

devCodeBot Technical Documentation

Updated 6/12/2020

Contents

Overview and functionality.....	3
Front end Bot application	3
Code overview.....	3
Local setup	4
Setting up local environment variables	10
Description of App.js	12

Overview and functionality

devCodeBot is a Slack bot created to assist devCodeCamp students in performing search queries, timeboxing themselves on issues, asking instructors for help, and learning critical time management skills. Additional benefits and outcomes of the bot for the instructors includes logged usage metrics per student, an extensible back-end framework for controlling how the bot interacts with and guides students through problem solving steps, and enforces/bolsters best practices for students seeking help.

Front end Bot application

Code overview

The front-end to the devCodeBot application is a Node.js program under **/devCodeBotSrc**. **package.json** contains the following list of npm dependencies that can be installed to the hosting environment using ``npm install``:

```

1 {
2   "name": "devcodebotsrc",
3   "version": "1.0.0",
4   "description": "Houses Node.js packages for devCodeBot",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "start": "node app.js"
9   },
10  "repository": {
11    "type": "git",
12    "url": "git+https://github.com/aseichter2007/devCodeBot.git"
13  },
14  "author": "",
15  "license": "MIT",
16  "bugs": {
17    "url": "https://github.com/aseichter2007/devCodeBot/issues"
18  },
19  "homepage": "https://github.com/aseichter2007/devCodeBot#readme",
20  "dependencies": {
21    "@slack/events-api": "^2.3.3",
22    "@slack/interactive-messages": "^1.6.0",
23    "@slack/web-api": "^5.10.0",
24    "axios": "^0.19.2",
25    "unix-timestamp": "^0.2.0",
26    "dotenv": "^8.2.0"
27  }
28 }
29

```

The entry point into the application is `app.js`, which contains all the logic for interacting with the bot using various Slack API endpoints. A `.env` file needs to be placed into the root of the application to set up environment variables. Also in this directory are two folders, **BlockKits** and **CrudModal**, which contain pre-formatted payloads which `devCodeBot` uses to respond to specific user actions in the Slack interface.

Local setup

The following steps indicate how to set up the application for local development and testing.

1. Download or pull the code from [GitHub](#) to a local filepath.
2. [Create an App](#) in the `devCodeCamp` workspace.
3. In Basic Information, select "Bots" to enable bot functionality:

Building Apps for Slack

Create an app that's just for your workspace (or build one that can be used by any workspace) by following the steps below.

Add features and functionality

Choose and configure the tools you'll need to create your app (or review all [our documentation](#)).

Incoming Webhooks

Post messages from external sources into Slack.

Interactive Components

Add components like buttons and select menus to your app's interface, and create an interactive experience for users.

Slash Commands

Allow users to perform app actions by typing commands in Slack.

Event Subscriptions

Make it easy for your app to respond to activity in Slack.

Bots

Allow users to interact with your app through channels and conversations. 🌟

Permissions

Configure permissions to allow your app to interact with the Slack API.

4. In the Basic Information panel on the left-hand side of the screen, scroll down to “Display Information” to add branding to the application:

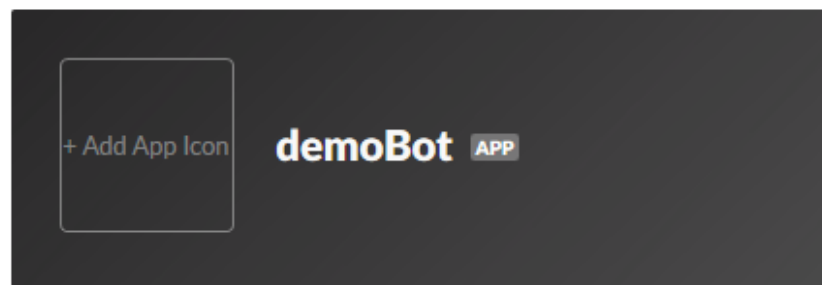
Display Information

This information will be shown in the Slack App Directory and in the Slack App. For more information, view our [App Detail Guidelines](#).

App name

Short description

App icon & Preview



Background color














5. Click "Save Changes" on the bottom of the screen.
6. Click "App Home" in the left pane. Under the section "Your App's Presence in Slack", toggle "Always Show My Bot as Online" to **on**. This improves affordance of the application in Slack by appearing to be always on and available for interactions.
7. Click "OAuth & Permissions" in the left pane and add the following Bot Token Scopes:

Scopes

A Slack app's capabilities and permissions are governed by the [scopes](#) it requests.

Bot Token Scopes

Scopes that govern what your app can access.

OAuth Scope	Description	
app_mentions:read	View messages that directly mention @fowleycodebot in conversations that the app is in	
chat:write	Send messages as @fowleycodebot	
chat:write:public	Send messages to channels @fowleycodebot isn't a member of	
im:history	View messages and other content in direct messages that fowleyCodeBot has been added to	
im:read	View basic information about direct messages that fowleyCodeBot has been added to	
im:write	Start direct messages with people	
mpim:write	Start group direct messages with people	
usergroups:read	View user groups in the workspace	
usergroups:write	Create and manage user groups	
users.profile:read	View profile details about people in the workspace	
users:read	View people in the workspace	

Do not add anything to “User Token Scopes”. This is for having the app perform actions on behalf of a user, which is not a functionality of the bot.

- You will now have the option to Install the app to your workspace by clicking the button at the top of the page:

OAuth Tokens & Redirect URLs

These [OAuth Tokens](#) will be automatically generated when you finish connecting the app to your workspace. You'll use these tokens to authenticate your app.



[Install App to Workspace](#)

Allow permissions for the bot authorization scopes you set; the bot will now appear in the “Apps” section of the devCodeCamp workspace.

9. Click “Event Subscriptions” in the left pane and toggle “Enable Events” to **on**. This enables interactions with Slack’s Events API, which is relied on heavily in devCodeBot’s source code in `app.js`.
10. Open the “Subscribe to Bot Events” panel and add the following Events subscriptions to the application:

Subscribe to bot events

Apps can subscribe to receive events the bot user has access to (like new messages in a channel). If you add an event here, we'll add the necessary [OAuth scope](#) for you.

Event Name	Description	Required Scope	
app_mention	Subscribe to only the message events that mention your app or bot	<code>app_mentions:read</code>	
message.im	A message was posted in a direct message channel	<code>im:history</code>	

These enable event listeners for the conversational workflow in `app.js`.

11. The next step is to create a Request URL endpoint to listen to and handle the requests invoked from the Events API. For local development and testing purposes, we forwarded local port 3000 to the internet using Ngrok. Read on how to set up local port forwarding with Ngrok [here](#).
12. The URL needs to respond to a challenge parameter sent from Slack. There is a built-in mechanism for enabling this interaction from a Node package. To enable it, you need to run a command that sets up a temporary challenge server that listens on the same port you’ve configured your tunnel to forward to (in our case, port 3000).

13. Open a command line and navigate to the root of the Node application (**/devCodeBotSrc**).
14. Run ``npm i`` in your command line to install all Node dependencies in the project (this assumes you have Node.js installed on the machine you are setting this up on).
15. Run the following command to start up the server:

```
.\node_modules\.bin\slack-verify --secret {YOUR_APP_SECRET} [--path=/slack/events] [--port=3000]
```

You can get your app's signing secret from the "Basic Information" tab in your app's homepage.

16. Success should look like the following:

```
c:\dCC\Week_10\devCodeBotSrc>.\node_modules\.bin\slack-verify --secret [REDACTED] [--path=/slack/events] [--port=3000]
The verification server is now listening at the URL: http://:::3000/slack/events
```

17. Leave this terminal open. Back in the Events Subscriptions window of your app, copy your `http://` (**not https://**) Ngrok tunnel address, append ``/slack/events``, on the end, and paste it into the Request URL box. At this point, Slack sends an HTTP POST request to your listener you created in the last step and should return a 200 "success" message with the challenge parameter back to Slack. If everything is set up properly, you should see the following:

Enable Events On

Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. [Learn more.](#)

Request URL Verified ✓

Change

We'll send HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a **challenge** parameter, and your endpoint must respond with the challenge value. [Learn more.](#)

Save your changes at the bottom of the screen.

18. You may now stop the verification server you set up on port 3000.
19. In the application dashboard, click "Interactivity & Shortcuts" in the left pane.
20. Toggle "Interactivity" to **on**.

21. Paste your http ngrok tunnel address in the Request URL, appended with `slack/actions`. This enables interactivity between modal submissions in the application interface using the Slack Interactions API. You should see the following:

Interactivity On

Any interactions with shortcuts, modals, or interactive components (such as buttons, select menus, and datepickers) will be sent to a URL you specify. [Learn more.](#)

Request URL

http://fb239b4980e4.ngrok.io/slack/actions

Slack will send an HTTP POST request with information to this URL when users interact with a shortcut or interactive component.

22. Save your changes at the bottom of the screen.

This completes the app setup for local testing and development. Next, you will need to create and populate a `.env` file in the root of the application directory to provide it with tokens, keys, and environment-specific variables.

Setting up local environment variables

1. In your app's dashboard, click "Basic Information" in the left panel. Scroll down to and make note of everything under App Credentials.
2. Create a file in the `/devCodeBotSrc` root named `.env`. This file should be ignored in `.gitignore`; if it is sent to any public repository with client tokens and secrets, Slack will detect that and disable your application automatically.
3. Add the following to `.env`:

```

1 SLACK_SIGNING_SECRET=
2 SLACK_BOT_TOKEN=xoxb-1
3 SLACK_CLIENT_ID=116
4 SLACK_CLIENT_SECRET=0c3
5 PORT=3000
6 INSTRUCTOR_CHANNEL_ID=C6

```

SLACK_SIGNING_SECRET: this is the Signing Secret value from the App Credentials dashboard.

SLACK_BOT_TOKEN: this is the Bot User OAuth Access Token from the OAuth & Permissions tab in the app dashboard. It will always start with `xoxb-`.

SLACK_CLIENT_ID: the Client ID value from App Credentials.

SLACK_CLIENT_SECRET: the Client Secret value from App Credentials.

PORT: the port you are using for local development and testing. This is the same as whatever you configured ngrok to listen to.

INSTRUCTOR_CHANNEL_ID: This is the ID of the channel you want question cards posted to. You can get the channel ID from the URL after navigating to the target channel in the Slack web interface (it will always start with C).

With this file saved, setup is complete! Make sure the backend API is running, then run the Bot application with either `npm start` or using the built-in Node debugger in VS Code. Output should be as follows:

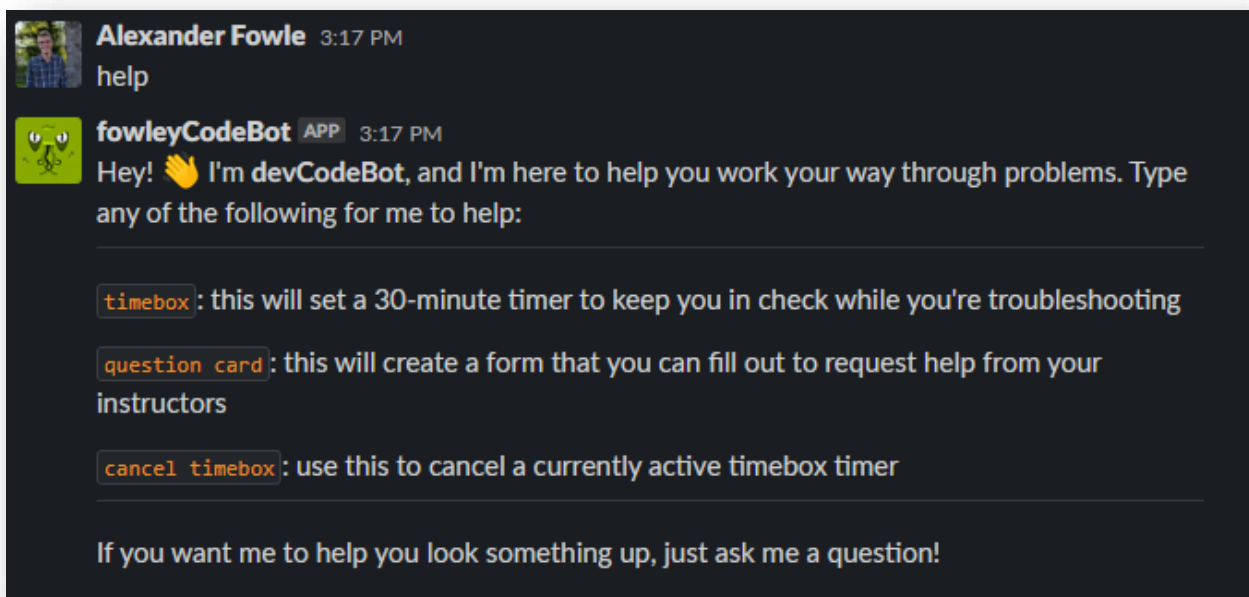
```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Program Files\nodejs\node.exe --inspect-brk=23818 devCodeBotSrc\app.js
Debugger listening on ws://127.0.0.1:23818/89e2b014-b510-45dc-922d-14114a4b2d7f
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
server listening on port 3000

```

In Slack, navigate to your Bot's conversation under Apps, and send it "help" to receive a message back:



Description of App.js

App.js is the program's entry point and hosts all of the logic in handling conversation and commands. It is currently written to leverage the `dotenv` npm package and Node express server to produce a local runtime. It also uses environment variables specified in the .env file to establish connections with the various Slack APIs it makes use of. There is also an import of an npm package called `unix-timestamp` that is used to convert times into the Unix Epoch format, which is a required value for sending delayed messages (implemented in the timebox feature).

`slackEvents.on('message')` is the function that controls how the bot responds to specific commands. A switch case controlled by the text a user sends to the bot performs various actions, with the default case (unrecognized text) treating the text as a search query. For example, if a user sends 'help' to the bot, the bot will perform all the actions in the case 'help' block; in this case, it parses a message saved in the file `./BlockKits/greeting.js` and simply posts the payload back to the user using the `web.chat.postMessage()` method.

The default `case`, which outputs a list of related Google searches, implements an instance of `axios` to invoke various actions on the backend API to format the provided query into something that may be more helpful to the user.

Finally, the functions in **app.js** that implement `slackInteractions` control the flow user-submitted data in the Question Card process. `slackInteractions.action({ actionId: "launchQuestionCardModal" })` creates an instance of the Question Card in a given context, and `slackInteractions.viewSubmission('questionCardSubmit')` parses user input in the modal form fields into a formatted response to send to the instructors per the channel ID specified in the .env file.

`http.createServer(app)` at the bottom of the file is what actually creates the local server on the port specified in the .env file; this block (and anything related to the setup of the Express server) should not be needed in production.