**WANDERLUST**

Using fetch, async, and await, you’ll request information from the [Foursquare API](https://developer.foursquare.com/) and [WeatherStack API](https://weatherstack.com/documentation) to create a travel website.

Before you begin, you’ll need to register for developer accounts for both of the APIs above. They’re both free.

For Foursquare there are client id and secret in your **script.js** file, but if you wish you can create your own account. *(Once you make an account,* [*create a new app*](https://foursquare.com/developers/register) *and fill out the form (you can put any link in the “App or Company URL” field). The Foursquare API will then give you a client ID and a client secret.)*

For WeatherStack, follow the instructions for the [Quick Start Guide.](https://weatherstack.com/quickstart) When prompted, sign up. WeatherStack will give you an API Key.

## Add API Information

**1.**

Save the client ID you obtained from the Foursquare API to const clientId. It is already done.

**2.**

Save the client secret you obtained from the Foursquare API to const clientSecret. It is already done.

**3.**

Create a const called url. You can check [the Foursquare documentation](https://developer.foursquare.com/docs/venues/explore) to see the explore venue API endpoint.

Assign url to 'https://api.foursquare.com/v2/venues/explore?near='.

**4.**

Save the API Key you obtained from WeatherStack to const apiKey.

**5.**

Create a const called weatherUrl, save 'http://api.weatherstack.com/current?access\_key=' as the value.

See examples of WeatherStack API on [the WeatherStack documentation](https://weatherstack.com/documentation).

## Get Data from Foursquare

**6.**

Create getVenues() as an asynchronous function that will return a Promise.

**7.**

Inside of that function, add a const called city. Save the value from the user’s input field.

**8.**

Add a const called urlToFetch. This string will contain the combined text of the entire request URL

* the API endpoint URL
* the user’s input city
* a limit parameter with the number of venues you wish to return (use 10)
* the client\_id parameter and your client ID
* the client\_secret parameter and your client secret
* the v (version) parameter and today’s date in this format: YYYYMMDD

Each key-value parameter pair should be joined to the others with &.

For example, to request 5 venues with a client ID of 1234, that portion of the URL would be limit=5&client\_id=1234.

**9.**

Add try/catch statements with empty code blocks.

**10.**

In the catch code block, log the error to the console.

**11.**

In the try code block, use fetch() to send a GET request to urlToFetch. await the response and save it to a variable called response using the const keyword.

**12.**

Create a conditional statement that checks if the ok property of the response object evaluates to a truthy value.

**13.**

Copy and paste the URL into a new tab in your browser. It might be difficult to read, so try using an **extension** such as [JSON View](https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=en) to parse the data.

**14.**

Convert the response object to a JSON object. await the resolution of this method and save it to a variable called jsonResponse using the const keyword.

Log jsonResponse to the [console](https://www.digitalocean.com/community/tutorials/how-to-use-the-javascript-developer-console).

**15.**

Explore the object in the console. There’s a lot of information in there. Let’s save some of that data to a variable called venues. Specifically, follow this nesting chain from the jsonResponse variable to get an array of venue data:

* the response property (an object)
* the groups property (an array)
* the first element of groups
* the items property (an array of venue data)

**16.**

Log venues to the console and see what the API sent back. There should be an array with the number of objects you selected in the limit parameter.

**17.**

Return venues as the very last line of the try code block.

**18.**

Create a click event handler to the button, call the function. Enter a city in the browser and see what is logged! Explore the object!

## Get Data from WeatherStack

**19.**

Create getForecast() as an asynchronous function. Add empty try/catch blocks inside. Log the error inside the catch block.

**20.**

Before the try code block, create a const called urlToFetch that includes:

* the base weatherUrl
* your API key as the apiKey variable
* the &query= parameter (representing the location query) with a value of the user’s input ($input.val())

Don’t forget to join parameter key-value pairs after the API key with &.

**21.**

Inside of the try block, await the response of calling fetch() and passing it the URL you created in a previous step. Save the response to a variable response using the const keyword.

**22.**

Create a conditional statement that checks the ok property of the response object. If this evaluates to a truthy value, await the response of calling .json() on the response object. Save the resolution of this Promise to a variable called jsonResponse using the const keyword.

**23.**

Log jsonResponse to the console.

Create a click event handler to the button, call the function. Enter a city in the browser and see what is logged! Explore the object!

**24.**

Return jsonResponse at the bottom of the try code block.

## Render Data

**25.**

Check the helper functions createVenueHTML() and createWeatherHTML() which is provided in **./helpers.js** and linked from **index.html**.

**26.**

Create a anonymous function inside the click event listener of button. (delete others).

**27.**

Inside this function; add a seperate .then() methods to each getVenues() and getForecast() functions. .then()‘s callback function should take a single parameter and returns createVenueHTML() and createWeatherHTML() functions with these parameters.

1. **28.**
2. Time to hook up the forecast data and the render function. Dont forget to empty, DOM fields before a new search.

## Complete!

**29.**

Congratulations! You should now be able to search for venue and weather details by city and see the response on the page!

## Challenges

**30.**

Include additional information about the weather.

**31.**

Include additional information about each venue from the response.

For a real challenge, try fetching venue photos! This will require an additional request for [venue details](https://developer.foursquare.com/docs/api/venues/details) for each venue, as the photo information is not returned in the initial request.