By Name

Shiny Pokémon Tracker

The table of contents

[About the app 3](#_Toc206961719)

[Index 3](#_Toc206961720)

[Navigation 3](#_Toc206961721)

[Sections 4](#_Toc206961722)

[Current hunt 5](#_Toc206961723)

[New hunt 6](#_Toc206961724)

[Change current hunt 6](#_Toc206961725)

[Target and shiny lists 7](#_Toc206961726)

[Showcase 7](#_Toc206961727)

[Info 7](#_Toc206961728)

[Styling the app 7](#_Toc206961729)

[JavaScript 8](#_Toc206961730)

[My code 8](#_Toc206961731)

[The borrowed code 13](#_Toc206961732)

Shiny Pokémon Tracker

# About the app

The purpose of this app is to help with shiny hunting on different Pokémon games. The user sets the information of their target pokémon, and the hunt can start. Keep track of how many pokémon you’ve encountered before finding and capturing a shiny!

The app is a single page application. It uses Tailwindcss framework for styling the HTML and JavaScript to create user interactions and enables saving in web browser's memory. Prototyping was done in Figma and my code editor was Visual Studio Code. I chose a blue, gray and white theme for the app. It was inspired by a pokémon named Poliwag.

# Index

The index page contains the title of the app (Shiny Pokémon Tracker), a button and a picture of a Poliwag. The font on this page is from Google Fonts and it’s called Karantina. The title has a drop shadow to mimic a stroke on the font. The button and the picture have drop shadows too. The button changes its color when the user hovers their mouse on top of it. The button redirects to the actual app. The index page has a gradient background. It changes from white to blue.

# Navigation

The app’s navigation is executed with big buttons. There is no actual nav bar. The buttons take up most of the screen space on small screens. They’ve been arranged with a flexbox. The buttons are links that open the app's different sections. The buttons’ corners are rounded, and they have drop shadows to make them look less dull. The color of the buttons changes on hover.

On larger screens the layout becomes different. The arrangement changes to a grid, and the buttons’ shape becomes more rectangular. The container is placed horizontally at the center, and it gets a white border.

I chose to have lowercase text on the navigation buttons. I think it looks more stylish than capitalizing the first letters. That theme continues on other sections. The color of the font is white. It looks nice on blue background. The used font is Karantina. The background of navigation is the same as on the index page.

Navigation still contains the debug buttons I used while making the app. I decided to keep but hide them since I want to continue developing the tracker after the project’s deadline has come and gone.

# Sections

All the sections have the same base layout. A header is at the top of the page. Its background color is blue. The header contains the name of the section and a button to close it. In reality, there are two buttons. One of them is hidden according to screen/window size. The font color of the header is white, and section name is lowercased.

The header is supposed to stick on top of the section but is not doing that at the moment. The feature worked in earlier versions, but I changed how the layout works. This bug will be fixed later.

The section’s title is under the header. Both the title and the header use Karantina as their font. The font of the title is bolded, and its color is dark gray. The text after the title is using a sans serf typeface and the same font color as the title. Sections’ background color is light gray.

On bigger screens the sections are draggable. When the user opens a section, it can be moved around the screen. This feature causes some known bugs that are listed in the app’s info section.

Currently, the user can open only one section at a time. I’ve planned to change this so the user can have multiple sections open at the same time.

# Current hunt

The current hunt section forms from a picture, the name of the species, hunt method, starting date and encounter count of the hunt. The specie’s name is using Karantina as its font and it has a blue drop shadow. The color of the font is white.

Under the previous items there are buttons which decrease or increase the encounter count. Between the buttons is the actual number that displays encounter count.

There are three more buttons after the last ones. The first one shows notes about the hunt. After pressing the show notes buttons, the notes become visible, and the show button disappears. It gets replaced by hide notes button that does exactly what it says.

Belove the notes are PAUSE and CAPTURED buttons. The first one just returns the user to navigation, just like the buttons in the header. The second one moves the user to navigation and moves the current hunt to shinies list, removing the hunt from targets list.

The font for all of the buttons is Karantina. The buttons are blue, their corners are rounded, and the font color is white. I decided to have show and hide notes in lowercase. It matched the navigation buttons. PAUSE and CAPTURED buttons are in all caps because they’re “function” buttons; they run a more important task. In the future, there will be an EDIT button which allows the user to edit their hunt information.

Current hunt section is not fully functioning at the moment. It doesn’t display the right picture of the hunt. Instead, it shows a picture of a shiny Bellossom. I haven’t implemented adding a picture for the hunts yet, but I plan to do so in the future.

Also, when there is no active hunt (current hunt), the button of this section should be hidden in the navigation. This feature will be added later. For now, if there is no active hunt, all the fields of the section are empty (except for the picture which is hard coded to be there).

# New hunt

The new hunt selection has a form for starting a new hunt. The form consists of four fields and a button. The fields’ background color is white, and the corners are rounded. The font color is light gray. The buttons are like the buttons in the current hunt section.

The first field is a regular text field. The user types in their chosen pokémon species. In the future I want this field to be a selection menu of all the currently known Pokémon.

The second field is a selection menu with 3 options. I’m planning to add more options later. The third field is for the start date of the hunt, and the fourth one is for possible notes for the hunt. The last field is bigger than the other ones.

I want to add more fields later, for example in which Pokémon game the user is doing their hunt.

# Change current hunt

The change hunt section has a list of user’s target pokémon. From this section you can change the current hunt.

The list consists of buttons which are created with JS from the targets list. The buttons have a gray background color and white text. When the user hovers over a button, its color changes to blue.

By pressing a button the user is redirected to current hunt section. Simultaneously, current hunt is set to the pokémon that the user chose from the list.

# Target and shiny lists

Both the target and the shiny lists function the same. They have the same kind of list as in ‘change hunt’ section.

# Showcase

The buttons in target and shiny lists redirect the user to showcase section. The section in question doesn’t have a title like all the other sections have. Instead, it has the name of the pokémon species the user chose from either one of the lists. The species font is styled similarly as the font in current hunt section.

The other information is shown under the species name as white text fields that are actually paragraphs. The information is the same as the user would see in the current hunt section. The fields have rounded corners, and the font is dark gray.

# Info

The info section has information about the app, future features and known bugs.

# Styling the app

I used Tailwindcss framework instead of vanilla css to style my application. Using Tailwind speeds up development time because you can style your website or application without leaving your html files. I installed Tailwind with their CLI tool. You can read more about Tailwind on their website.

When using Tailwind’s style classes I tried to avoid repetition. I checked every HTML element one by one to make sure there isn’t any. But I’m only a human so there probably is some. At least I moved font classes from inner divs to sections. Now that I think about it, I probably could have placed sections’ font style classes to the main “container” under the body tag.

# JavaScript

I used JavaScript to make the app’s interactivity. JS is mostly known as Web pages’ scripting language but nowadays it is also used outside of them. JS has frameworks and libraries to help developers with their frontend development. I didn’t use any in my project but briefly thought about using Vue.js.

# My code

Here I talk about the functions and variables I used in my app.

**LocalStorage**

LocalStorage is a read-only property and is used to store data to browser’s memory. The data is stored as a JSON string. Later I refer localStorage as ls.

**Target class**

The Target class has a constructor and one method. The constructor initializes the class’s variables. The setID method checks if there is anything saved in localStorage under the key currentID. If there is no data the method returns the id, which is set to 0.

If data was found, the method changes the data to a numeric value and adds one to it. Then the method saves the id to ls under the key currentID and returns the value.

The data that is under currentID should always be an integer (after using JSON.parse) since the first value that was saved there should be 0. I wanted to keep track of how many new targets the user creates and the id makes them all unique and prevents a scenario where the user creates multiple hunts of the same target pokémon and the application would experience a bug.

I was supposed to make another class named Shinies. I decided not to do so for a simpler app. Later I will rename the Target class to Pokemon since I really don’t need two classes. When moving the objects from one array to another I can just add more variables to the object (like date of capture).

**Function getSavedData**

The getSavedData function returns a value from ls if there is any on the given key.

**Variables targets, shinies and currentHunt**

The targets variable is an array of Target objects. The shinies variable is an array of Target objects after they’ve been removed (captured, not deleted) from the targets array. The currentHunt variable is an object of the user’s current hunt. The values of these variables are set by the getSavedData function.

**Function toggleSelections**

This function toggles the visibility of different sections and their background color on smaller screens.

**Function toggleBig**

The toggleBig function is very similar to the toggleSelections function. It toggles the main tag’s and sections’ visibility. At first there was supposed to be only the toggleSelections function, hence the function’s if statement. For some unknown reason I made two similar functions instead.

**Function createNewTarget**

This function creates a new Target object and sets its values. The values come from the new hunt section’s form and Target object’s method. After setting the values the object is pushed to targets array. The array and the object are then saved to ls. The function returns a new Target object.

**Function displayCurrentHunt**

The displayCurrentHunt function displays information about a created Target object. I added the default values to the function, so I didn’t have to change my code in other places.

**Function createCurrentHunt**

This function calls the createNewTarget function and then the displayCurrentHunt function. It then hides the new hunt section and resets its form. After that the function checks user’s window size and calls a section toggle function accordingly.

**Function checkCurrentHunt**

The checkCurrentHunt function checks if the currentHunt object isn’t null. If this is true it calls the displayCurrentHunt function. This function is called when the app is loaded.

**Function showTargetNotes**

This function toggles the current hunt section’s notes field.

**Function changeCount**

The function changeCount changes the encounter count of the current hunt in the current hunt section. On every execution it saves the current hunt with its new value to ls. The object is also being saved to targets array and then the array is saved to ls. This is to prevent any data loss in case the user accidentally closes the app or presses refresh.

Finding the right object from the array is being done by using the array.findIndex method. I must use a function inside of the parentheses. I must use findIndex instead of indexOf because I’ve stored the objects to ls which changes them to different objects than they used to be.

**Function stopCurrentHunt**

This function stops the user’s current hunt by pushing the currentHunt object to the shinies array and splicing it from the targets array. The use of array.findIndex is explained in the previous function’s description. After editing the arrays, they are saved to ls. CurrentHunt is then set to null and is removed from ls. Then I’ve hardcoded what could have been done with displayCurrentHunt with few edits. I believe I did this because I didn’t want to break any previous code. I think toggle selection functions could have been used here but my excuse is I ran out of time and the fear of bugs.

**Function changeHunt**

The changeHunt function changes the currentHunt object to be what user selected from the targets array. Then it calls the displayCurrentHunt function and toggles the visibility of current hunt and change hunt sections.

**Function showPokemon**

This function opens the ‘showcase’ section which then displays the selected pokémon’s information.

**Function showList**

The showList function creates list items that are then displayed in a chosen list. The function has a for each loop that creates a button, sets its text value and class values.

It then checks if the function is being used in the change hunt section or in the “x list” sections (x being targets or shinies). After that it adds an event listener to the button, either for the changeHunt or the showPokemon function. After this the function creates a new list item and makes the button its child and then the item to be the HTML list’s child.

If the used array was empty, display a message informing about that instead. Lastly the function checks the window size and calls a toggle selection function accordingly.

# The borrowed code

To enable dragging the sections on bigger screens, I had to find a way to do so. I used Google and one of the search results was a page from W3Schools. I opened the site and copied the code. VS Code crossed out some of the code and it said that some of it wasn’t up to current standards. The code was still working. I’m using Firefox for testing.

I had no idea how to fix the code, so I just used Copilot. It changed the code to be up to date. I tested it and it still worked so I decided to use it in its current form.

The code consists of a function that has functions inside of it and another function. The first functions sets position values to 0. The closeDragElement removes event listeners when the left mouse button is released.

The second function, named elementDrag, makes sure that an event is a mouse event. The new position is calculated from the mouse pointer’s current position. Then the element’s new position is set with its css left and top properties.

The third function is called dragMouseDown. It makes sure that the user can drag the element only with the left mouse button. Then it gets the mouse pointer’s position. Lastly it adds two event listeners, one for mouse button up and one for down.

After the functions there is an if statement. It attaches the dragMouseDown function to an element. In the original tutorial it said to make a div element with a header (like in my app). But for some reason I’ve made the main tag the header instead of the section headers. I will change this later and it should fix some of the current bugs. After the change I remove the else statement since I don’t want to drag a section from anywhere else other than its header.

After the big function, the code makes a media query list object that checks if the browser window is at least 72rem wide.

Then there’s a function called enableDragIfWide. It checks if the element the user is trying to drag is the set element. Then the function is called. After that the code adds an event listener to the media query list object so the dragging function can be used.