Welcome to the Final Practical

First, you're going to piece together a working web app using modules that display's results from an API for 50% of your grade. We'll give you much of this, however, you'll need to put the parts together, add a few lines of code and get it working.

Then once you have the basics working, you'll perform as many of the following feats as possible for 10% to 15% each:

- Convert your getData function to use Fetch instead of an xhr object.
- Convert your CSS to use Bulma Cards for layout instead of standard CSS.
- Optionally*, convert your vanilla javaScript to use TypeScript with strict types.
- Convert your app to ES5 JavaScript whether you got TypeScript working or not so that it works on the widest possible number of browsers.
- Upload your Final to Banjo and provide a link in the Final Practical Assignment Dropbox.

Getting Started:

1. Download the Starter Files.
    - **They are attached to the Final Practical assignment dropbox for your section.**
2. Examine your files:
    - Each file is going to need you to add a little bit to it to make the basic app work:
3. First: **final.html**
    - Have it use the styles.css (for now) in the css folder.
    - Change the name Ace Coder in the footer to be your name.
    - *You'll be asked to add a class in the **ui.js** section below.*
    - *You'll be asked to add a link to the **main.js** module in the appropriate section below.*
4. Second: **api.js**
    - Examine the familiar XMLHttpRequest method for downloading API data… Be sure that you're clear on how the callback methods are utilized.
    - Fill in the line of code that will parse the JSON data returned as the xhr.responseText into a JS Array and store it in the variable that gets passed to the callback function.

5. Third: **ui.js**
    - Examine the 3 exported functions. The comments tell you what will should be passed to each.
    - Fill in the line of code here that adds the card element (a div first created in line 8) to the results container (selected in line 4).
    - Examine the showError() and clearError() functions and make a change to the final.html code to initially hide the #errorMessage section (it just involves adding a class).
6. Fourth: **main.js**
    - Add two lines to the top of final.html to load the above modules and import all of their functions.
    - Add a link to this file in final.html so that it loads it as a JS module.

Following the above steps, your app should function. That much is worth about 50%

For the final 50%+ here are 5 feats to accomplish:

1. **Convert your getData function to use Fetch instead of an xhr object.**
    - The use of async and await is welcome, but not required.
    - You should be sure to catch errors. (one of the api "types" could send ill-formatted data.)
    - Some class notes you could reference:
        - HW-ajax-5.md (https://github.com/rit-igm-web/igme-330-shared/blob/main/notes/HW-ajax-4.md) through HW-ajax-7.md (https://github.com/rit-igm-web/igme-330-shared/blob/main/notes/HW-ajax-7.md)
        - These Draft notes about Promises/Fetch/async-await, etc: https://github.com/dccircuit/IGME-330-Spring-2024/blob/main/notes/promises-fetch-more.md
        - In case you wanted to look back to the example code in the online quiz part that you just took, this is what the question asked you about:

```
// (assume the API returned { "status":"success" })
fetch("https://api.example.com/status")
      .then(response => response.json())
      .then(data => console.log(data.status))
      .catch(error => console.error(error));
```

2. **Convert your CSS to use Bulma Cards for layout instead of standard CSS.**
    - Bulma Card documentation: https://bulma.io/documentation/components/card/
    - We recommend the following skeleton layout to get responsive columns with cards inside:

    Showing below in this format: *tag (bulma classes)*

- section (columns is-multiline) – apply to section id="results" in html
    - div (column is-one-quarter) – not already there, add in html or js
        - div (card)
            - div (card-image)
                - figure (image is-square) – image element goes in here.
            - div (card-content)
                - div (media)
                    - div (media-content)
                        - p (title is-4) – animal name goes in here.
                        - p (subtitle is-6) – breed goes in here.
                    - div (content) – location goes in here.

SAVE GAME? (Y/N)

RIGHT HERE,
Before going on and trying to do anything with TypeScript,
SAVE YOUR PROGRESS
(that is, make a zip, or make a duplicate copy of your work folder, or whatever)
SO THAT YOU CAN RESTORE BACK TO THIS POINT
if you lose a life on this optional* TypeScript 'Boss Level'

*If you choose not to try the TypeScript portion, you can still get all points for the Practical by doing everything else correctly. However, if you *also* convert things to TypeScript, you can receive bonus points on your practical (how much is up to your instructor).

3. **Optionally (see above), convert your app to use TypeScript with strict types:**

   Add the following files to configure TypeScript to the root of your project document:

   - Note: all three of these files are also available in the extra text file included in the myCourses Practical Assignment Dropbox called config_files.txt – if you'd prefer to copy/paste from there.

package.json

```
{
  "name": "final-practical",
  "version": "1.0.0",
  "main": "index.js",
 "scripts": {
  "build": "webpack --mode production --watch"
},
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "ts-loader": "^9.5.1",
    "typescript": "^5.7.2",
    "webpack": "^5.97.1",
    "webpack-cli": "^5.1.4",
    "webpack-dev-server": "^5.1.0"
  }
}
```

webpack.config.js:

```
const path = require('path');
module.exports = {
  entry: './src/main.ts',
  module: {
    rules: [
      {
        test: [/\.ts?$/,/\.js?$/],
        use: 'ts-loader',
        exclude: /node_modules/,
      },
    ],
  },
```

```
  resolve: {
    extensions: ['.tsx', '.ts', '.js'],
  },
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  devtool: 'source-map'
};
```

tsconfig.json:

```
{
    "compilerOptions": {
        "target": "ES5",
        "module": "ES6",
        "sourceMap": true
    },
    "include": ["src/**/*"]
}
```

Notice that the above files all reference an 'src' folder, not your 'js' folder... See the "first steps" below to create that new folder.

After adding the above files to your root application folder, you should be able to use

```
npm i
```

to install all of the files in package.json and then use the command:

```
npm run build
```

to try to convert your TypeScript to ES5.
(spoiler alert: this won't work right yet... your job is to make it work).

Here are some notes you've seen before about converting apps to TypeScript:

- https://github.com/rit-igm-web/igme-330-shared/blob/main/hw/hw3-typescript-notes.md

  (which, in turn, references these steps:)

- https://github.com/rit-igm-web/igme-330-shared/blob/main/notes/intro-typescript.md#iii-use-node--webpack-to-transpile-a-typescript-app-to-js

Some ~~suggested~~ important first steps:

- make a new **src** folder for your TypeScript. (don't just edit your existing .js files… if you can't get TypeScript working, you can fallback to these)
- copy and rename all .js files in the js folder to .ts files in the **src** folder.
- Start fixing errors first:
    - The module import paths still say .js (you should remove extensions entirely)
    - You need to use type assertion to tell the creatureTypeSelector that it will be an "HTMLSelectElement"
- We haven't done it before (unless you figured it out for your HW3), but the api.ts file has a function that expects callback functions as parameters. So Add these lines at the top of api.ts:

  ```
  // Define the type for the callback parameters
  type SuccessCallback = (param: object[]) => void; // Callback for successful data retrieval
  type ErrorCallback = (param: string) => void;  // Callback for error handling
  ```

  As you can see, these lines define that the callback function must follow a specific **signature**, meaning it should accept a parameter of a defined type and must not return anything.

- Use these new types (defined above) to add explicit types to the two callback parameters to getData
- There is an object that contains an adoptable creature, you should set up an interface using

  ```
  interface Creature {
     // provide types for picture, name, breed,
     // and location (all strings)
  }
  ```

- You can either define this interface at the top of each module that uses it (both **main.ts** and **ui.ts**) or you could put this interface in a module of its own and import it in both files. Your choice.
- You can then strictly type `data` as an array of Creatures in both files.
- Continue through the rest of the code looking for anything else that needs type assertions and/or should be explicitly typed.

4. **Convert your app to ES5 JavaScript whether you got TypeScript working or not so that it works on the widest possible number of browsers.**
    - The above steps should have resulted in this when you had a successful build.
        - If you didn't get your TypeScript working, you should be able to use these instructions to just try to convert your ES6 code to ES5:
            - https://github.com/rit-igm-web/igme-330-shared/blob/main/notes/bundling-transpiling.md#iv-bundling-an-es6-project-with-webpack
            - *Important note:* These instructions assume that javascript files are located in an **src** folder. You'll need to make sure that the configuration path and the actual file path matches.
    - Either way, you'll end up with a bundle.js file that needs to be linked to in a script tag in your html file.
    - Do this, formatting (and positioning) the script tag properly

5. **Upload your Final to Banjo and provide a link in the Final Practical Assignment Dropbox.**
    - Only upload the **necessary files** into your banjo account. Put them anywhere you want, but the link should be directly to your final.html.

After you have completed as many of the 'feats' as you are able:

1. Rename your directory from **final-practical-start** to **lastname-firstinitial-final-practical**
2. Remove any **node_modules** folder (if you made one)
3. *zip* up that directory. (it should now be called **lastname-firstinitial-final-practical.zip**)
4. attach the zip file to the practical dropbox (like any other assignment)
5. if you were requested to get your file uploaded to and working on banjo, include the direct URL to the **final.html** file. If the URL is not included in the dropbox, we won't go looking for it. This is the only way to submit it (unless your instructor gave you a different method).