Async JavaScript and HTTP Requests

Wanderlust

Using fetch, async, and await, you’ll request information from the [Foursquare API](https://developer.foursquare.com/) and [OpenWeather API](https://openweathermap.org/current) 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, 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. You’ll need to save both of those in **main.js**.

For OpenWeather, follow the instructions for the [I. Registration process: How to start](https://openweathermap.org/guide/#how). When prompted, use your first name, and other for the OpenWeather questions for their data collection. OpenWeather will give you an API Key, which you’ll also need to save in **main.js**.

You can view [a live version of this project here](https://content.codecademy.com/courses/intermediate-javascript-requests/wanderlust/wanderlust-v3/index.html).

**Tasks**

40/41 Complete

Mark the tasks as complete by checking them off

**Add API Information**

1.

Save the client ID you obtained from the Foursquare API to const clientId.

2.

Save the client secret you obtained from the Foursquare API to const clientSecret.

3.

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

Add the near parameter without a value. To add a query to the end of a URL, be sure to use ? followed by the first key (near) and an =. You’ll add the value of the near parameter to this URL string when you make the request itself.

const url = 'https://api.foursquare.com/v2/venues/explore?near=';

4.

Save the API Key you obtained from OpenWeather to const openWeatherKey.

5.

Create a const called weatherUrl, save 'https://api.openweathermap.org/data/2.5/weather' as the value.

See examples of OpenWeather API calls under ‘Examples of API calls’ on [the OpenWeather documentation](https://openweathermap.org/current).

const weatherUrl = 'https://api.openweathermap.org/data/2.5/weather';

**Get Data from Foursquare**

6.

The following steps will guide you through constructing the getVenues() function which is called from executeSearch().

Turn getVenues() into an asynchronous function that returns a Promise.

const getVenues = async () => {  
   
};

7.

Inside of that function, add a const called city. Save the value from the user’s input field on the page with $input.val().

const city = $input.val();

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

const urlToFetch = `${url}${city}&limit=10&client\_id=${clientId}&client\_secret=${clientSecret}&v=20180101`;

or:

const urlToFetch = url + city + '&limit=10&client\_id=' + clientId + '&client\_secret=' + clientSecret + '&v=20180401';

9.

Add try/catch statements with empty code blocks.

try {  
   
} catch (error) {  
   
}

10.

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

catch (error) {  
  console.log(error);  
}

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.

const response = await fetch(urlToFetch);

12.

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

For example:

if (response.ok) {  
  // ...  
}

13.

Log the response to the console. In the browser window with the Wanderlust page, enter a city in the search field and submit. Make sure that you have your own browser’s JavaScript console open so that you can see the response that is logged to the console.

14.

Notice that the response includes the URL you requested but doesn’t include the information you asked for. 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. If you don’t want to use an extension, you can also try [JSBeautifier](http://jsbeautifier.org/). Examine the object.

15.

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).

const jsonResponse = await response.json();  
console.log(jsonResponse);

16.

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)

const venues = jsonResponse.response.groups[0].items;

17.

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. You’ll *only* want the venue property inside of these objects. Use .map() to save these venues to the venues array from the previous step.

Add .map(parameter => parameter.valueToStore) to the end of the code from the previous step.

const venues = jsonResponse.response.groups[0].items.map(item => item.venue);

18.

Return venues as the very last line of the try code block. Open the hint to peek at the whole try block.

try {  
  const response = await fetch(urlToFetch);  
  if (response.ok) {  
    const jsonResponse = await response.json();  
    const venues = jsonResponse.response.groups[0].items.map(item => item.venue);  
    console.log(venues);  
    return venues;  
  }  
}

19.

Run the code and see what is logged to the console now!

**Get Data from OpenWeather**

20.

The following steps will guide you through constructing the getForecast() function which is called from executeSearch().

Turn getForecast() into an asynchronous function that returns a Promise. Add empty try/catch blocks inside. Log the error inside the catch block.

const getForecast = async () => {  
  try {  
   
   
  } catch (error) {  
    console.log(error);  
  }  
   
}

21.

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

* the base weatherUrl
* the q parameter (representing the location query) with a value of the user’s input ($input.val())
* and your API key as the APPID parameter

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

const urlToFetch = `${weatherUrl}?&q=${$input.val()}&APPID=${openWeatherKey}`;

or:

const urlToFetch = weatherUrl + '?&q=' + $input.val() + '&APPID=' + openWeatherKey;

22.

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 called response using the const keyword.

const response = await fetch(urlToFetch);

23.

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.

if (response.ok) {  
  const jsonResponse = await response.json();   
}

24.

Log jsonResponse to the console. Enter a city in the browser and see what is logged! Explore the object!

25.

Return jsonResponse at the bottom of the try code block. Open the hint to inspect the complete try block inside getForecast().

try {  
  const response = await fetch(urlToFetch);  
  if (response.ok) {  
    const jsonResponse = await response.json();  
    return jsonResponse;  
  }  
}

**Render Data From Foursquare API**

26.

If you want to follow the steps and render the data with guidance, that’s great! If not, check the hint and update the renderVenues() function provided in **main.js**.

const renderVenues = (venues) => {  
  $venueDivs.forEach(($venue, index) => {  
    const venue = venues[index];  
    const venueIcon = venue.categories[0].icon;  
    const venueImgSrc = `${venueIcon.prefix}bg\_64${venueIcon.suffix}`;  
    let venueContent = createVenueHTML(venue.name, venue.location, venueImgSrc);  
    $venue.append(venueContent);  
  });  
  $destination.append(`<h2>${venues[0].location.city}</h2>`);  
}

This solution uses a helper function createVenueHTML() which is provided in **public/helpers.js** and linked from **index.html**.

27.

In **main.js**. There’s a function called renderVenues that calls the .forEach() method on the $venueDivs array. This is an array of the <div>s in **index.html** where you will render the information returned in the response from the Foursquare API.

Towards the bottom of this method, there is a variable called venueContent. It’s an empty string for now, but you’ll be replacing it with an HTML string to render correct venue information.

Start by creating a const venue to represent the individual venue object inside of the .forEach() callback. Save the current venue at venues[index] to this variable.

$venueDivs.forEach(($venue, index) => {  
  const venue = venues[index];  
  // ...  
});

28.

Create a venueIcon const to save the value of the object representing the venue icon. This is accessible as the icon property of the first element in the array of categories of the venue object.

const venueIcon = venue.categories[0].icon;

29.

Now, construct the full source URL for the venue icon. The venueIcon has a prefix and suffix field used to construct a source path. You can inspect more information about the icon object in the Response Fields and sample Response in the [Foursquare documentation](https://developer.foursquare.com/docs/api/venues/categories) or log venueIcon to the console to inspect it.

Concatenate or combine the prefix property of venueIcon, the string 'bg\_64', and the suffix, and save to a const venueImgSrc. 'bg\_64' is required to fetch icons with a gray background that will show up against the background of the Wanderlust page.

const venueImgSrc = `${venueIcon.prefix}bg\_64${venueIcon.suffix}`;

Or:

const venueImgSrc = venueIcon.prefix + 'bg\_64' + venueIcon.suffix;

30.

Now construct the HTML string to add the new venue. You can use your knowledge of the venue object shape and [this HTML template](https://content.codecademy.com/courses/intermediate-javascript-requests/wanderlust/wanderlust%20templates/venueHTMLTemplate.txt) or follow the steps below. Open the hint for more information about implementing from the HTML template.

Construct a string with the HTML text. Replace the uppercase text (like VENUE\_NAME\_PROPERTY) with the values from the venue object and the constructed icon source URL. Save this string to the venueContent variable, and make sure that it is still passed into $venue.append().

31.

A helper function named createVenueHTML() has been provided to construct the HTML string to display the venue information. You can examine it in **public/helpers.js**, and it is linked in **index.html** so that you can use it in **main.js**.

Pass createVenueHTML() the venue’s name, location, and image source URL and save the returned string to the venueContent variable.

let venueContent = createVenueHTML(venue.name, venue.location, venueImgSrc);  
$venue.append(venueContent);

32.

Now it’s time to hook up your renderVenues() function to the data fetched by getVenues().

In the executeSearch() function towards the bottom of **main.js**, getVenues() and getForecast() are already being called.

Chain a .then() method to getVenues(). .then()‘s callback function should take a single parameter, venues, and return renderVenues(venues).

Save your code, search for a location, and you should be able to see venue information displayed towards the bottom of the Wanderlust page!

const executeSearch = () => {  
  // ...  
  getVenues().then(venues => renderVenues(venues));  
  // ...  
}

**Render Data from OpenWeather**

33.

If you want to follow the steps and render the data with guidance, that’s great! If not, check the hint and copy and paste the function provided in **main.js**.

const renderForecast = (day) => {  
  const weatherContent = createWeatherHTML(day);  
  $weatherDiv.append(weatherContent);  
};

34.

Now construct the HTML string to add the weather forecast in the renderForecast function. When it’s called, the jsonResponse returned from calling getForecast() will be passed in as day. You can inspect the day object’s shape and this [HTML template](https://content.codecademy.com/courses/intermediate-javascript-requests/wanderlust/wanderlust%20templates/forecastHTMLTemplate.txt) or follow the steps below. Open the hint for more information about implementing from the HTML template.

Construct a string with the HTML text. Replace the uppercase text (like DAY\_TEMP\_F) with the values from the day object. Save this string to the weatherContent variable, and make sure that it is still passed into $weatherDiv.append().

The current day for the DAY placeholder may present a challenge. You can construct a new [JavaScript Date object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date) using the dt property of the day object and use the [.getDay() method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getDay) to get an integer representing the current day of the week. Use the provided weekDays array to retrieve a string with the day’s name.

For the DAY\_TEMP\_F you’ll need to convert the temperature provided from OpenWeather to Fahrenheit using the following equation: ((DAY\_TEMP - 273.15) \* 9 / 5 + 32).toFixed(0). You can get DAY\_TEMP from day.main.temp.

35.

A helper function named createWeatherHTML() has been provided to construct the HTML string to display the weather information. You can examine it in **public/helpers.js**, and it is linked in **index.html** so that you can use it in **main.js**.

Pass createWeatherHTML() the day and save the returned string to the weatherContent variable.

let weatherContent = createWeatherHTML(day);  
$weatherDiv.append(weatherContent);

36.

Time to hook up the forecast data and the render function.

Inside executeSearch(), add a .then() method to getForecast(). .then()‘s callback function should take a single parameter, forecast, and return renderForecast(forecast).

const executeSearch = () => {  
  // ...  
  getForecast().then(forecast => renderForecast(forecast));  
  // ...  
}

**Complete!**

37.

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

**Note:** The OpenWeather API endpoint we use can take input of the form: city, state, like Baltimore, Maryland. You can read more about the OpenWeather API [here](https://openweathermap.org/current).

**Challenges**

38.

Try out any of the challenges below to get more practice!

39.

Fetch more than 4 venues and randomize which ones are added to the page.

40.

Include additional information about the weather.

41.

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.