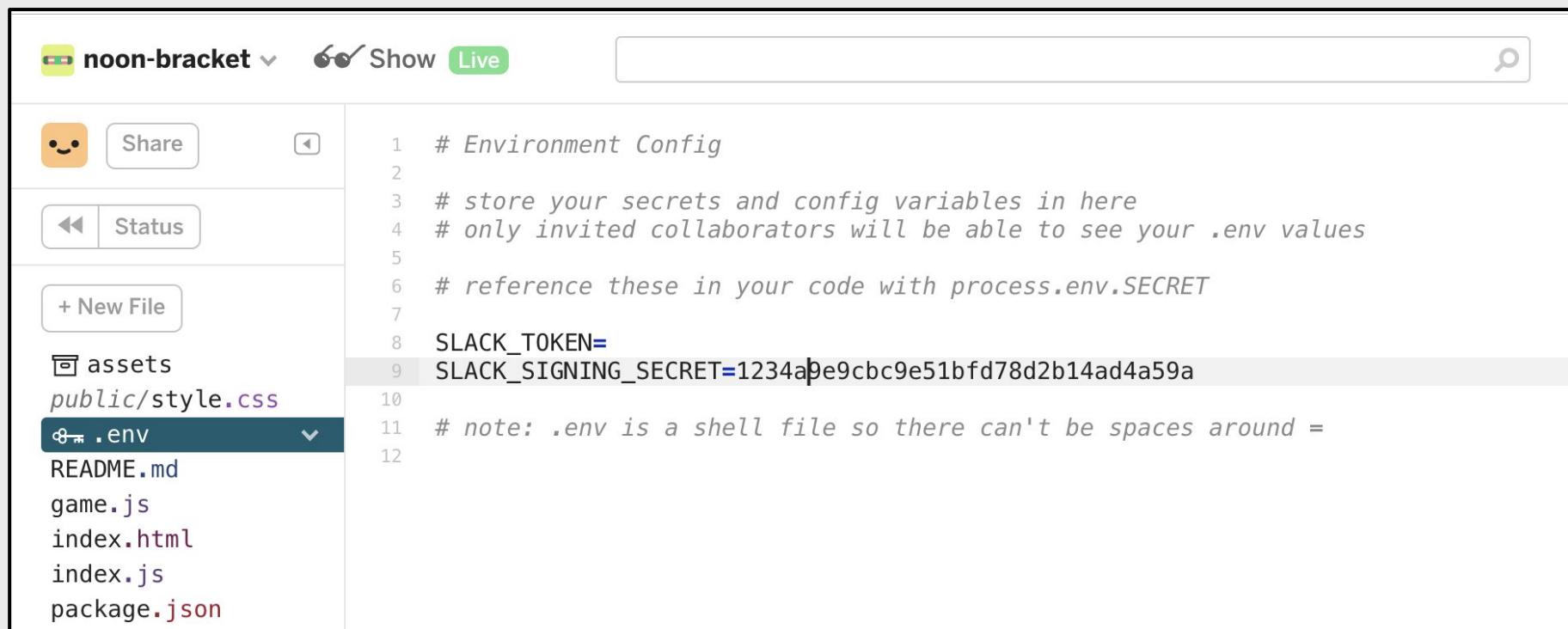
1. Navigate to the Basic Information panel in your Slack App dashboard.

2. Find your app's Signing Secret. Click **Show**. Copy the token.

3. We need to use this secret to send our calls to the Slack API. Return to your project on Glitch.



Verifying Your Identity to the Slack API



The screenshot shows a Glitch project interface. At the top, it says "noon-bracket" with a dropdown arrow, "Show Live", and a search icon. On the left, there's a sidebar with icons for file operations like Share and Status, and a "+ New File" button. Below that is a list of files: assets, public/style.css, .env (which is selected and highlighted in blue), README.md, game.js, index.html, index.js, and package.json. The main area is a code editor with the following content:

```
1 # Environment Config
2
3 # store your secrets and config variables in here
4 # only invited collaborators will be able to see your .env values
5
6 # reference these in your code with process.env.SECRET
7
8 SLACK_TOKEN=
9 SLACK_SIGNING_SECRET=1234ape9cbc9e51bfd78d2b14ad4a59a
10
11 # note: .env is a shell file so there can't be spaces around =
12
```

4. In your Glitch project, open **.env**.
5. Paste the SECRET you just copied onto Line 9.

Verifying Your Identity to the Slack API

Now that we saved our signing secret to our environment, let's use a verification tool to send that information to the Slack API.

The screenshot shows the Glitch development environment. On the left, there's a sidebar with file navigation and a 'Tools' button highlighted with a red box. The main area displays the contents of the README.md file:

```
Welcome to MLH Localhost: Build Apps for Slack!
Click Show in the header to run your app.

Your Project
← README.md
This file contains information about all the other files in this project!
← index.html
This is just a placeholder file because you'll be interacting with this app in a Slack workspace instead of through your browser!
← game.js
This file manages the logic of the game.
← index.js
```

The screenshot shows the Slack interface with the 'Logs' and 'Console' sections highlighted with red boxes. The 'Logs' section shows the message 'server listening on port 5000'. The 'Console' section also shows the same message.

```
Logs
server listening on port 5000
server listening on port 5000
server listening on port 5000
```

6. In Glitch, click **Tools**, then click **Logs**, then **Console** to open the console.

Verifying Your Identity to the Slack API

7. Enter the following command (all on one line):

```
./node_modules/.bin/slack-verify  
--secret=$SLACK_SIGNING_SECRET  
--port=3000 --path=/events
```

You should see the output below:

```
app@noon-bracket:~ 18:38  
$ ./node_modules/.bin/slack-verify --secret $SLACK_SIGNING_SECRET --port=3000 --path=/events  
The verification server is now listening at the URL: http://:::3000/events
```

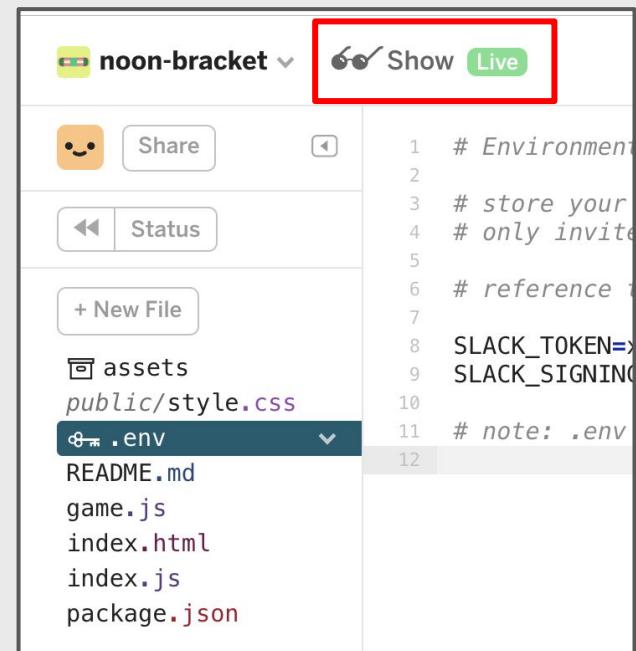
We asked the slack-verify package to run the verification tool. This allows us to prove our identity to the Slack API.

Verifying Your Identity to the Slack API

- Click Show Live.

Your app will open in a new tab.

- Copy the URL of your project from this tab and return to the Slack API dashboard.



A screenshot of a file browser window titled "noon-bracket". At the top right, there is a "Show" button with a checkmark and a "Live" button, both of which are highlighted with a red box. Below the buttons are two small icons: a smiley face and a "Share" button. Underneath these are "Status" and "New File" buttons. The main area shows a list of files and folders:

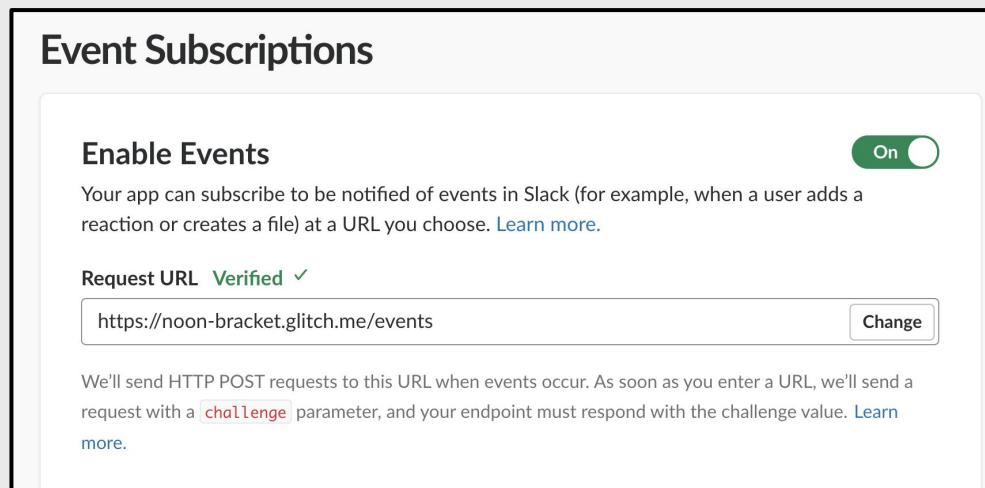
- assets
- public/style.css
- .env
- README.md
- game.js
- index.html
- index.js
- package.json

The ".env" file is currently selected, indicated by a dark blue background.



Verifying Your Identity to the Slack API IV

You started the verification tool, so now you can connect your app to the /events endpoint. We'll explain an endpoint at the end of this section.



10. Navigate to the Event Subscriptions section of your Slack App dashboard.
11. Enable Events. Enter the URL you just copied with /events on the end.
12. When **Verified**✓ appears, the Slack API has verified your identity and now you can make calls to the Slack API!

Verifying Your Identity to the Slack API V

Now that the Slack API has verified your identity, let's stop the verification tool.

```
app@noon-bracket:~ 17:51
$ ./node_modules/.bin/slack-verify --secret $SLACK_SIGNING_SECRET --port=3000 --path=/events
The verification server is now listening at the URL: http://:::3000/events
^C

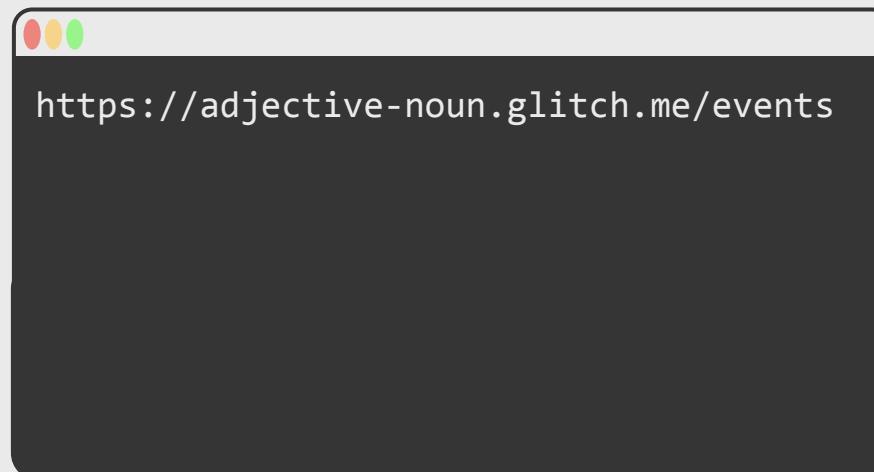
app@noon-bracket:~ 17:54
$ █
```

13. Return to the Glitch console where you started the verification tool.
14. Enter CTRL C to stop the verification tool.

What's an Endpoint?

Communicate using endpoints

- An endpoint is a URL that you use to connect to service on a web server.
- For the Slack API we'll use two endpoints.
- We'll use the `/events` endpoint to listen for events in our Slack workspace.
- We'll use the `/actions` endpoint to send information back to our app and interact with it.



Now, we have to tell the app what events to listen for so that the bot knows what to do.

Select Workspace Events for Our App

There are many different events that happen in a Slack workspace
- people join and leave channels, people send messages. We need to select the events that matter to our app.

The screenshot shows the 'Event Subscriptions' section of the Slack developer console. On the left, there's a sidebar with links: 'Event Subscriptions' (which is active and highlighted in blue), 'Bot Users', 'User ID Translation', 'Slack ❤️', 'Help', 'Contact', 'Policies', and 'Our Blog'. The main area is titled 'Subscribe to Workspace Events' and contains a sub-instruction: 'To subscribe to an event, your app must have access to the related OAuth permission scope.' Below this is a table listing four events:

Event Name	Description	Required Scope
member_joined_channel	A user joined a public or private channel	channels:read or groups:read
member_left_channel	A user left a public or private channel	channels:read or groups:read
message.channels	A message was posted to a channel	channels:history
message.groups	A message was posted to a private channel	groups:history

At the bottom of the table is a button labeled 'Add Workspace Event'.

What are we doing?

- To run our app, we'll want to listen to events that happen within the Slack workspace.
- We want our app to run when a member joins or leaves a group or when they post a message.

Select Workspace Events for Our App

Adding Events

1. Navigate to the Events Subscriptions section of the Slack API dashboard.

2. Click **Add Workspace Event**.

3. Add the following events:

Member_joined_channel

Member_left_channel

Message.channels

Message.groups

The screenshot shows the 'Event Subscriptions' section of the Slack API dashboard. On the left, there's a sidebar with links: 'Event Subscriptions' (which is active), 'Bot Users', 'User ID Translation', 'Slack ♥', 'Help', 'Contact', 'Policies', and 'Our Blog'. On the right, there's a table titled 'Subscribe to Workspace Events' with the following data:

Event Name	Description	Required Scope
member_joined_channel	A user joined a public or private channel	channels:read or groups:read
member_left_channel	A user left a public or private channel	channels:read or groups:read
message.channels	A message was posted to a channel	channels:history
message.groups	A message was posted to a private channel	groups:history

At the bottom of the table is a button labeled 'Add Workspace Event'.

4. Be 100% that you click **Save Changes**.

Configure Interactive Components

Just like we configured events, we also need to configure the interactivity of our app.

1. Navigate to the Interactive Components tab in the Slack App dashboard. Turn on Interactive Components.
2. Just like you did for /events, enter your app url with /actions at the end.
3. Click **Save Changes** at the bottom.

The screenshot shows the 'Interactive Components' section of the Slack App Dashboard. At the top, there's a heading 'Interactive Components'. Below it, a section titled 'Interactivity' has a toggle switch labeled 'On' (which is turned on). A descriptive text states: 'Any interactions with actions, dialogs, message buttons, or message menus will be sent to a URL you specify.' with a 'Learn more.' link. A 'Request URL' input field contains the value 'https://noon-bracket.glitch.me/actions'. A note below says: 'We'll send an HTTP POST request with information to this URL when users interact with a component (like a button or dialog).' Below this is a 'Actions' section with a table header: 'Name', 'Description', and 'Callback ID'. A 'Create New Action' button is located in the first row of the table. At the bottom right are 'Discard Changes' and 'Save Changes' buttons.

Add Scopes to Our App

The events that we selected require certain permissions.

Our app requires a few additional ones.

- There's two permissions we need to add:

`bot`

`chat:write:bot`

- These additional permissions will allow our app to:

- send messages to users – this is how our game will gather questions from a user.
- post results of the game to the channel.

Scopes

Scopes define the API methods this app is allowed to call, and thus which information and capabilities are available on a workspace it's installed on. Many scopes are restricted to specific resources like channels or files.

If your app is submitted to the Slack App Directory, we'll review your reasons for requesting each scope. After your app is listed in the Directory, it will only be able to use permission scopes Slack has approved.

Select Permission Scopes

conversations.app_home:create

Conversations

Direct message any members (already added) conversations.app_home:create
view messages, activity, and files in public channels

Features

Interactive Components

Slash Commands

OAuth & Permissions

Event Subscriptions

User ID Translation

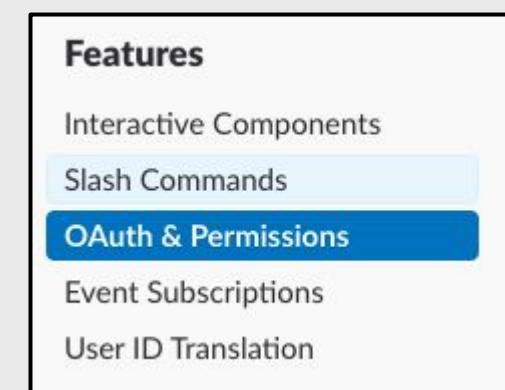
Add Scopes to Our App

1. Navigate to the OAuth & Permissions section of the Slack App dashboard.
2. Scroll down to Scopes.
3. Add the following Permission Scopes:

bot

chat:write:bot

4. Be 100% that you click **Save Changes** before you navigate away from this page.
5. Then, in the yellow bar at the top, reinstall your app.



A screenshot of the Slack OAuth & Permissions 'Scopes' section. It shows a note: 'Scopes define the API methods this app is allowed to call, and thus which information and capabilities are available on a workspace it's installed on. Many scopes are restricted to specific resources like channels or files.' Below this, there's a 'Select Permission Scopes' section with a search bar ('Add permission by scope or API method...'), a 'CONVERSATIONS' section, and two selected scopes: 'Access user's public channels' (with 'channels:history') and 'Access information about user's public channels' (with 'channels:read'). Each selected scope has a delete icon to its right.

Reinstall App

6. Select Authorize.

The screenshot shows the Slack app authorization dialog. At the top, there are two small icons: one for 'localhost' (a blue square with a grid) and one for 'Two Truths and a Lie' (a colorful hexagonal pattern). Below these are two arrows pointing towards each other, indicating a connection or comparison.

On Localhost, Two Truths and a Lie would like to:

Confirm your identity on Localhost

Access information about your channels

Send messages as Two Truths and a Lie

Access content in your public channels

Two Truths and a Lie will be able to access any messages and activity you can see in public channels.

Access content in your private channels

Two Truths and a Lie will be able to access any messages and activity you can see in private channels.

At the bottom are two buttons: 'Cancel' (white background) and 'Authorize' (green background).

Great! We created our app, installed it to our workspace, and saved our slack API credentials to our code.

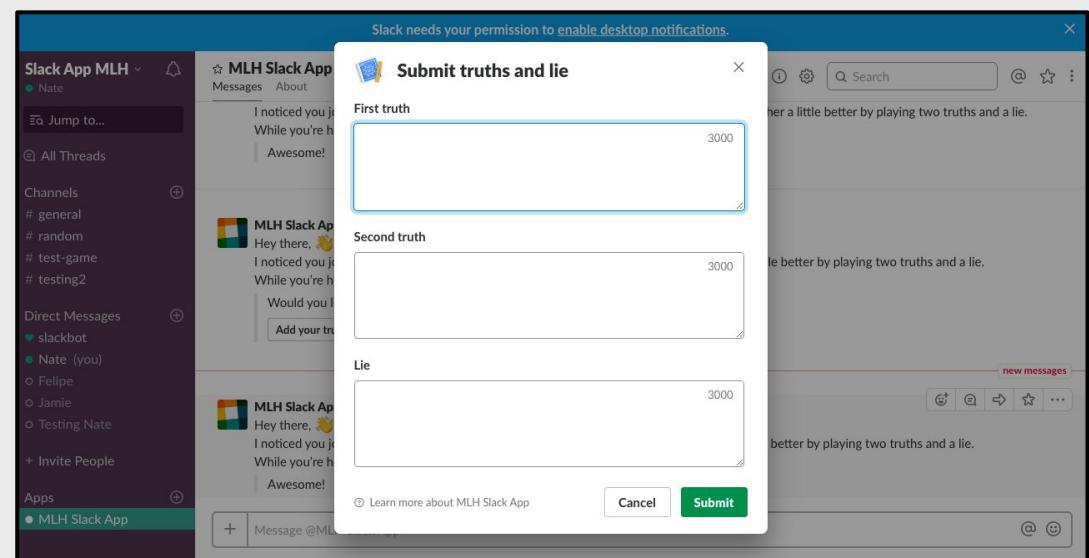
Let's try it out!

Let's Test It Out!

Return to the workspace you joined earlier and see if your app is working!

mlhlocal.host/localhostslack

1. Open the workspace.
2. Join the #twotruths channel
3. Play!



This is where it went wrong! Pay close attention to the next few slides.

Code Review

game.js

```
55      };
56      this.web.dialog.open({trigger_id: "blank_trigger_id", dialog}); // there's a bug here!
57      const msg = payload.original_message,
58      msg.attachments[0].text = 'Awesome!';
59      msg.attachments[0].actions = [];
60      respond(msg);
61  }
```

What's missing?

- In order for the dialog box (pop-up) to open, it waits for a trigger_id, which is sent to the function when certain events or interactions occur between users and your app (like when a user clicks a button). In this case, the trigger_id should be coming from the payload. Let's fix this.

Code Review

Correct Code

Make sure your code looks like the code below, then let's move on!

game.js

```
56      '',
57      this.web.dialog.open({trigger_id: payload.trigger_id, dialog});
58      const msg = payload.original_message;
59      msg.attachments[0].text = 'Awesome!';
60      msg.attachments[0].actions = [];
61      respond(msg);
```

Let's review the rest of the code in this function, then test the app!

Okay, let's test the app!

**You'll have to leave a channel, re-enter,
and play the game again to find the
next bug!**

What Went Wrong?

Two Truths and a Lie APP 3:07 PM

Welcome @Jamie Wittenberg to the team! 🎉

Which do you think is [@Jamie Wittenberg](#)'s lie?

1 My favorite programming language is JavaScript.
2 I love learning new APIs
3 The first code I ever wrote was in Ruby.

1 **2** **3**

Yes! [@Jamie Wittenberg](#)'s lie is, "The first code I ever wrote was in Ruby."

- We fixed the trigger_id bug, and now you were able to play the game.
- But, when you submitted your guess, it revealed your guess to the whole channel, rather than revealing it privately to you!
- What we need is an **ephemeral**.

Key Term

ephemeral: A message that is displayed privately to the user, rather than the whole channel

Code Review

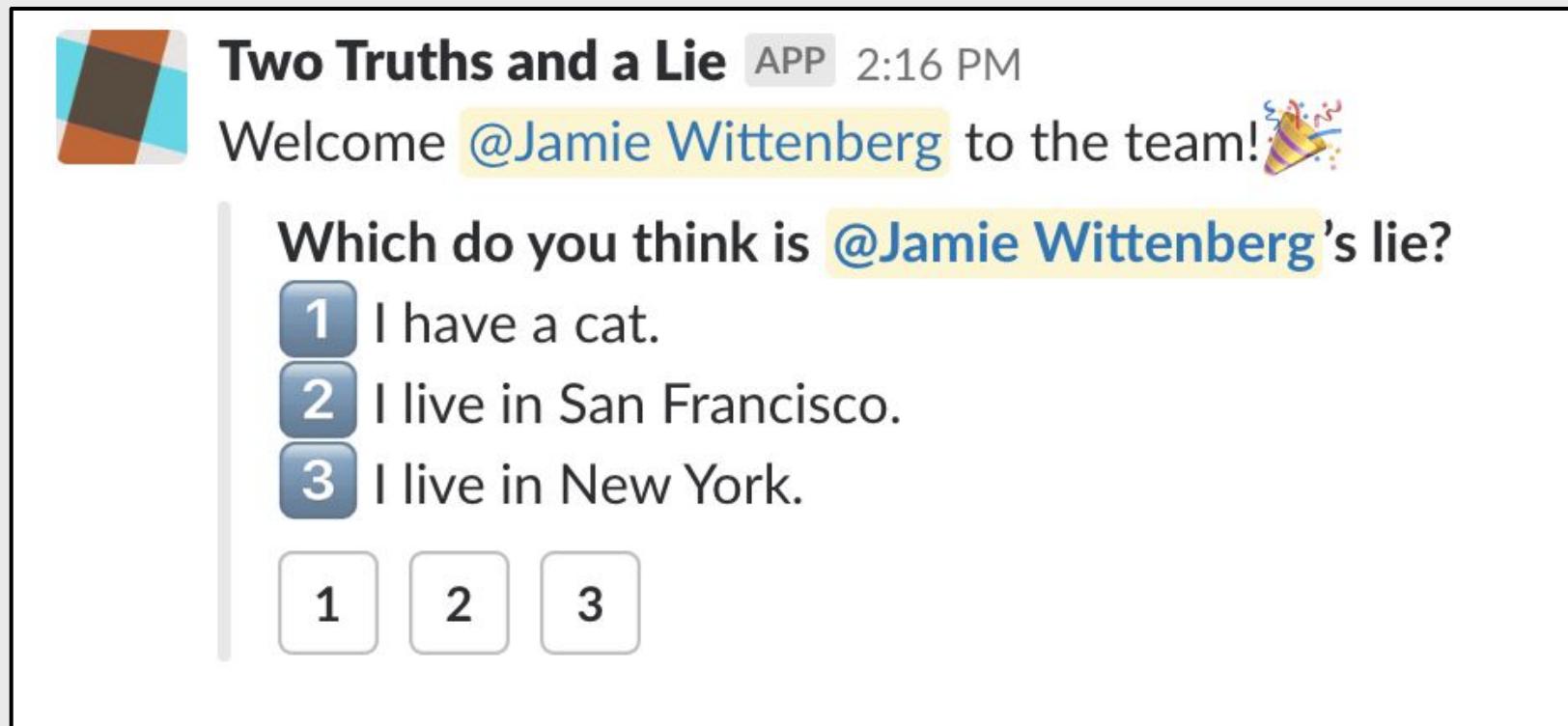
game.js

```
94  broadcastChoices() {
95    const text = `Welcome <@$ {this.userId}> to the team! 🎉`;
96    const attachments = [
97      text: (`*Which do you think is <@$ {this.userId}>'s lie?*` +
98        + '\n:one: ' + this.choices[0]
99        + '\n:two: ' + this.choices[1]
100       + '\n:three: ' + this.choices[2]),
101      fallback: 'You're unable to play 😞',
102      callback_id: 'guess',
103      attachment_type: 'default',
104      actions: [
105        {name: `${this.channelId}/${this.userId}`, type: 'button', value: '0', text: '1'},
106        {name: `${this.channelId}/${this.userId}`, type: 'button', value: '1', text: '2'},
107        {name: `${this.channelId}/${this.userId}`, type: 'button', value: '2', text: '3'},
108      ],
109    ];
110    setTimeout(() => this.revealAnswer(), revealTime);
111    return this.web.chat.postMessage({channel: this.channelId, text, attachments})
112      .then(r => this.broadcastTs = r.ts);
113  }
```

Much like the `promptToPlay()` function, `broadcastChoices()` sends a message to the channel where the game is being played and shows the users Truths and Lies as buttons.

Code Review

Much like the `promptToPlay()` function, `broadcastChoices()` sends a message to the channel where the game is being played and shows the users Truths and Lies as buttons.



A screenshot of a Slack message from the "Two Truths and a Lie" app. The message was sent at 2:16 PM and welcomes @Jamie Wittenberg to the team, accompanied by a small party hat emoji. The main question is "Which do you think is @Jamie Wittenberg's lie?". Three numbered options are listed: 1. I have a cat., 2. I live in San Francisco., and 3. I live in New York. Below each option is a small numbered button (1, 2, or 3) for users to select their answer.

Two Truths and a Lie APP 2:16 PM

Welcome [@Jamie Wittenberg](#) to the team! 🎉

Which do you think is [@Jamie Wittenberg](#)'s lie?

- 1 I have a cat.
- 2 I live in San Francisco.
- 3 I live in New York.

1 2 3

Code Review

game.js

```
115  respondToChoice(payload) {
116    console.log(payload);
117    const num = parseInt(payload.actions[0].value, 10);
118    const text = this.lie === num
119      ? `Yes! <@${this.userId}>'s lie is, "${this.choices[num]}"`
120      : `No, <@${this.userId}> did not lie about "${this.choices[num]}`";
121    console.log(`${payload.user.id} thinks "${this.choices[num]}" is ${this.userId}'s lie in ${this.channelId}`);
122    this.web.chat.postMessage({channel: this.channelId, text, user: payload.user.id});
123 }
```

The `respondToChoice()` function is supposed to secretly tell the player if their guess is correct. **Which part of the code posts that information?**

- Only visible to you

No, [@Jamie Wittenberg](#) did not lie about "I live in New York."

Code Review

game.js

```
115  respondToChoice(payload) {
116    console.log(payload);
117    const num = parseInt(payload.actions[0].value, 10);
118    const text = this.lie === num
119      ? `Yes! <@${this.userId}>'s lie is, "${this.choices[num]}"`
120      : `No, <@${this.userId}> did not lie about "${this.choices[num]}`";
121    console.log(`#${payload.user.id} thinks "${this.choices[num]}" is ${this.userId}'s lie in ${this.channelId}`);
122    this.web.chat.postMessage({channel: this.channelId, text, user: payload.user.id});
123 }
```

Line 122. However, Line 122 says `this.web.chat.postMessage`. Let's change that to `this.web.chat.postEphemeral`.

⌚ Only visible to you

No, [@Jamie Wittenberg](#) did not lie about "I live in New York."

Code Review

game.js

```
115  respondToChoice(payload) {  
116    console.log(payload);  
117    const num = parseInt(payload.actions[0].value, 10);  
118    const text = this.lie === num  
119      ? `Yes! <@${this.userId}>'s lie is, "${this.choices[num]}"`  
120      : `No, <@${this.userId}> did not lie about "${this.choices[num]}"`;  
121    console.log(`${payload.user.id} thinks "${this.choices[num]}" is ${this.userId}'s lie in ${this.channelId}`);  
122    this.web.chat.postMessage({channel: this.channelId, text, user: payload.user.id});  
123  }
```

Correct Code

Let's test your app again!

⌚ Only visible to you

No, [@Jamie Wittenberg](#) did not lie about "I live in New York."

Code Review

game.js

```
125v revealAnswer() {
126    const text = `Welcome <@${this.userId}> to the team! 🎉`;
127    const choices = this.choices.map((c, i) => i === this.lie ? `*${c}*` : c);
128v     const attachments = [
129        text: (`*<@${this.userId}>'s lie is ${this.lie + 1}*` +
130            + '\n:one: ' + choices[0]
131            + '\n:two: ' + choices[1]
132            + '\n:three: ' + choices[2]),
133        callback_id: 'revealed',
134        attachment_type: 'default',
135        actions: [],
136    ];
137    console.log(`Revealed answer for ${this.userId} on ${this.channelId}`);
138    this.web.chat.update({channel: this.channelId, text, attachments, ts: this.broadcastTs});
139 }
140 }
```

After a certain amount of time, the game will reveal the answer to everyone in the channel!

**Once you've tested your app, read on to
learn how the rest of the app works,
and for more practice!**

index.js

index.js contains the code that creates the game server, requires the packages necessary for the game to work, and makes requests to the Slack Events and Actions API endpoints.

Let's dive into the code!

```
1  const http = require('http');
2  const express = require('express');
3  const { createEventAdapter } = require('@slack/events-api');
4  const { createMessageAdapter } = require('@slack/interactive-messages');
5  const { WebClient } = require('@slack/client');
6  const Game = require('./game');
7
8  const port = 5000;
9
10 const web = new WebClient(process.env.SLACK_TOKEN);
11 const slackEvents = createEventAdapter(process.env.SLACK_SIGNING_SECRET, {includeBody: 1});
12 const slackInteractions = createMessageAdapter(process.env.SLACK_SIGNING_SECRET);
13
14 // /** @type {Record<any, Record<any, Game>>} */
15 const games = {};
16
17 slackEvents.on('error', console.error);
18 slackEvents.on('message', (evt, body) => {
19   try {
20     if (evt.subtype === 'channel_join') {
21       console.log(`#${evt.channel} joined ${evt.user}`);
22       games[evt.channel] || (games[evt.channel] = {});
23       const game = games[evt.channel][evt.user] = new Game(web, evt.user, evt.channel);
24       game.promptToPlay();
25     }
26   } catch (e) {
27     console.error(e);
28   }
29 });
30
31 slackInteractions.action('init', (payload, respond) => {
32   try {
33     const channelId = payload.actions[0].name;
34     const game = games[channelId][payload.user.id];
35     if (payload.actions[0].value === 'yes') {
36       game.accept(payload, respond);
37     } else {
38       game.decline(payload, respond);
39       delete games[channelId][payload.user.id];
40     }
41   } catch (e) {
42     console.error(e);
43   }
44 });


```

Code Review

index.js

```
1  const http = require('http');
2  const express = require('express');
3~ const { createEventAdapter } = require('@slack/events-api');
4~ const { createMessageAdapter } = require('@slack/interactive-messages');
5~ const { WebClient } = require('@slack/client');
6  const Game = require('./game');
7
8  const port = 5000;
9
10 const web = new WebClient(process.env.SLACK_TOKEN);
11~ const slackEvents = createEventAdapter(process.env.SLACK_SIGNING_SECRET, {includeBody: 1});
12~ const slackInteractions = createMessageAdapter(process.env.SLACK_SIGNING_SECRET);
```

- **Lines 1-5:** Each of these declarations allows us to use an external JavaScript package.
- **Line 6:** This allows us to use the code written in game.js.
- **Line 8:** Set the localhost port to use for development.

Code Review

index.js

```
1  const http = require('http');
2  const express = require('express');
3~ const { createEventAdapter } = require('@slack/events-api');
4~ const { createMessageAdapter } = require('@slack/interactive-messages');
5~ const { WebClient } = require('@slack/client');
6  const Game = require('./game');
7
8  const port = 5000;
9
10 const web = new WebClient(process.env.SLACK_TOKEN);
11~ const slackEvents = createEventAdapter(process.env.SLACK_SIGNING_SECRET, {includeBody: 1});
12~ const slackInteractions = createMessageAdapter(process.env.SLACK_SIGNING_SECRET);
```

- **Lines 10-12:** Remember earlier when we saved our Slack API token and our Slack Signing Secret to the .env file? These lines retrieve that information and pass it to different functions that allow our app to make calls to the Slack API.

Code Review

index.js

```
17 slackEvents.on('error', console.error);
18
19
20
21
22
23
24
25
26
27
28
29 );
```

```
17 slackEvents.on('error', console.error);
18 slackEvents.on('message', (evt, body) => {
19   try {
20     if (evt.subtype === 'channel_join') {
21       console.log(`#${evt.user} joined ${evt.channel}`);
22       games[evt.channel] || (games[evt.channel] = {});
23       const game = games[evt.channel][evt.user] = new Game(web, evt.user, evt.channel);
24       game.promptToPlay();
25     }
26   } catch (e) {
27     console.error(e);
28   }
29});
```

- **Line 17:** Log any errors to the console.
- **Lines 18-20:** Tells the slackEvents API to listen for the event type channel_join.
- **Line 23:** Create a new Game (the Two Truths and a Lie app).
- **Line 24:** Call the `promptToPlay()` function, which starts the game.
- **Lines 26-28:** Handle errors (we won't cover this code again).

Code Review

index.js

```
31v slackInteractions.action('init', (payload, respond) => {
32v   try {
33v     const channelId = payload.actions[0].name;
34v     const game = games[channelId][payload.user.id];
35v     if (payload.actions[0].value === 'yes') {
36v       game.accept(payload, respond);
37v     } else {
38v       game.decline(payload, respond);
39v       delete games[channelId][payload.user.id];
40v     }
41v   } catch (e) {
42v     console.error(e);
43v   }
44v });

});
```

- **Line 31:** Tells the slackInteractions API to listen for users response to the promptToPlay function().
- **Lines 35-36:** Start the game if the player says yes.
- **Lines 37-38:** Delete the new instance of the game if the player says no.

Code Review

index.js

```
46~ slackInteractions.action({within: 'dialog', callbackId: 'accept'}, (payload, respond) => {
47~   try {
48~     const game = games[payload.state][payload.user.id];
49~     return game.setChoices(payload, respond);
50~   } catch (e) {
51~     console.error(e);
52~   }
53~ });
```

- **Line 46:** Tells the slackInteractions API to listen for the player's choices.
- **Lines 48-49:** Return the player's choices at the end of the setChoices() function.

Code Review

index.js

```
55~ slackInteractions.action({within: 'dialog', callbackId: 'guess'}, (payload, respond) => {
56~   try {
57~     const [channelId, userId] = payload.actions[0].name.split('/');
58~     const game = games[channelId][userId];
59~     game.respondToChoice(payload);
60~   } catch (e) {
61~     console.error(e);
62~   }
63~ });


```

- **Line 55:** Listens for players's guesses
- **Lines 57-59:** Send the player's guess to the respondToChoice() function.

Code Review

index.js

```
67 const app = express();
68 app.use('/events', slackEvents.expressMiddleware());
69 app.use('/actions', slackInteractions.expressMiddleware());
70 app.use(express.static('public'));
71 app.get('/', (req, res) => res.sendFile(__dirname + '/index.html'));
72 app.get('/localhost.mlh.io/activities/build-slack-apps', (req, res) => res.redirect('http://localhost.mlh.io'));
73
74 http.createServer(app).listen(port, () => console.log(`server listening on port ${port}`));
75
```

- **Line 67:** Creates the Express app.
- **Lines 68-69:** Combine both event adapters into one express app.
- **Lines 70-72:** This tells the app to display index.html when someone navigates to the URL of our code/app (but this has nothing to do with how the game is played)
- **Line 74:** Create the server and start the game!

Amazing! Read on for more practice.

Table of Contents

- 1.** Introduction to Apps for Slack
- 2.** Configure and Try the App
- 3.** Review and Update the Code
-  **4.** Review & Quiz
- 5.** Next Steps



Let's recap quickly...

- 1 Slack is an awesome messaging app for companies and organizations
- 2 Slack's API gives you the ability to write apps that talk to workspaces using their SDK
- 3 Slack Apps can use a variety of Slack events in the UI.

Best Practices

This app was designed to create a great experience for users of your workspace. You should consider these best practices when making your own Slack Apps!

Best Practices

1. Apps should give clear instructions for use.
2. Apps allow users to opt out when possible.
3. Add a help function to your app!
4. Plan interactions before you build the app and test them afterwards.
5. Iterate and accept feedback!

What did you learn today?

We created a fun quiz to test your knowledge and see what you learned from this workshop.

<http://mlhlocal.host/quiz>

Table of Contents

1. Introduction to Apps for Slack

2. Configure and Try the App

3. Review and Update the Code

4. Review & Quiz

 **5.** Next Steps



Where to go from here...

- 1 Visit api.slack.com to learn more.
- 2 Join community discussions at api.slack.com/support
- 3 If you build an app you think others will use, you can submit it to the Slack App directory

Keep Learning: Practice Problems for Later

Extra Practice Problem 1: Add this game to your own Slack workspace!

Extra Practice Problem 2: Edit the revealAnswer() function to show a list of how people voted. You could even add an award for participants who get the most answers right!

Keep the Momentum Going!

- 1 Sign up to host this awesome workshop at
<https://localhost.mlh.io/>
- 2 Running continuous workshops earns you points towards your [Momentum Rewards](#)
- 3 Redeem your Rewards Points for custom stickers, t-shirts, and pizza!



Workshop

Build Apps for Slack

MLH localhost

