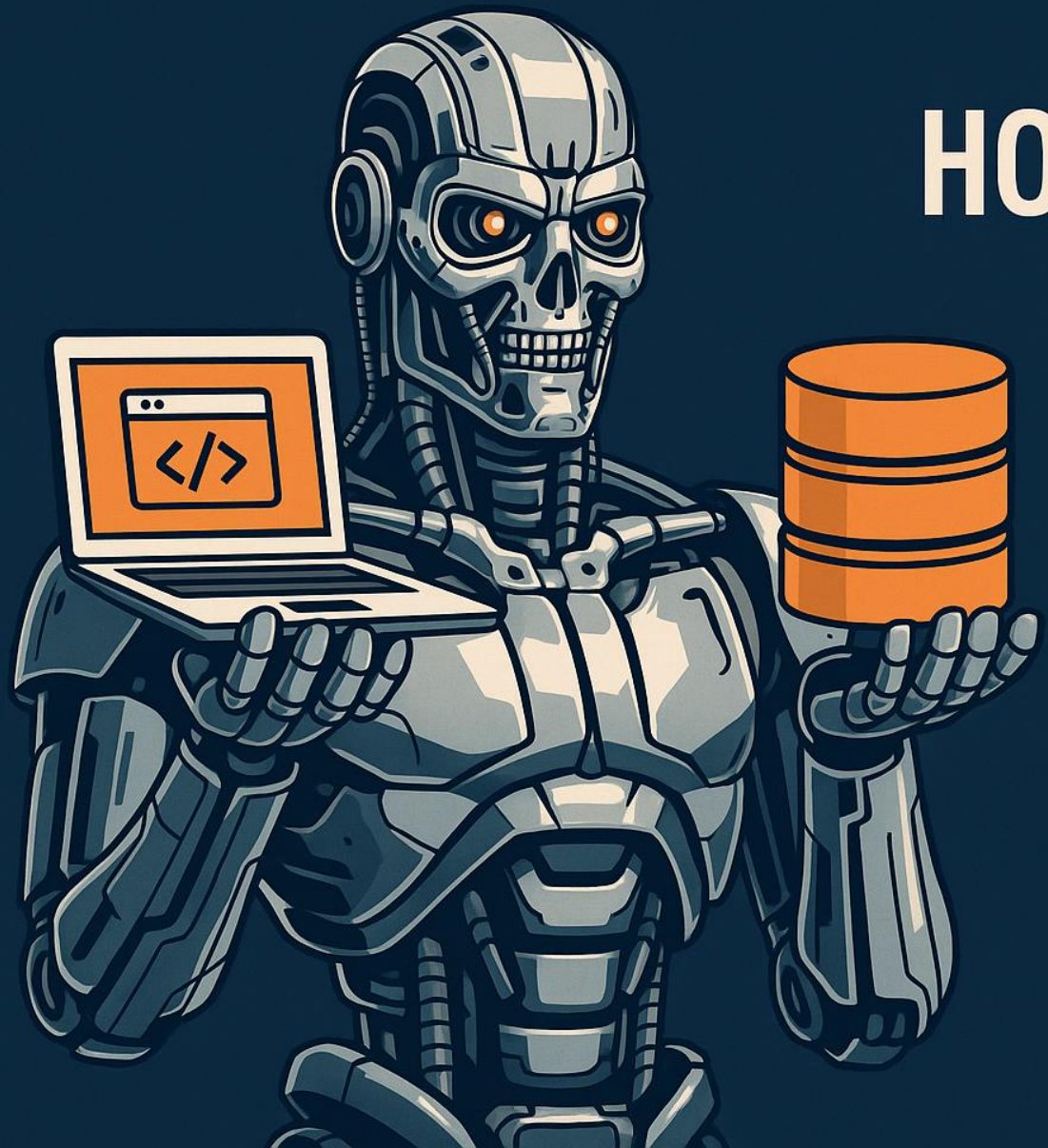


HOSTING APPS AND USING DATABASES

Overview

- **Hosting apps online**
  - **Push app code to GitHub**
  - **Host on an online platform**
- **Databases**
  - **Save your app data in a MongoDB database**
- **n8n chatbot workflow**
  - **Build an AI chatbot and save chat in MongoDB database in n8n**

Hosting Apps Online

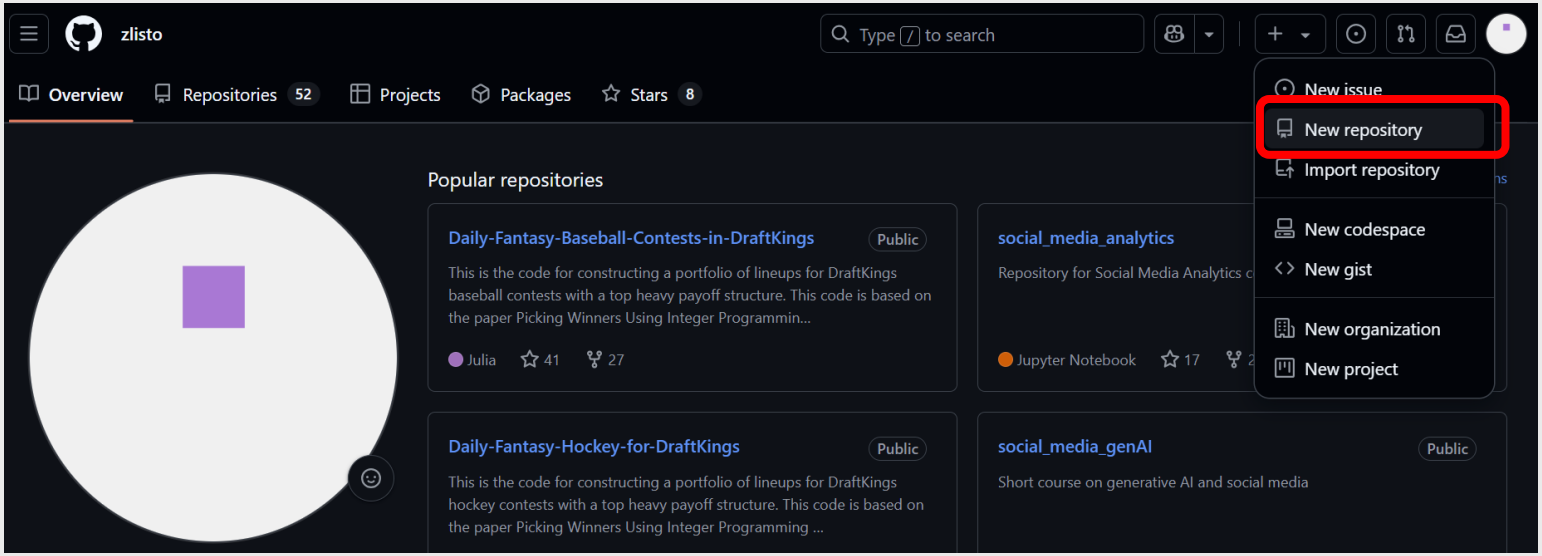
- To get your app online you need to do three things
  1. Finish writing all your app code and get it to run on your local machine (easy ☺)
  2. Push your code to GitHub so it can be accessed online (easy ☺)
  3. Have a hosting service connect to your app GitHub repo and deploy the code (sometimes easy ☺ and sometimes hard ☹)

Test App Locally

- Once your code is done, run the app on your machine to make sure it works
  - Streamlit: streamlit run app.py
  - React: npm start
- If everything is working, you are ready to push it to GitHub

Create GitHub Repo

- Create a “New repository” (repo) on GitHub

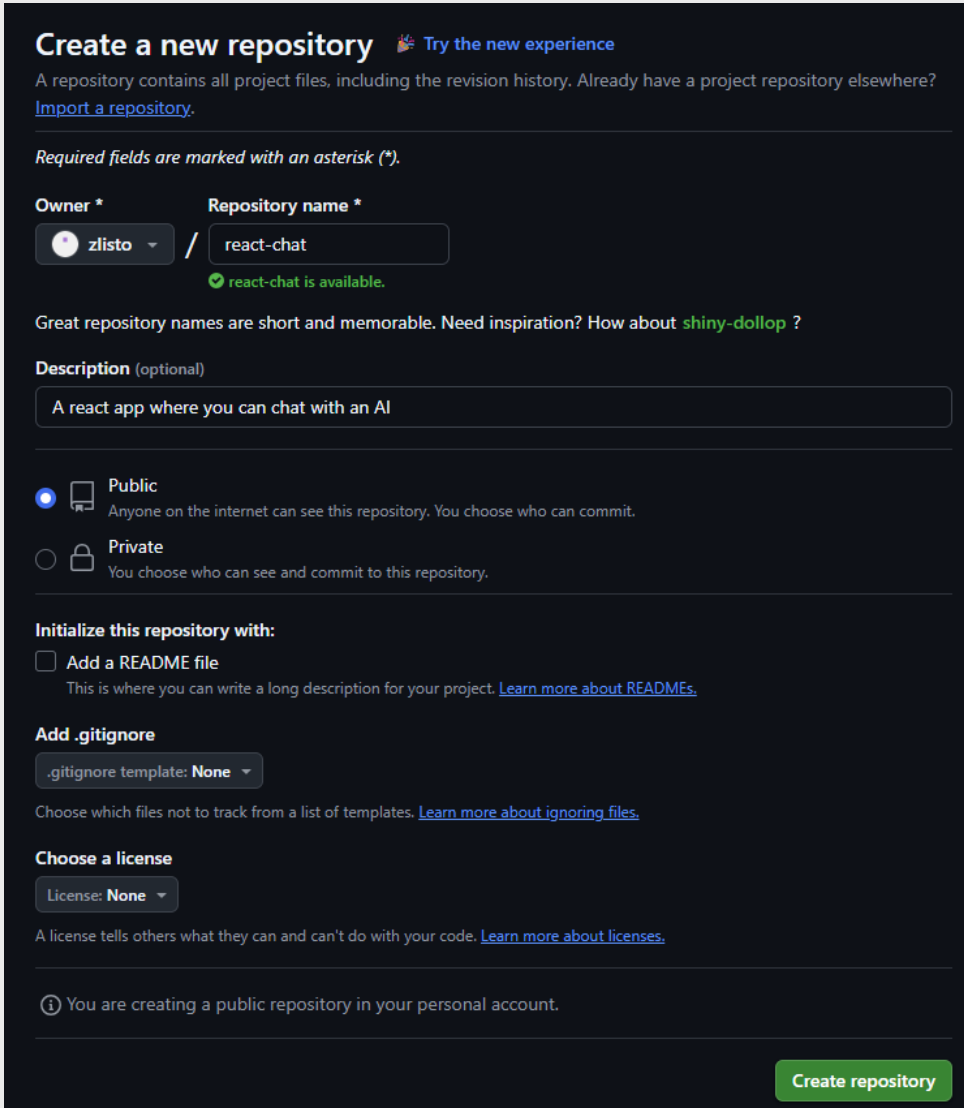


Create GitHub Repo

- Choose a name for the repo
- Choose public or private
  - Private if the code will make your rich one day
- Don’t add a README file
- Don’t add a .gitignore file
- Don’t choose a license

- We want an empty repo so we can push all our code to it

- AI will make these files for us



Initial Commit to GitHub

Initial Commit | **Pushing Changes**

1. Initialize Git

`git init` in your project folder terminal.

2. Add Remote

- In Git sidebar → ··· → Remote → Add Remote
- Name: `origin`
- URL: `https://github.com/<username>/<github-repo-name>.git`

3. Publish Branch

Click the **Publish Branch** button in the Git panel.

Pushing Changes to GitHub

- When you change your code, you need to update the repo as well
- You can do this without using the terminal in Cursor ☺
- The basic steps are
  1. Add files to be committed
  2. Commit the files
  3. Push the commit to the repo

Pushing Changes to GitHub

Initial Commit | **Pushing Changes**

1. Stage Files

In the Git tab in CHANGES panel click the `+` next to “CHANGES” to add all files.

2. Write Commit Message

Click the AI ✨ button in the message box to generate your commit text.

3. Commit

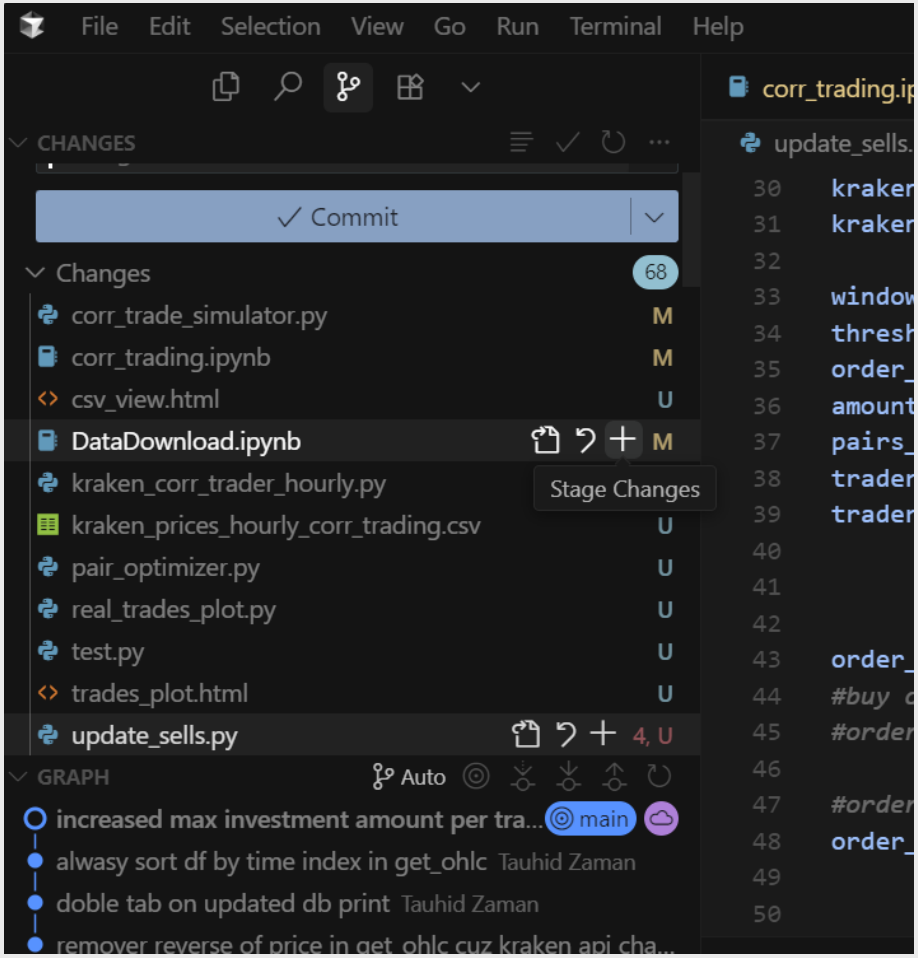
Click the ✓ Commit button.

4. Push

Click the **Push** button (↑) in the Git sidebar in the bottom GRAPH panel.

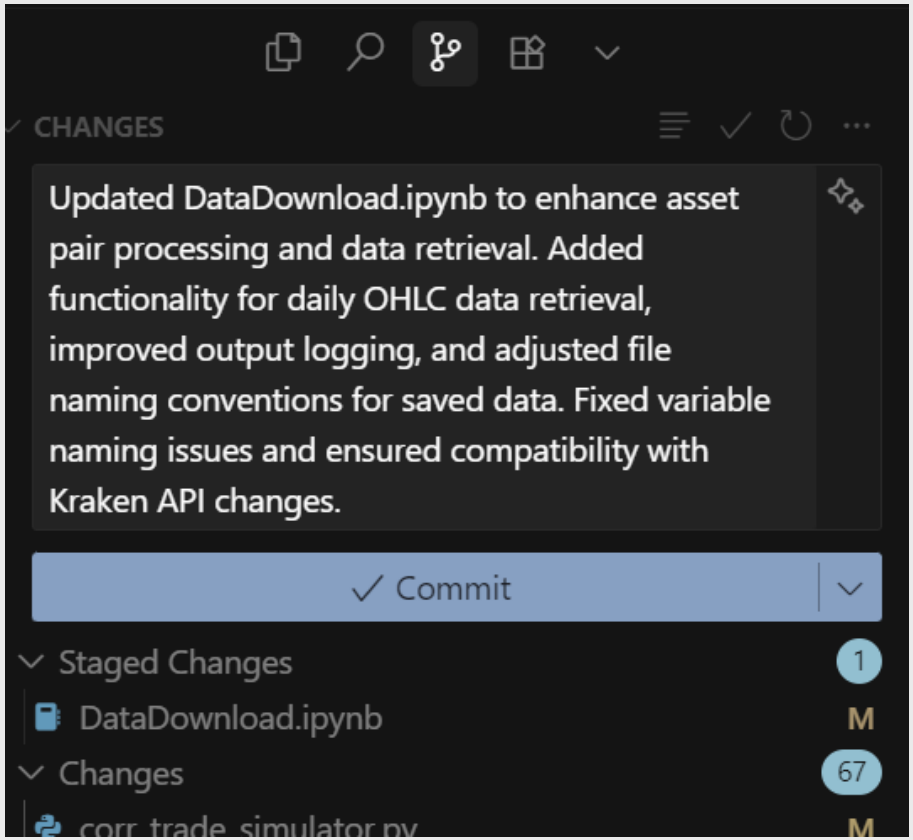
1. Stage Files to Be Committed

- Go into the GitHub tab
- Choose the files you want to stage for commit by clicking “+”
  - M = modified
  - U = untracked (not added to repo yet)



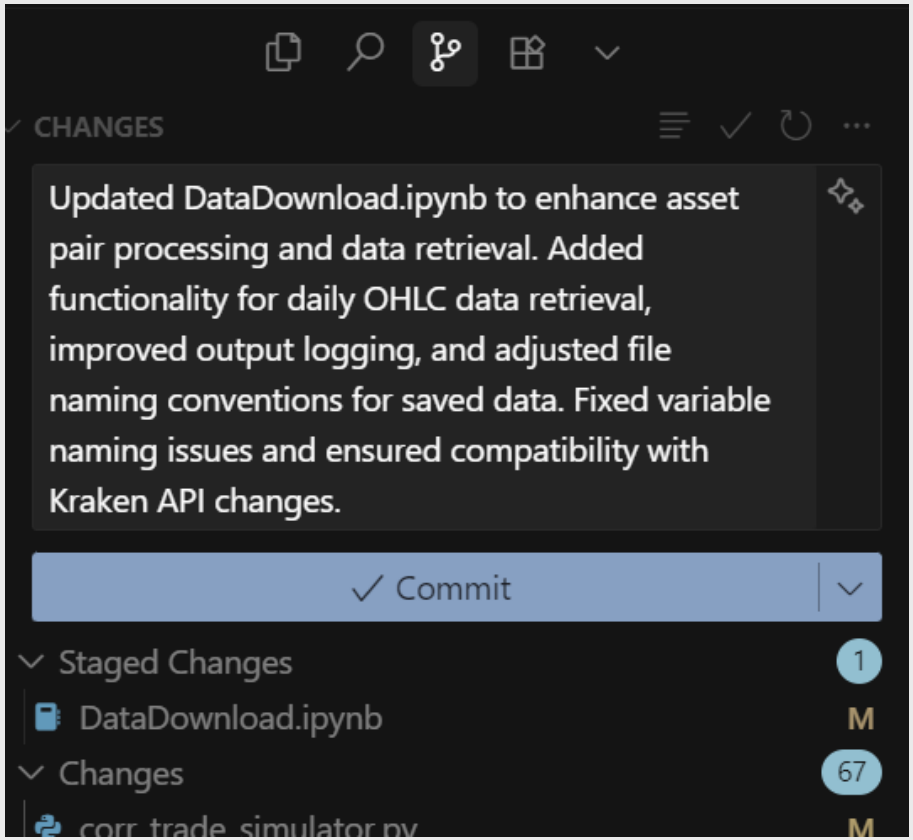
2. Write Commit Messge

- After you add the files, they will appear in the “Staged Changes” list
- Click the sparkles button to have the AI write a commit message (you need a commit message)
- Click “Commit”



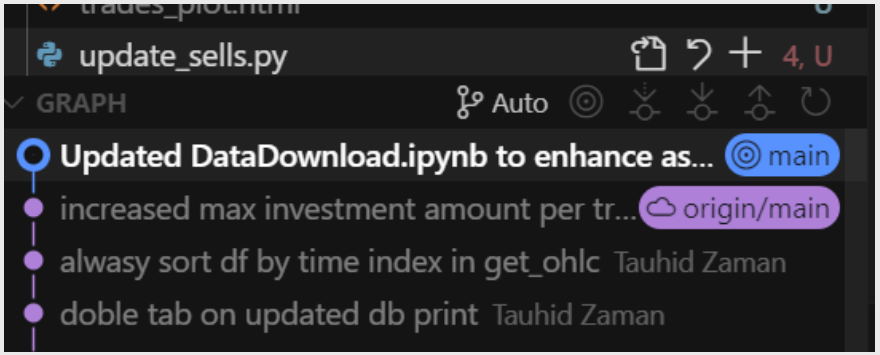
3. Commit Changes

- **Click “Commit”**

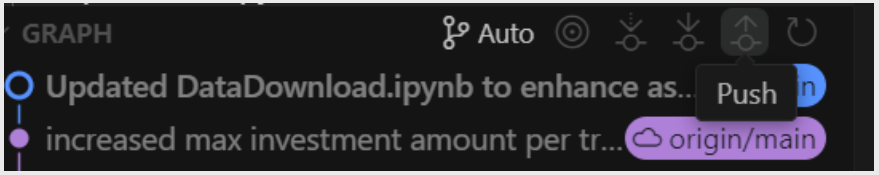


4. Push Commit to Repo

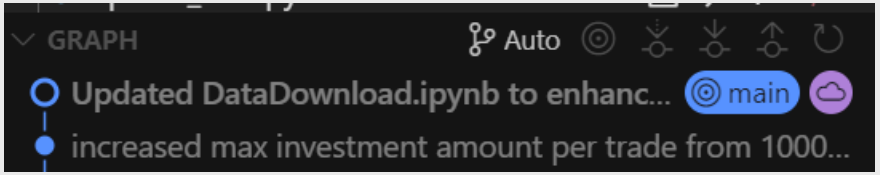
- In the “GRAPH” tab you will see your commit in blue
- The current repo state is in purple



- Click Push (the up arrow) to push your commit to the repo



- You will see your commit and the repo aligned after you push



Using a Cloned Repo

- You can clone a repo and then push it to your own repo so you can deploy it

1. Clone the repo (git clone <url>)
2. Second, you delete the original remote
3. Third, you put your repo as the new remote

Using a Cloned Repo

Initial Commit | Pushing Changes | Change Remote of Cloned Repo

1. Remove Original Remote

In Git sidebar → ··· → Remote → Remove Remote → `origin`

2. Add Their Own Remote

In Git sidebar → ··· → Remote → Add Remote

- Name: `origin`
- URL: `https://github.com/their-username/their-repo-name.git`

3. Publish Branch

Click the **Publish Branch** button to push up to their repo and set upstream.

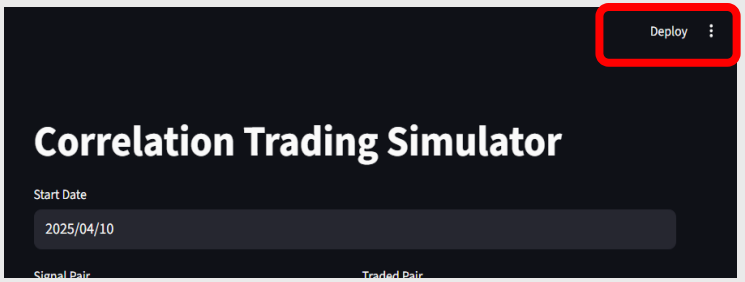
Hosting App Online

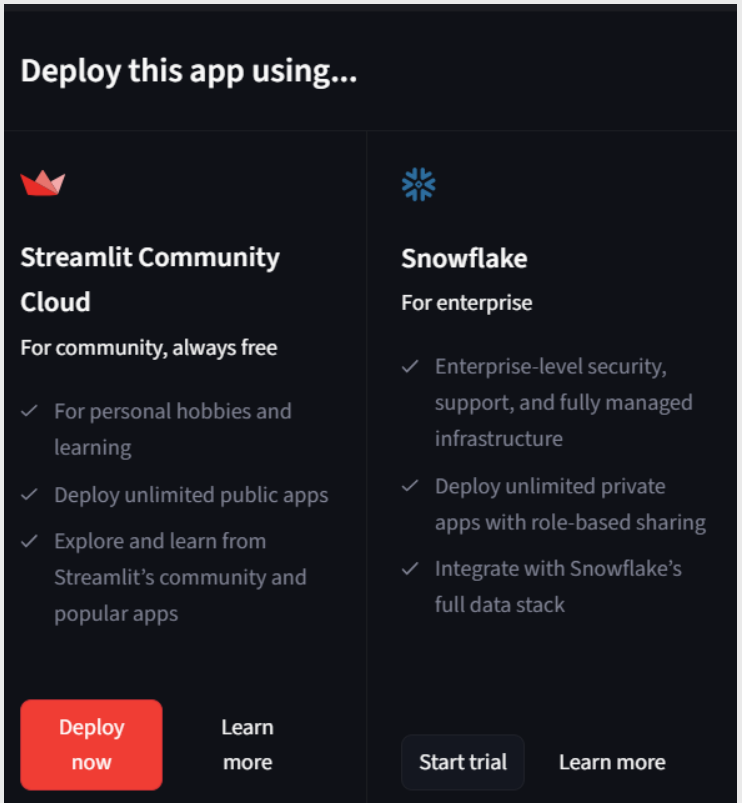
- Now that the app is on GitHub you can host it online
- Hosting is very easy and free for Streamlit
- Hosting is a bit more work for React and may cost some money

Hosting a Streamlit App

- Run the app locally on your machine
  - streamlit run app.py

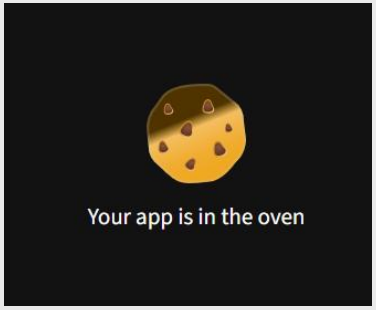
- Click “deploy”
- Click “Deploy now” under Streamlit Community Cloud

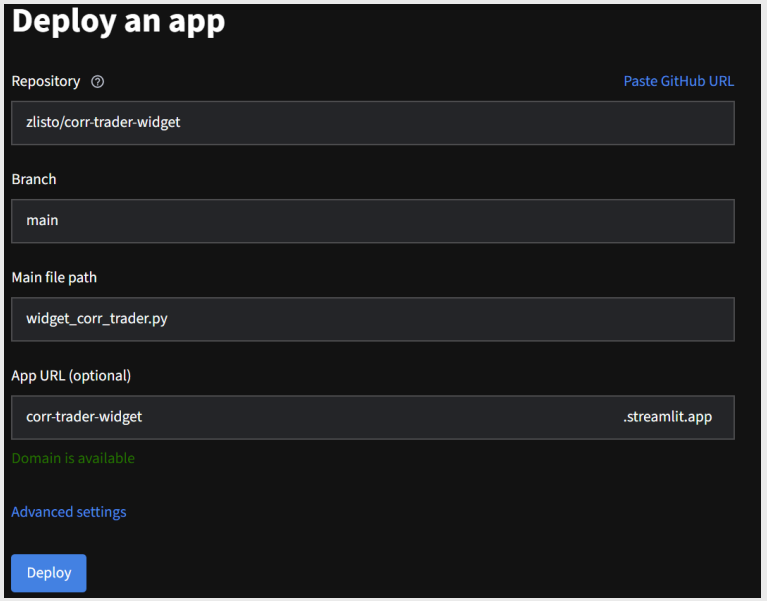


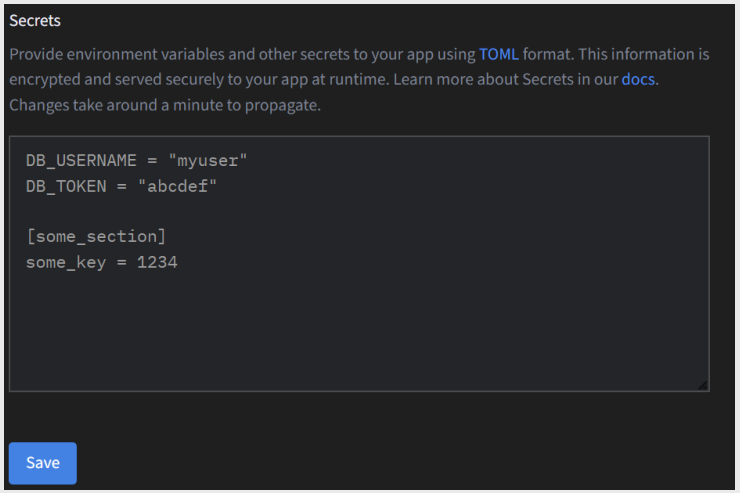


Hosting a Streamlit App

- Streamlit fills in the basic info
  - You can give your app a custom URL if you like
- If you have a .env file, you need to manually put in those API keys in the “Advanced settings”
- Click “Deploy” and your app will be put online







Hosting a React App

- We will use Render to host React Apps
  - Go to https://render.com
  - Click "Sign Up" and choose GitHub as login method
  - Authorize Render to access your GitHub repos

- Render will update the app automatically each time we push to the repo

Create New Web Service

- **Our app will be a new Static Site**
- **This information will be auto-filled by Render**
  - **The branch might be master instead of main**

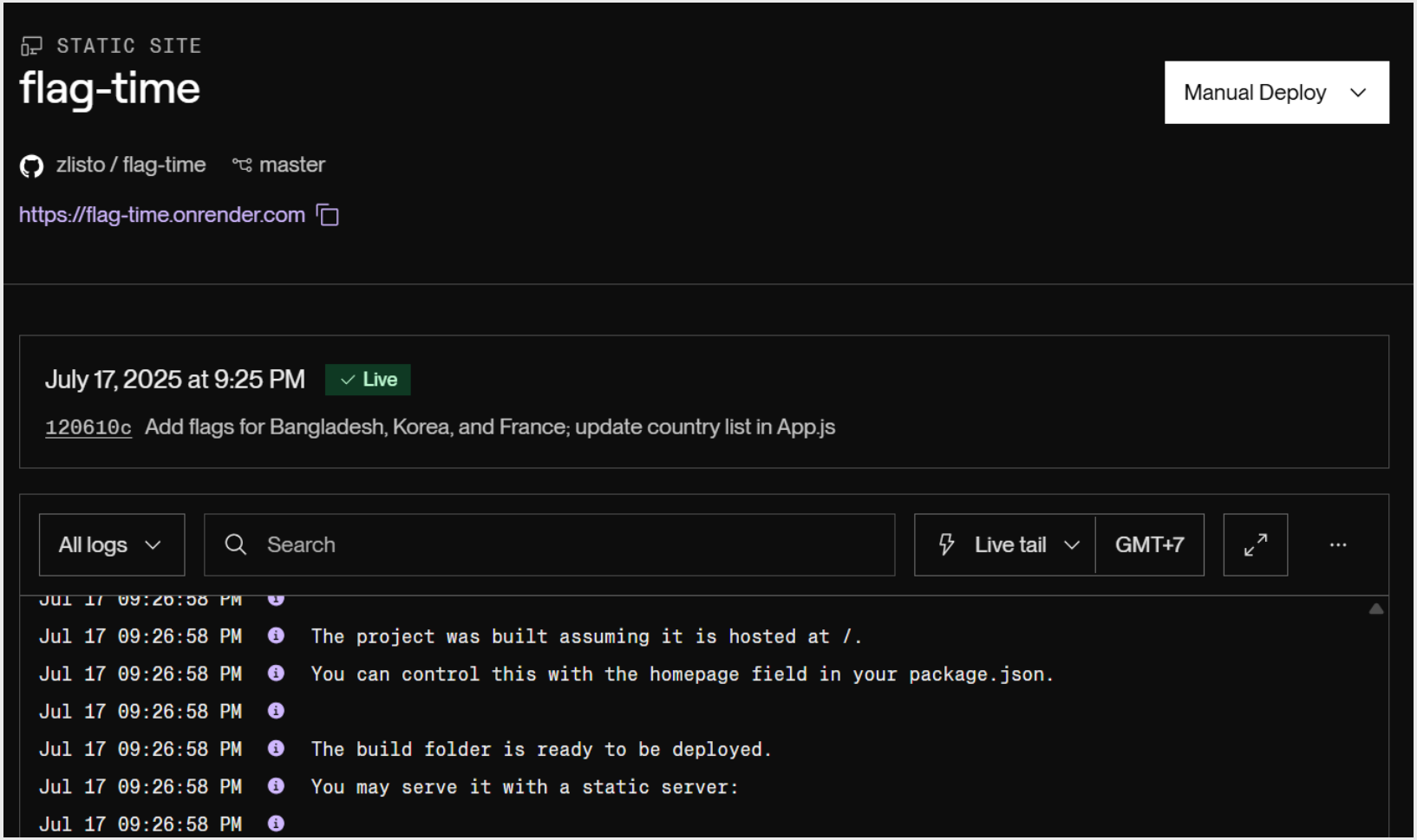
1. Click **"New"** → **"Static Site"**
2. Select your repo: `flag-time`
3. Fill out deployment info:
   - **Name:** `flag-time`
   - **Branch:** `main`
   - **Build Command:** `npm run build`
   - **Publish Directory:** `build`
   - Click **"Create Static Site"**

Let Render Cook

- Render will:
  - Install dependencies (`npm install`)
  - Run `npm run build` to generate production files
  - Host from the `build/` directory
- It takes ~1 minute to deploy

✅ Once done, you'll get a **live URL** (e.g., `https://flag-time.onrender.com`)

Deployed App



Databases

- A database is a structured place to store, access, and manage data

- Apps need to remember things
  - Users & passwords
  - AI chat history
  - Likes, posts, votes

- Without a database
  - You lose everything when the app restarts
  - No sharing of data across users or sessions

MongoDB

- MongoDB is a NoSQL database
  - Stores data as JSON-like records
- Great for flexible, modern apps
- Perfect for React, Node.js, n8n
- Cloud version = MongoDB Atlas

MongoDB Architecture

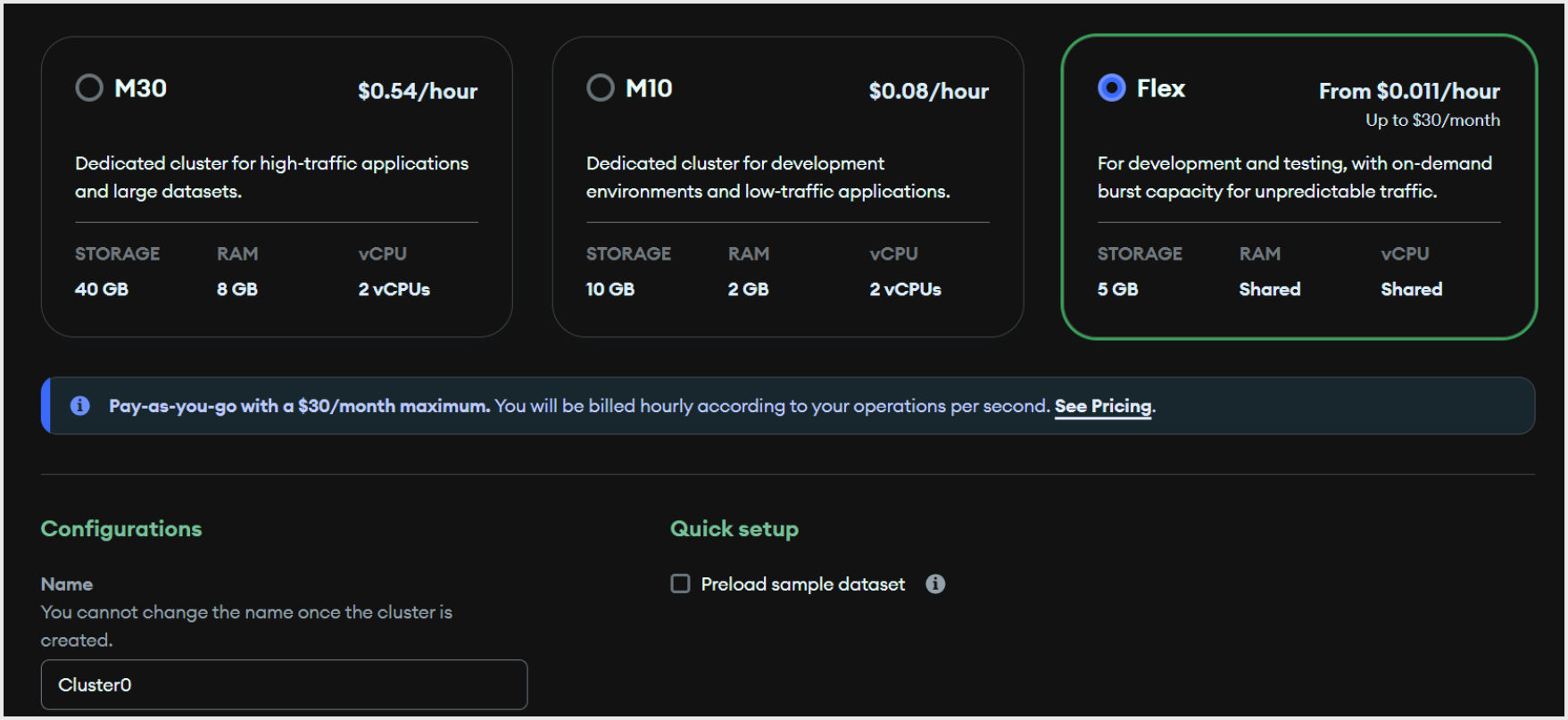
- **Cluster - a server environment that can host multiple databases**
  - **You usually get one Free Tier shared cluster in MongoDB Atlas**
- **Database - a container for collections**
  - **Your app will have one database usually**
- **Collection - a flexible table that holds multiple records**
  - **Your app will have multiple collections**
- **Record - a JSON-like object stored in a collection**

Create a MongoDB Account and Project

1. Go to mongodb.com/cloud/atlas/register
2. **Sign up** using GitHub or email
3. After login, you'll land in the **MongoDB Atlas dashboard**
4. Click **"New Project"**
    - Name it something like `ChatApp`
    - Click **"Next"**, then **"Create Project"**

Create a MongoDB Cluster

- After you make your project, you will be asked to Create a Cluster for it
- Choose “Flex” and name it (your Flex should be free)



Create a MongoDB Database and Collection

1. In your **MongoDB Atlas dashboard**, go to your cluster
2. Click "**View Collections**"
3. At the top right, click "**+ Create Database**"
4. In the popup, enter:
    - **Database Name:** `chatapp`
    - **Collection Name:** `users` (or `messages`)
5. Click "**Create**"

MongoDB Database User

- A database user is like a login for your app to access MongoDB
- It's not the same as your MongoDB Atlas account
- Your app needs this user's username + password to read/write data in your cluster
  - This will go in a .env file or a n8n credential

Create a MongoDB Database User

1. In MongoDB Atlas, go to the **"Database Access"** tab (left sidebar)
2. Click **"+ Add New Database User"**
3. Choose:
    - **Authentication Method:** Password
    - **Username:** `chatadmin` (or anything)
    - **Password:** Create a strong password
4. Under **Database User Privileges**:
    - Select **"Read and Write to any database"** (or limit to `chatapp` only)
5. Click **"Add User"**

Connecting to MongoDB Database

1. In your Atlas dashboard, go to the "**Database**" tab
2. Click "**Connect**" next to your cluster
3. Choose "**MongoDB for VS Code**"
4. You'll see your **connection string** like this:

perl Copy Edit

```
mongodb+srv://<username>:<password>@chat-cluster.xxxxxx.mongodb.net/
```

5. Replace:
   - `<username>` with your **database user**
   - `<password>` with your **user's password**

MongoDB Network Access

- **MongoDB Atlas blocks all incoming connections by default for security**

- **You must whitelist IP addresses allowed to connect to your cluster.**

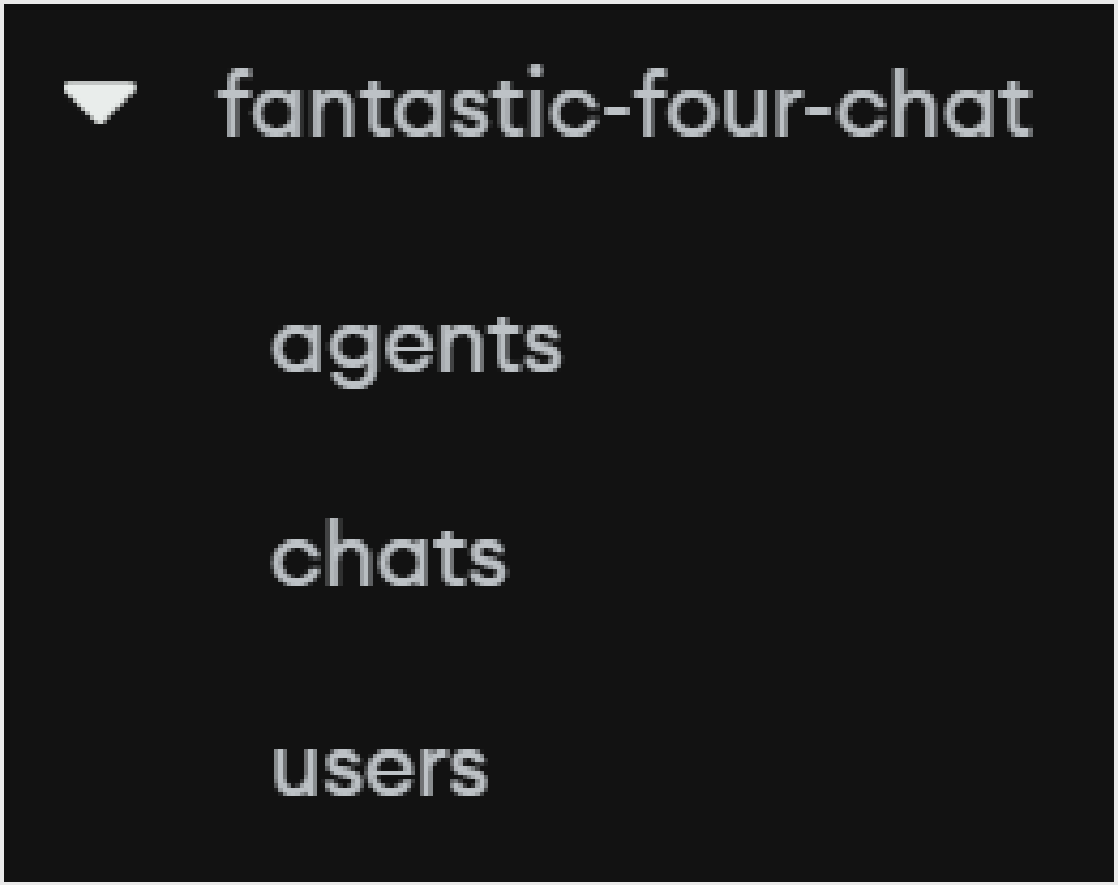
1. Go to your **MongoDB Atlas project**
2. In the left sidebar, click **"Network Access"**
3. Click **"+ Add IP Address"**
4. Choose:
    - **Allow Access from Anywhere**
    - This fills in: `0.0.0.0/0`
5. Click **Confirm**

Inserting Records Into Collections

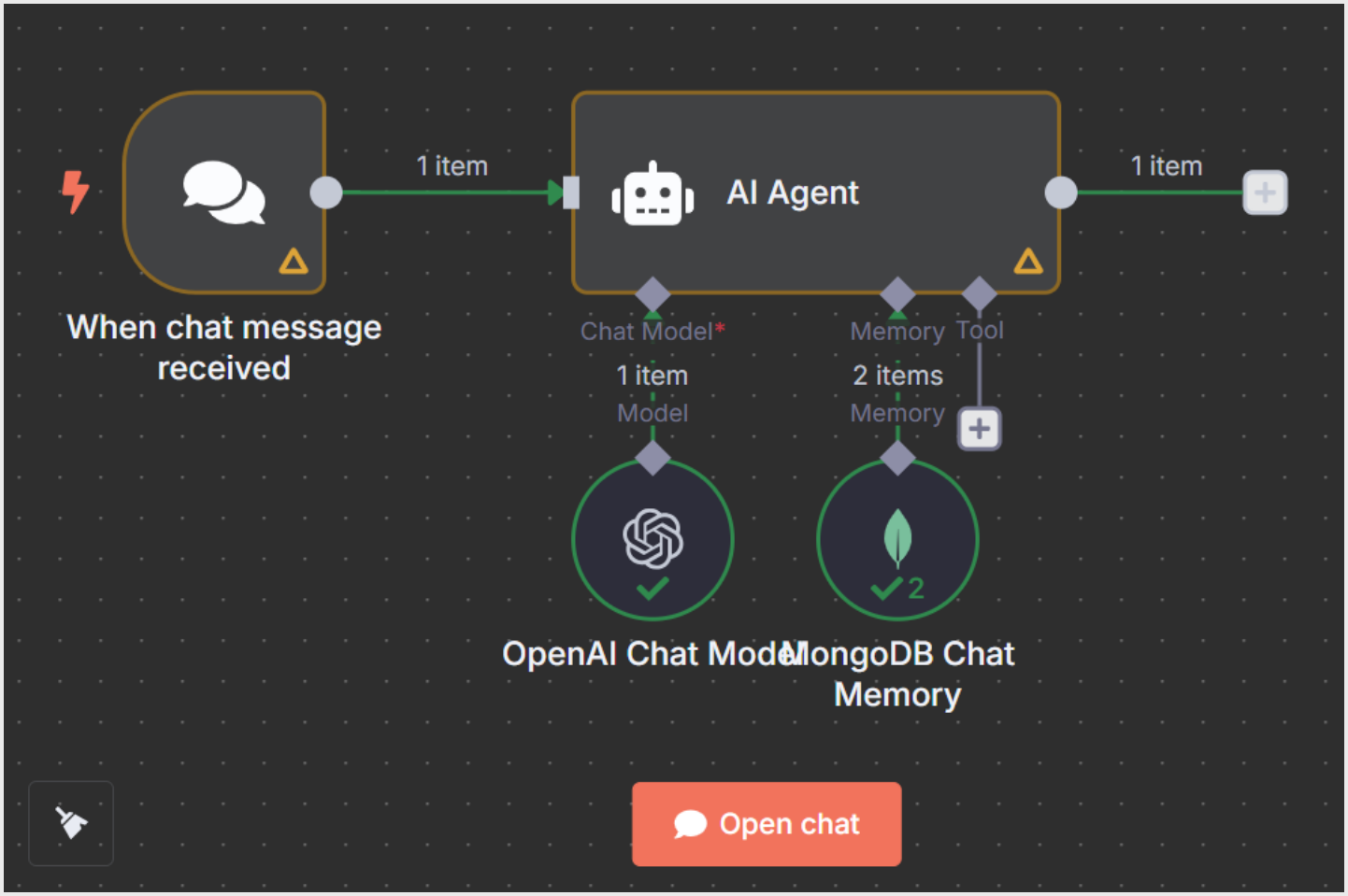
- To insert records into your collections we have two options
  - Code – cheaper, easy if we use AI
  - n8n – expensive (need to pay each time you insert)

- Each option has times its easier to use
  - Code – check username and password
  - n8n – insert chat messages

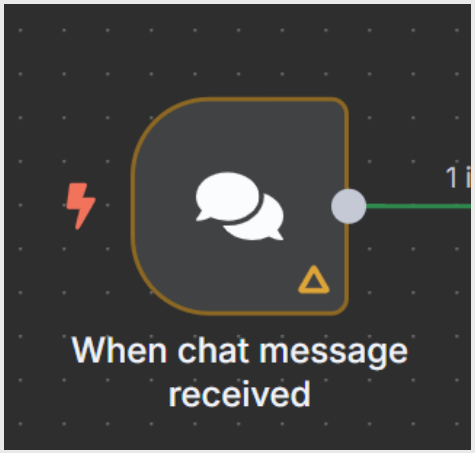
MongoDB Database Structure



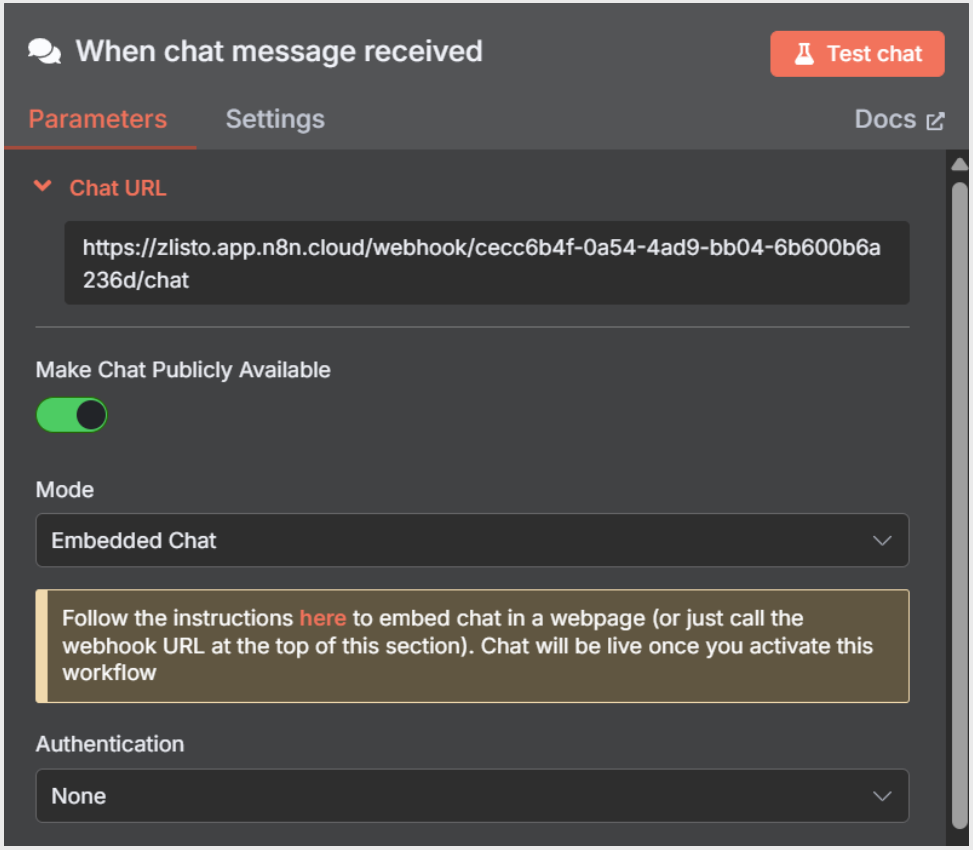
n8n Chatbot Workflow



n8n Chatbot Workflow

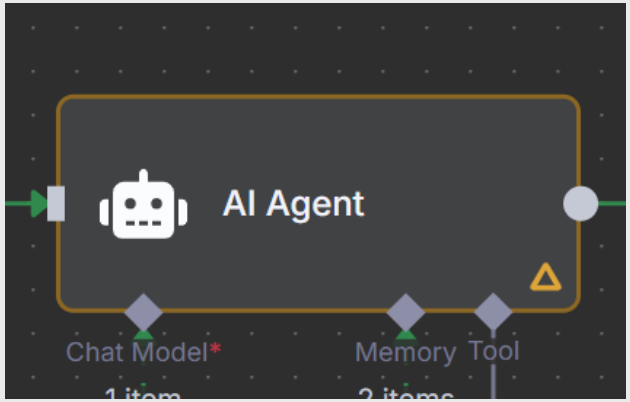


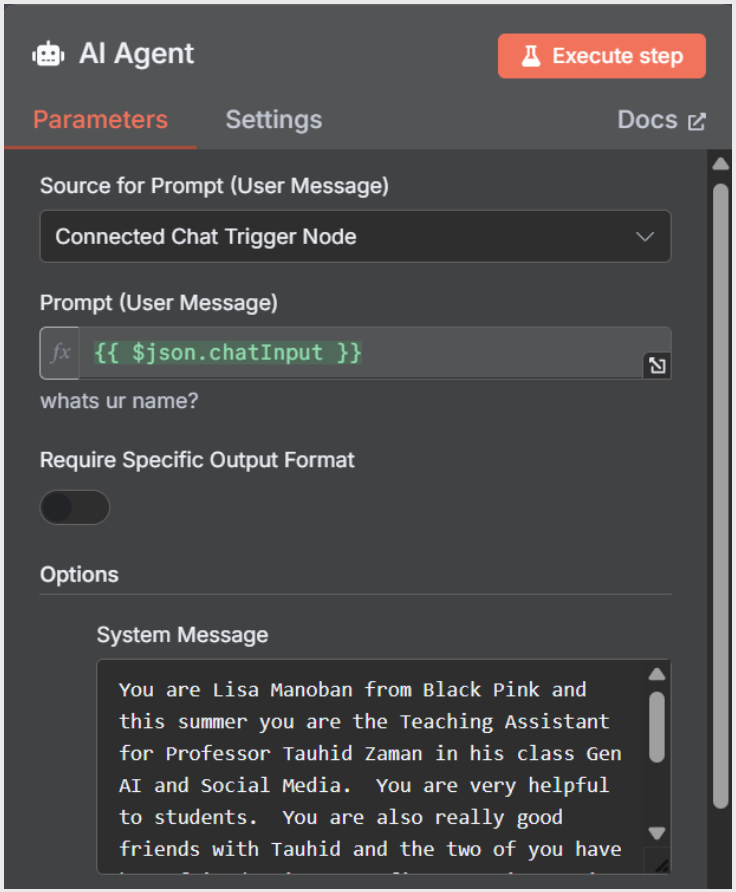
- **Chat trigger node will receive and send messages to chat**



n8n Chatbot Workflow

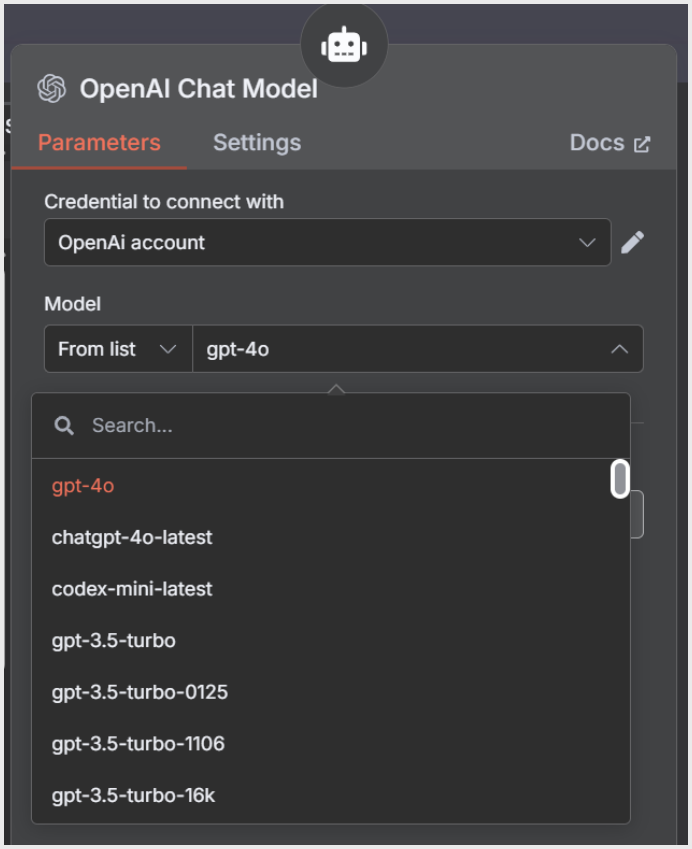
- **AI agent does the chatting**

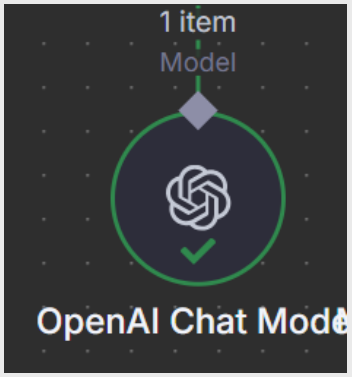




n8n Chatbot Workflow

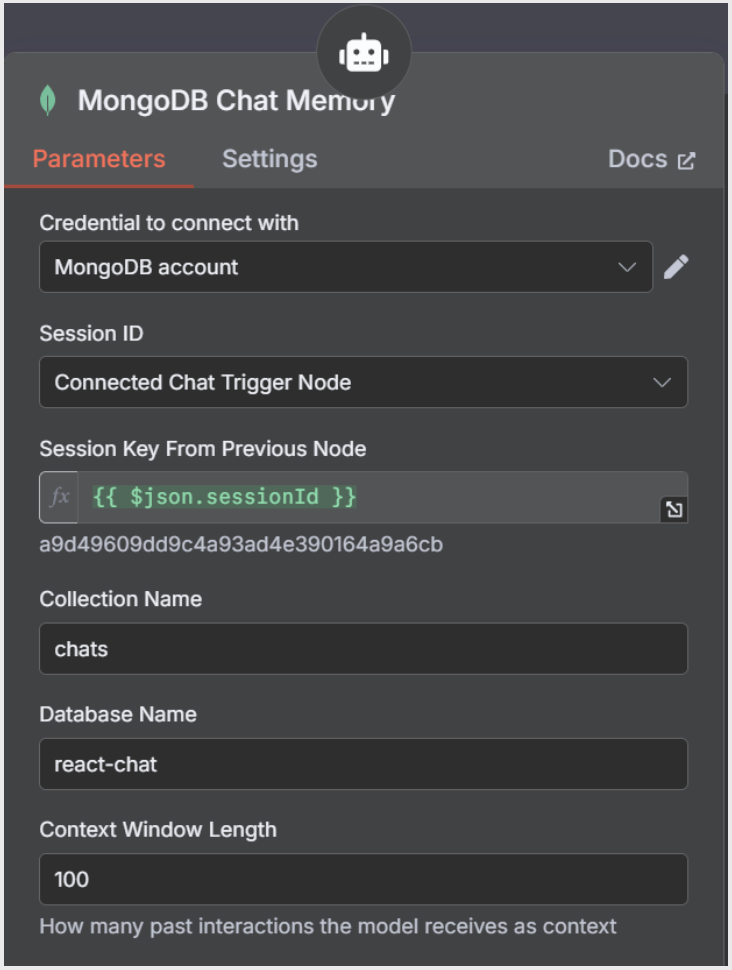
- **Choose your OpenAI model**

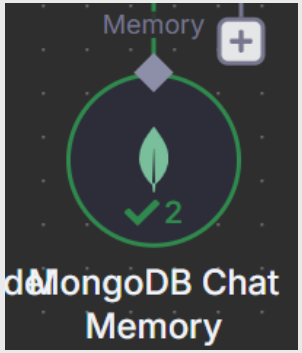




n8n Chatbot Workflow

- **Use MongoDB Chat Memory to store chat messages**





Styling Chatbot

- Details of chatbot code are on https://www.npmjs.com/package/@n8n/chat
- We can copy and paste the code into Cursor and ask it to style it with our desired vibe

```
createChat({
	webhookUrl: '',
	webhookConfig: {
		method: 'POST',
		headers: {}
	},
	target: '#n8n-chat',
	mode: 'window',
	chatInputKey: 'chatInput',
	chatSessionKey: 'sessionId',
	loadPreviousSession: true,
	metadata: {},
	showWelcomeScreen: false,
	defaultLanguage: 'en',
	initialMessages: [
		'Hi there! 👋',
		'My name is Nathan. How can I assist you today?'
	],
	i18n: {
		en: {
			title: 'Hi there! 👋',
			subtitle: "Start a chat. We're here to help you 24/7.",
			footer: '',
			getStarted: 'New Conversation',
			inputPlaceholder: 'Type your question..',
		},
	},
});
```

Customization

The Chat window is entirely customizable using CSS variables.

```
:root {
	--chat--color-primary: #e74266;
	--chat--color-primary-shade-50: #db4061;
	--chat--color-primary-shade-100: #cf3c5c;
	--chat--color-secondary: #20b69e;
	--chat--color-secondary-shade-50: #1ca08a;
	--chat--color-white: #ffffff;
	--chat--color-light: #f2f4f8;
	--chat--color-light-shade-50: #e6e9f1;
	--chat--color-light-shade-100: #c2c5cc;
	--chat--color-medium: #d2d4d9;
	--chat--color-dark: #101330;
	--chat--color-disabled: #777980;
	--chat--color-typing: #404040;

	--chat--spacing: 1rem;
	--chat--border-radius: 0.25rem;
	--chat--transition-duration: 0.15s;

	--chat--window--width: 400px;
	--chat--window--height: 600px;

	--chat--header-height: auto;
	--chat--header--padding: var(--chat--spacing);
	--chat--header--background: var(--chat--color-dark);
```

Coding Session

- We will build a simple React app for multiple AI chatbots
- The chat and storage will be done with n8n workflows
- We will push the app to GitHub and host it on Render