Set Up the Hugging Face API Key

Estimated time needed: 10 minutes

What you will learn

In this lab, you will generate an **API Key** which will allow you to use **Hugging Face** AI models and libraries. Once you obtain your API key, you will learn how to use it with a **Hugging Face** model to generate vector embeddings.

Learning objectives

After completing this lab, you will be able to:

- Generate your own Hugging Face API key
- Use a Hugging Face model with your API key to generating vector embeddings

Prerequisites

- Basic knowledge of Hugging Face
- Intermediate knowledge of Node.js

Important notice about this lab environment

Skills Network Cloud IDE (based on Theia and Docker) is an open-source IDE (Integrated Development Environment) that provides an environment for hands-on labs in course and project-related labs.

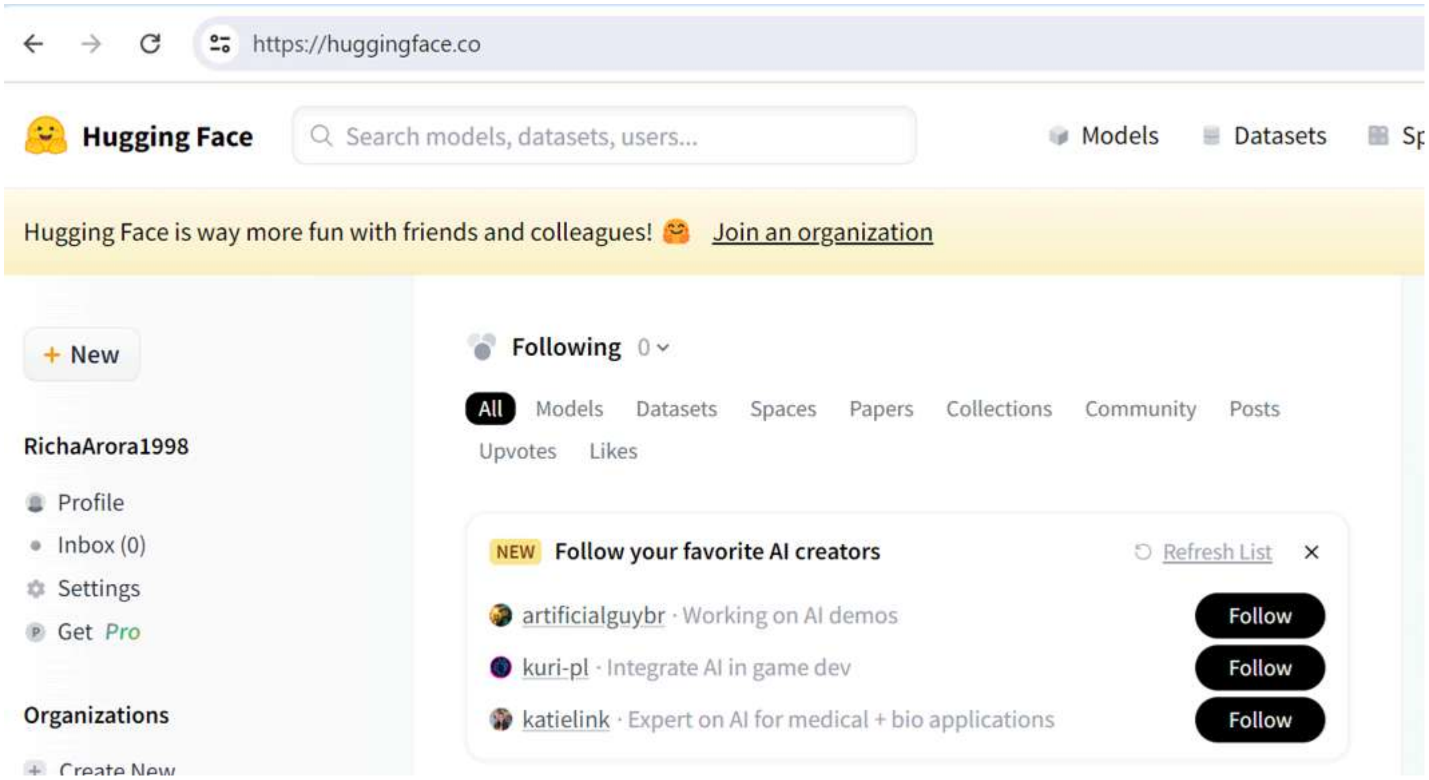
Please be aware that sessions for this lab environment are not persistent. Every time you connect to this lab, a new environment is created for you.

You will lose data if you exit the environment without saving to GitHub or another external source.

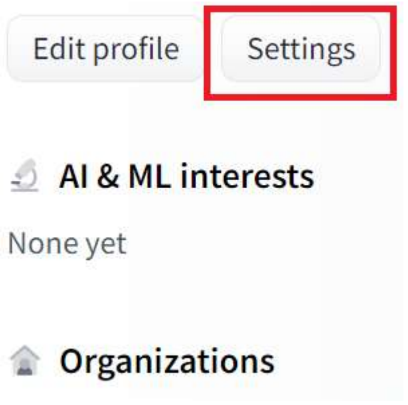
Plan to complete these labs in a single session to avoid losing your data.

Task 1: Create API key

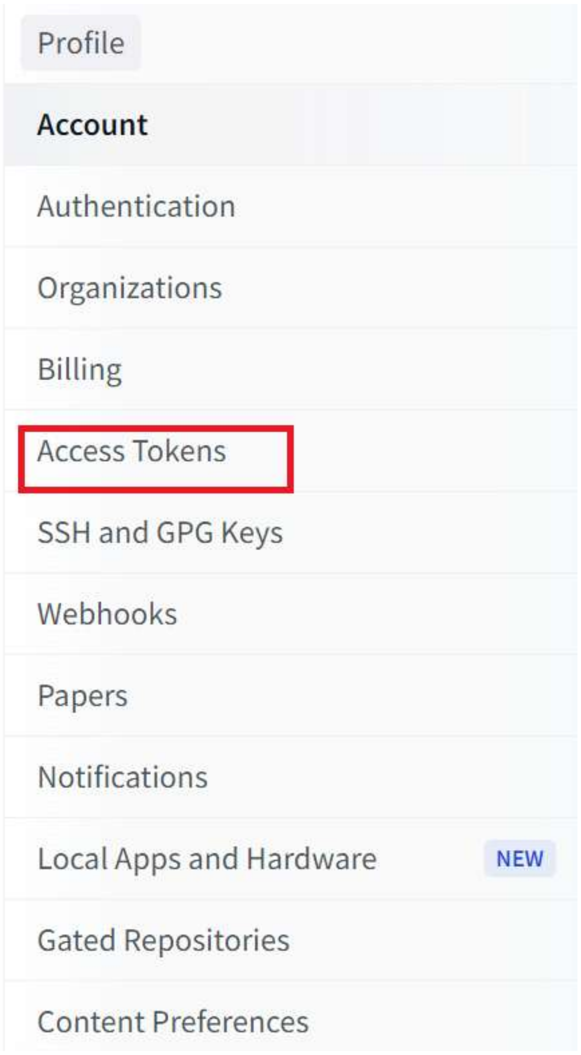
1. Register yourself with the Hugging Face community if you have not already done so: Join Hugging Face.
2. After registration, you will see a dashboard.
3. In the top right corner, select the **Profile icon**, as shown at number 1, in the screenshot. A drop down will open where you need to select **huggingfaceprofile** as shown at number 2 in the screenshot.


https://huggingface.co
Hugging Face
Search models, datasets, users...
Models
Datasets
Sp
Hugging Face is way more fun with friends and colleagues! Join an organization
+ New
RichaArora1998
Profile
Inbox (0)
Settings
Get Pro
Organizations
Create New
Following 0
All
Models
Datasets
Spaces
Papers
Collections
Community
Posts
Upvotes
Likes
NEW Follow your favorite AI creators
Refresh List
artificialguybr · Working on AI demos
kuri-pl · Integrate AI in game dev
katielink · Expert on AI for medical + bio applications
Follow
Follow
Follow

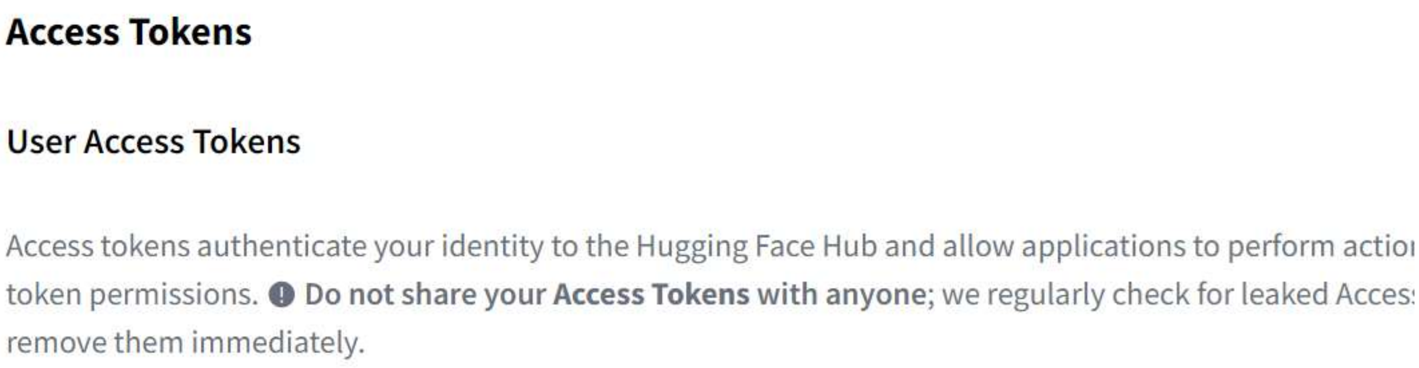
4. Select **Settings** on the left hand navigation bar.



5. Next, on the left hand navigation bar, you need to select **Access Tokens**, as shown in the screenshot.



6. Above step action will lead you to one that section where you can generate new access token. For this, select **Create new token**, as shown in the given screenshot.

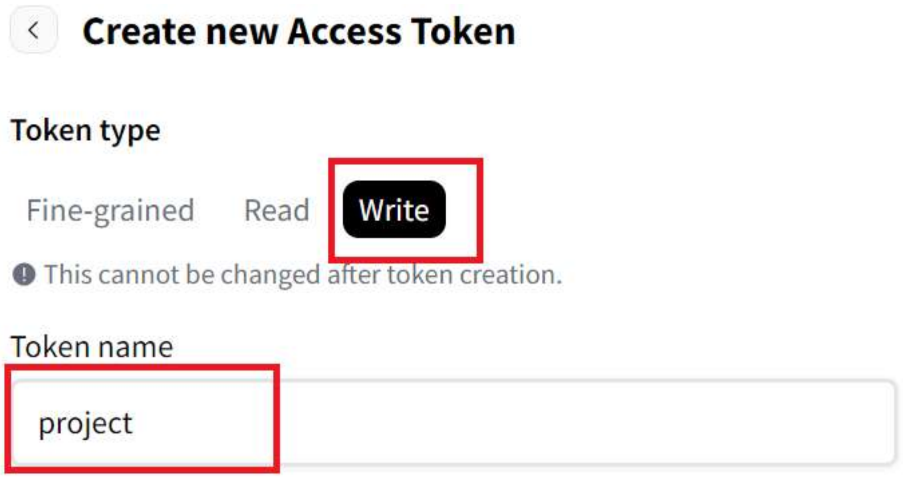


Access Tokens

User Access Tokens

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actio
token permissions. **Do not share your Access Tokens with anyone**; we regularly check for leaked Acces
remove them immediately.

7. After clicking on **Create new token**, you will be directed to a page where you need to select **write** and then enter a name for your token such as *project*.



Create new Access Token

Token type

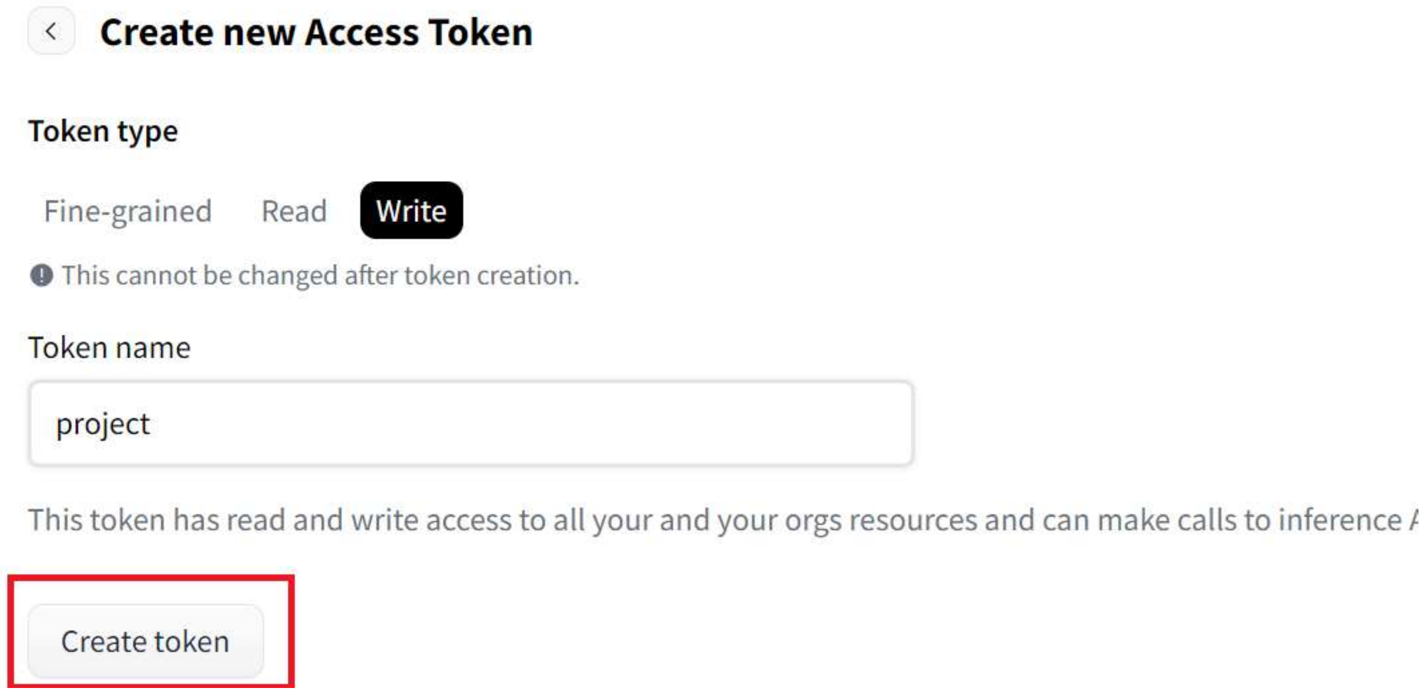
Fine-grained Read Write

This cannot be changed after token creation.

Token name

project

8. Now click on select the **Create token** button. This action will generate a pop up box with your genertaed token.



Create new Access Token

Token type

Fine-grained Read Write

This cannot be changed after token creation.

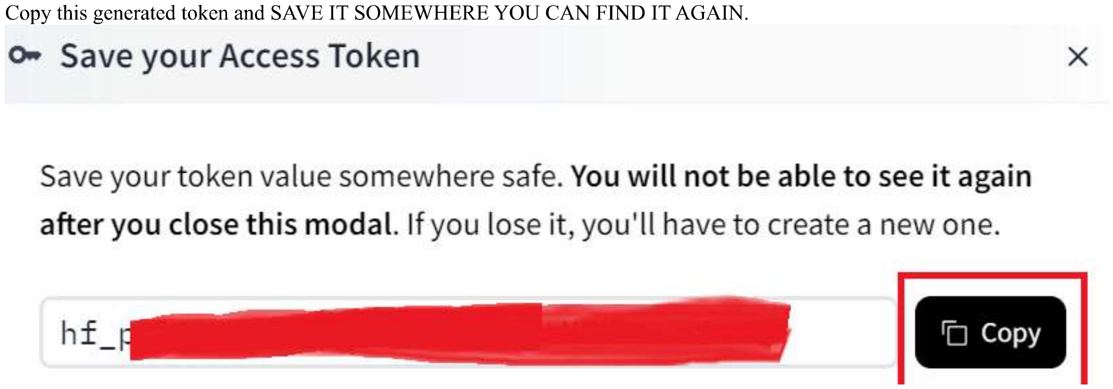
Token name

project

This token has read and write access to all your and your orgs resources and can make calls to inference

Create token

9. Copy this generated token and SAVE IT SOMEWHERE YOU CAN FIND IT AGAIN.



Save your Access Token

Save your token value somewhere safe. **You will not be able to see it again after you close this modal.** If you lose it, you'll have to create a new one.

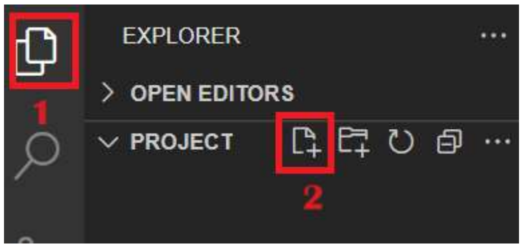
hf_p[redacted]

Note: Make sure that you save the generated token. For security purposes you will not see it again.

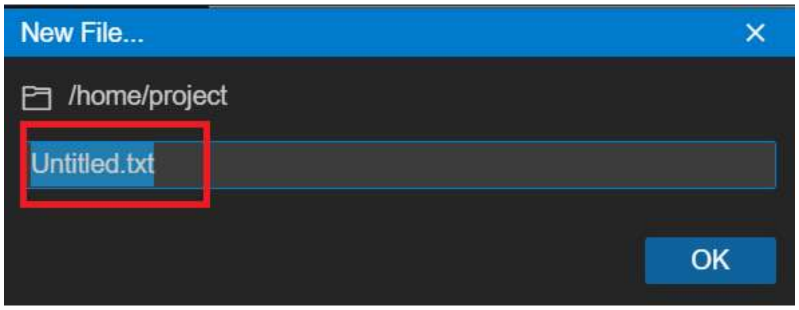
Task 2: Create the JavaScript file

Return to the Skills Network Author IDE environment. In this task, you will create the JavaScript file where you will use your API key so you can use a Hugging Face model.

1. Select **Explorer** on the left side of the terminal window, as shown at number 1 in the screenshot. Then, within the **Project** folder, select **New File** as shown at number 2 in the screenshot.



2. A pop-up box should display with the default file name **Untitled.txt**. Change the default name to `generateEmbeddings.js`.



3. You need to install the Hugging Face client library that provides access to the Hugging face inference models so Node.js can start working with it. For this, execute the command below. We will also use Hugging Face's `pdf-parse` model for this exercise.

```
npm install pdf-parse @huggingface/inference
```

4. In your JavaScript file, you need to import the necessary modules at the top of your file and initialize the Hugging Face inference client with your API key.

```
const { HfInference } = require("@huggingface/inference");
const hf = new HfInference("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");
```

`HfInference` is a class of *Hugging Face* which can provide access to many models for various tasks such as text generation, translation, summarization, and more.

Note: The free version of the Hugging Face API key only allows a limited number of accesses in a given time period, although the documentation doesn't make that limit clear. It does seem like the limit is on an hourly basis, so if you get an error after frequent accesses to their API, you may need to wait up to an hour until you can continue your work.

5. Next, we will convert text to embeddings. To do this, create a variable named `text` in your `generateEmbeddings.js` file and initialize it with any sentence.

```
const text = "Let's use a hugging face AI model";
```

6. Create a function named `getEmbeddings()` and include a `try` and `catch` block in it. In this function, you will call a method that will convert text into vector embeddings. Use the code below for this function.

```
const getEmbeddings = async () => {
  try {
    let embeddings = await convertTextToEmbedding(text);
    console.log(embeddings);
  } catch (err) {
    console.error("Error getting embeddings:", err);
  }
};
```

Let's review how this code works.

- `const getEmbeddings = async () => { ... }` declares an asynchronous arrow function named getEmbeddings.
- The function handles errors with a try-catch block.
- The asynchronous operation `let embeddings = await convertTextToEmbedding(text);` pauses the execution of the `getEmbeddings()` function until the `convertTextToEmbedding()` function resolves.
- The function `convertTextToEmbedding(text)` is presumably an asynchronous function that converts the given text into embeddings. The embeddings variable stores the result.
- The function `console.log(embeddings);` logs the result. If the `convertTextToEmbedding()` function successfully resolves, the resulting embeddings are logged to the console.
- If an error occurs, the `catch (err) { console.error("Error getting embeddings:", err); }` logs a custom error message, "Error getting embeddings:" followed by the error details.

7. Now create one more function named `convertTextToEmbedding()` where you will use the Hugging Face model to convert text into vector embeddings.

```
const convertTextToEmbedding = async (text) => {
  try {
    const result = await hf.featureExtraction({
      model: "sentence-transformers/all-MiniLM-L6-v2",
      inputs: text,
    });
    return result;
  } catch (err) {
    console.error("Error converting text to embeddings:", err);
```

```
      throw err;
    }
  };
```

Let's break down the `convertTextToEmbedding()` function step by step:

- Function Declaration: The function is asynchronous and takes text as an argument.
- API Call:
  - `const result = await hf.featureExtraction(...)` calls the hf.featureExtraction method asynchronously and waits for the result. This prevents blocking the main thread.
  - `model: "sentence-transformers/all-MiniLM-L6-v2"` specifies the model to use for feature extraction. This model is optimized for creating sentence embeddings.
  - `inputs: text` passes the input text to be converted into embeddings.
  - Then returns the result obtained from the featureExtraction method.
- Error Handling: If an error occurs, it logs the error message and rethrows the error.

9. Now call the `getEmbeddings()` function in the last of the program.

```
getEmbeddings();
```

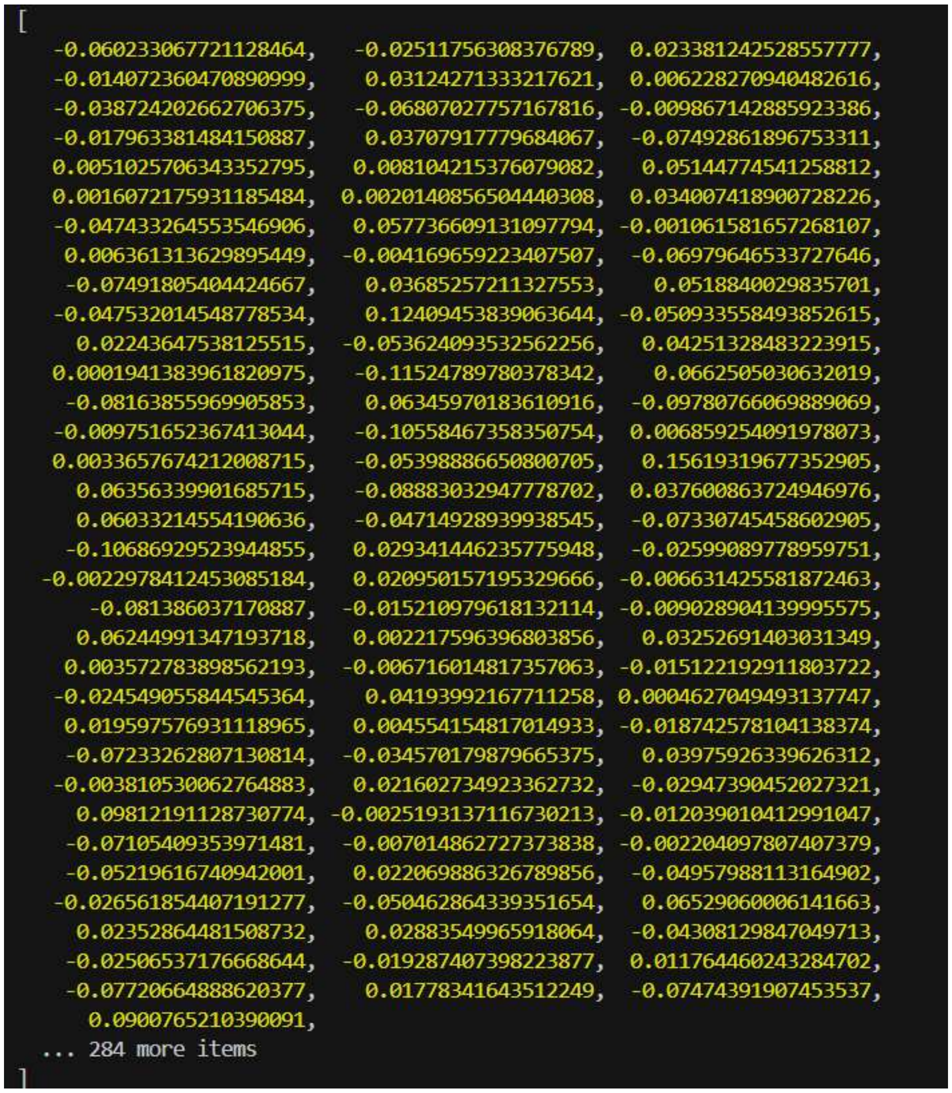
▶ Click here for the check the code flow of generateEmbeddings.js file

10. Now, let's run the program and view the output.

- Open the terminal and check the output by performing given command.

```
node generateEmbeddings
```

- The output will be similar to the following screenshot:



```
[
   -0.060233067721128464,   -0.02511756308376789,  0.023381242528557777,
   -0.014072360470890999,    0.03124271333217621,  0.006228270940482616,
   -0.038724202662706375,   -0.06807027757167816, -0.009867142885923386,
   -0.017963381484150887,    0.03707917779684067,  -0.07492861896753311,
   0.0051025706343352795,   0.008104215376079082,   0.05144774541258812,
   0.0016072175931185484,  0.0020140856504440308,  0.034007418900728226,
   -0.047433264553546906,   0.057736609131097794, -0.001061581657268107,
    0.006361313629895449,  -0.004169659223407507,  -0.06979646533727646,
    -0.07491805404424667,    0.03685257211327553,    0.0518840029835701,
   -0.047532014548778534,    0.12409453839063644, -0.050933558493852615,
     0.02243647538125515,  -0.053624093532562256,   0.04251328483223915,
   0.0001941383961820975,   -0.11524789780378342,    0.0662505030632019,
    -0.08163855969905853,    0.06345970183610916,  -0.09780766069889069,
   -0.009751652367413044,   -0.10558467358350754,  0.006859254091978073,
   0.0033657674212008715,   -0.05398886650800705,   0.15619319677352905,
     0.06356339901685715,   -0.08883032947778702,  0.037600863724946976,
     0.06033214554190636,   -0.04714928939938545,  -0.07330745458602905,
    -0.10686929523944855,   0.029341446235775948,  -0.02599089778959751,
  -0.0022978412453085184,   0.020950157195329666, -0.006631425581872463,
      -0.081386037170887,  -0.015210979618132114, -0.009028904139995575,
     0.06244991347193718,  0.002217596396803856,   0.03252691403031349,
    0.003572783898562193,  -0.006716014817357063, -0.015122192911803722,
   -0.024549055844545364,    0.04193992167711258, 0.0004627049493137747,
    0.019597576931118965,   0.004554154817014933, -0.018742578104138374,
    -0.07233262807130814,  -0.034570179879665375,   0.03975926339626312,
   -0.003810530062764883,   0.021602734923362732,  -0.02947390452027321,
     0.09812191128730774, -0.0025193137116730213, -0.012039010412991047,
    -0.07105409353971481,  -0.007014862727373838, -0.002204097807407379,
    -0.05219616740942001,   0.022069886326789856,  -0.04957988113164902,
   -0.026561854407191277,  -0.050462864339351654,   0.06529060006141663,
     0.02352864481508732,    0.02883549965918064,  -0.04308129847049713,
    -0.02506537176668644,  -0.019287407398223877,  0.011764460243284702,
    -0.07720664888620377,    0.01778341643512249,  -0.07474391907453537,
      0.0900765210390091,
  ... 284 more items
]
```

Conclusion

Congratulations! You successfully generated your own API key and used it to access and utilize a Hugging Face API.

You now know how to:

- Create an API key to use Hugging Face models.
- Import the HfInference class from the @huggingface/inference package and initialized it with your API key to access Hugging Face's Inference API.
- Create the asynchronous functions `convertTextToEmbedding()` and `getEmbeddings()` using the `await` keyword to handle promises for converting text into embeddings.
- Use a try/catch blocks in asynchronous functions to manage and log errors.

Author(s)

Richa Arora
Bethany Hudnutt