treehouse

Front End Techdegree

Project 8: Use an API to Create an Employee Directory

Table of Contents

Sections of this Guide:

Sections of this Guide:

- **How to approach this project** includes detailed guidance to help you think about how to organize your code, project and files.
- **How to succeed at this project** lists the grading requirements for the project, with hints, links to course videos to refresh your memory and helpful resources.

How to Approach This Project

There are a number of ways to approach and complete this project. Below, one approach is detailed in steps, but you do not have to follow this approach to the letter. You are free to explore your own solution, as long as it meets the requirements laid out in the "How to succeed at this project" section below.

Helpful Hint:

Having a good solid approach to how to begin a project will go a long way in making it easier to create. For instance, one way is to create a "static" version of the site using just HTML and CSS first, before adding in the JavaScript interactivity.

Step 1: Setup

Download and extract the project files. Inside the zip file should be:

- A PNG mockup for the desktop view of the site.
- A PNG mockup for the desktop view of the site with the modal visible.

Set up a new GitHub repository and push the project files to it.

- *Related video:* **Share Your Projects with GitHub**

Note: The following detailed walkthrough is just one way to go about making this project... there are plenty of other methods out there that are just as good! If you have a different way you would like to use, by all means give it a try!

Step 2: HTML

Usually the best place to start building out the code for a site is with the HTML, and this project is no exception. You will first build and style the site with hardcoded "placeholder" employees, before removing them and adding in the employee information with JavaScript. Start by creating your `index.html` file.

- Inside, add the necessary `!DOCTYPE` and `html` tags
- Next, add the `head` tag, along with the usual `meta` and `link` tags .
    - Don't forget to add a "viewport" `meta` tag!

```
<!DOCTYPE html>

<html lang="en">

<head>
  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Awesome Startup Employee Directory</title>

  <!-- style sheets -->
  <link rel="stylesheet" href="css/normalize.css">
```

```
  <link rel="stylesheet" href="css/styles.css">
</head>
```

- Inside the body you will need a header element containing an h1 that says “Awesome Startup Employee Directory”

```
<body>
  <header>
    <h1>Awesome Startup Employee Directory</h1>
  </header>
```

- Next, create a main element inside of the body that will act as a container for the employees. Give it a descriptive class of grid-container
    - From here you will build out the HTML for a single employee that you will use as a guide when creating it in JavaScript
    - The container for an individual employee will be a div with a class of card
        - Add an img tag with a class of avatar. This will be the profile image of each employee
            - Since we are not getting any information from an API yet, it may be helpful to use one of the images from the Web App Dashboard project as a placeholder for now.
        - Next, create a text-container div. This will make it easier to format the text next to the image in CSS
            - Inside, use an h2 for the name of an employee.
            - Create 2 p tags, one for the email and one for the address

```
  <main class="grid-container">
    <div class="card">
      <img class="avatar" src="member-1.jpg" />
      <div class="text-container">
        <h2 class="name">Haleigh Macchiarella</h2>
        <p class="email">dtucker@yakitri.edu</p>
        <p class="address">Chicago</p>
      </div>
    </div>

    <!-- add 11 copies here -->
  </main>
```

- Copy and paste the card structure to make a total of 12 card divs, along with their internal structure.
  Note: Don't worry about changing all of the names and information to match the mockups, these are just being used as placeholders to help us style the page.
- Create the modal markup
  - Also in the body, and after main, create a div that will be the overlay of the modal. Give this class names of overlay and hidden
  - Inside the overlay element, add a div with class modal. This will be the main container of the modal pop up window
    - Add a p tag with a class of modal-close, and text of X
    - Add a div with class of modal-text, this will be where we insert the employee information with JavaScript later
      - For now, create static placeholder information like we did for the employee cards
      - Add an additional hr and 3 more p elements inside the modal text-container div.
        - The additional p tags are for the phone, address, and date of birth

```
<div class="overlay hidden">
  <div class="modal">
    <button class="modal-close">X</button>
    <div class="modal-content">
      <!-- modal information will be added with JS here -->
      <img class="avatar" src="member-1.jpg" />
      <div class="text-container">
        <h2 class="name">Haleigh Macchiarella</h2>
        <p class="email">dtucker@yakitri.edu</p>
        <p class="address">Chicago</p>
        <hr />
        <p>(593) 364-3249</p>
        <p class="address">123 Anywhere St, WV 84814</p>
        <p>Birthday: 01/04/85</p>
      </div>
    </div>
  </div>
</div>
```

Step 3: CSS

- For the most part, the styling of the portfolio is similar to the Responsive Layout project in Unit 2. Flexbox or CSS grid could make the code simpler, but however you choose is fine.
- The modal window needs to use absolute positioning to display correctly, so we will go into detail on how to create that.
    - Since we will be using absolute positioning on the modal overlay, we will need to make the body element the container for the absolute positioning.
        - Set the `position` property of `body` to `relative`

```
body {
  position: relative;
}
```

    - Next, we will want to target the overlay element. This will create the semi transparent background of the modal that covers the whole window.
        - Set the `position` property to `absolute`, this removes the element from the normal document flow.
        - Set the `left`, `right`, `top`, and `bottom` properties all to 0, this will force the overlay to expand across the whole document.
          Note: Do not set any `width` or `height` properties for the overlay, the size is controlled by the properties above.
        - Set the `background` property using rgba so that you can control the transparency.

```
.overlay {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: rgba(100, 100, 100, 0.4)
}
```

    - Modal container
        - Set a `width` for the modal
        - Center it horizontally by setting the left and right margins to `auto`
        - Align the text to center
        - Add padding to create separation between the text and border

- Add personal styling preferences
- **Thinking ahead:** You will use absolute positioning for the close button in order to put it in the top right of the modal. To aid in this, set the modal position to relative so that the modal positioning is based off of the modal.

```
.modal {
  width: 375px;
  margin: 15% auto auto;
  position: relative;
  text-align: center;
  background: #fff;
  border: 1px solid #778;
  border-radius: 5px;
  padding: 30px 20px;
}
```

- Modal close button
  - Set the position property to absolute
  - Adjust the top and right properties until the position matches the mockups
  - Set the cursor property to pointer so that users know they can click the X

```
.modal-close {
  position: absolute;
  right: 15px;
  top: 5px;
  cursor: pointer;
}
```

- Modal content
  - The styles for the modal content are controlled by the same classes as the employee cards.
- Hidden class
  - The hidden class is just a helper class to hide the modal when the main portfolio is displayed
  - Set the display property to none, and call it done!

```
.hidden {
  display: none;
}
```

Step 4: JS

- Declare global variables
    - employees -- empty array that will hold values from the API
    - urlAPI -- string literal that stores the url of the API, complete with desired options.
    - gridContainer -- stores the DOM element that is the container for the employees
    - overlay -- stores the DOM element that acts as an overlay for the modal.
    - modalContainer -- stores the DOM element that is a container for the modal information.
    - modalClose -- stores the DOM element that is the modal's close button.

```
// global variables
let employees = [];
const urlAPI = `https://randomuser.me/api/?results=12&inc=name, picture,
email, location, phone, dob &noinfo &nat=US`
const gridContainer = document.querySelector(".grid-container");
const overlay = document.querySelector(".overlay");
const modalContainer = document.querySelector(".modal-content");
const modalClose = document.querySelector(".modal-close");
```

- Use `fetch` to retrieve information from the API
    - Pass the url information to fetch
    - Format the response as JSON.
    - `console.log` the response so that the data can be sifted using Chrome Dev Tools
        - **Note:** Whenever you are working with data from an API for the first time, it's helpful to log the results to the console so you can see how the data is formatted.
    - Return the results of the response
    - Pass control to the `displayEmployees` function, which will be created next
    - Catch any errors and display them in the console
    - *External Link:* **Random User: How-To**
    - *Related video:* **Working with the Fetch API**
    - *Related video:* **Asynchronous Programming With JavaScript**
    - *Related video:* **Debugging JavaScript in the Browser**

```
// fetch data from API
```

```
fetch(urlAPI)
  .then(res => res.json())
  .then(res => res.results)
  .then(displayEmployees)
  .catch(err => console.log(err))
```

- Create displayEmployees function that has a single parameter named employeeData
    - Set the employees variable equal to employeeData so that it can be accessed outside of this function
    - Create an empty string variable named employeeHTML. This will contain the HTML markup for the employee elements.
    - Loop through each employee in employees.
        - Create function scoped variables to store employee:
            - Name
            - Email
            - City
            - Picture
        - Using a template literal, add each employee markup to the employeeHTML variable. Use the static markup your created in index.html as a guide
    - Add the content of employeeHTML to the inner HTML of gridContainer
    - *Related video:* **Practice Template Literals**
    - *Related video:* **Practice forEach in JavaScript**

```
function displayEmployees(employeeData) {

  employees = employeeData;

  // store the employee HTML as we create it
  let employeeHTML = '';

  // loop through each employee and create HTML markup
  employees.forEach((employee, index) => {
    let name = employee.name;
    let email = employee.email;
    let city = employee.location.city;
    let picture = employee.picture;

    // template literals make this so much cleaner!
```

```
    employeeHTML += `
      <div class="card" data-index="${index}">
        <img class="avatar" src="${picture.large}" />
        <div class="text-container">
          <h2 class="name">${name.first} ${name.last}</h2>
          <p class="email">${email}</p>
          <p class="address">${city}</p>
        </div>
      </div>
    `
  });

  gridContainer.innerHTML = employeeHTML;
}
```

- Create a displayModal function that has a single parameter named index
    - Create function scoped variables for the information that needs to be displayed in the modal. Use the information from the employees array, at the index passed to this function.
      Note: for a cleaner, more advanced way to do this, use Object destructuring.
    - Declare a variable that creates a new Date Object, based on the employee's date of birth.
    - Create a variable named modalHTML that stores a template literal of the modal markup.
        - Use the index.html file as a guide.
    - Remove the hidden class from the overlay
    - Set the inner HTML of modalContainer equal to the content of modalHTML
    - *Related video:* [Introducing ECMA2015: Destructuring]

```
function displayModal(index) {

  // use object destructuring make our template literal cleaner
  let { name, dob, phone, email, location: { city, street, state, postcode
}, picture } = employees[index];

  let date = new Date(dob.date);

  const modalHTML = `
    <img class="avatar" src="${picture.large}" />
```

```
    <div class="text-container">
      <h2 class="name">${name.first} ${name.last}</h2>
      <p class="email">${email}</p>
      <p class="address">${city}</p>
      <hr />
      <p>${phone}</p>
      <p class="address">${street}, ${state} ${postcode}</p>
      <p>Birthday:
${date.getMonth()}/${date.getDate()}/${date.getFullYear()}</p>
    </div>
  `;

  overlay.classList.remove("hidden");
  modalContainer.innerHTML = modalHTML;
}
```

- Event Listeners
    - gridContainer click event
        - Check if the gridContainer itself was clicked, or a child element.
        - Use the closest() function to find the correct card that was clicked
        - Get the data-index attribute from the card to pass to the displayModal function
        - Call displayModal, and pass in the card index.

```
gridContainer.addEventListener('click', e => {

  // make sure the click is not on the gridContainer itself
  if (e.target !== gridContainer) {

    // select the card element based on its proximity to actual element
clicked
    const card = e.target.closest(".card");
    const index = card.getAttribute('data-index');

    displayModal(index);
  }
});
```

    - modalClose click event

    - Add the `hidden` class to the modal overlay

```
modalClose.addEventListener('click', () => {
  overlay.classList.add("hidden");
});
```

- Remove the employee and modal static markup from `index.html`

How to succeed at this project

Here are the things you need to do pass this project. Make sure you complete them ***before*** you turn in your project.

API Usage

- ❑ Use fetch to request 12 random users from the API
- ❑ New random employee information displays each time the page refreshes

User Directory

- ❑ The directory includes the following:
    - ❑ Employee Image
    - ❑ First and Last Name
    - ❑ Email
    - ❑ City

Modal Window

- ❑ Modal window displays the following details:
    - ❑ Employee image
    - ❑ Name
    - ❑ Email
    - ❑ Phone Number
    - ❑ Detailed Address, including street name and number, city, state and postcode
- ❑ Birthdate
- ❑ There is a way to close the modal window

Matching the Mockups

- ❑ Directory has been styled so that all the major elements are in place and roughly matches the mockups