Shiny Pokémon Tracker

By Susanna Kaunio

The table of contents

[**About the app** 3](#_Toc207098363)

[**Index** 3](#_Toc207098364)

[**Navigation** 4](#_Toc207098365)

[**Sections** 4](#_Toc207098366)

[**Current hunt** 5](#_Toc207098367)

[**New hunt** 6](#_Toc207098368)

[**Change current hunt** 6](#_Toc207098369)

[**Target and shiny lists** 7](#_Toc207098370)

[**Showcase** 7](#_Toc207098371)

[**Info** 7](#_Toc207098372)

[**The footer** 7](#_Toc207098373)

[**Styling the app** 8](#_Toc207098374)

[**JavaScript** 8](#_Toc207098375)

[**My code** 8](#_Toc207098376)

[**The borrowed code** 12](#_Toc207098377)

**Shiny Pokémon Tracker**

# **About the app**

The purpose of this app is to aid users with shiny hunting across various Pokémon games. The user enters the information of their target pokémon to begin a hunt. They can keep track of various data regarding shiny hunting, not limited to hunting method used, date started, number of encounters before capturing a shiny and a section for the user to input their own notes.

The app is a single page application, it uses Tailwindcss framework for styling the HTML & JavaScript to enhance user interaction and enable saving in the web browser’s memory. The prototyping was done in Figma and the code editor used was Visual Studio Code. I decided upon a color scheme of blue, gray and white. This was inspired by the pokémon and mascot of the app, Poliwag.

# **Index**

The index page contains the title of the app (Shiny Pokémon Tracker), a button and an image of a Poliwag. The font used is Karantina, from Google Fonts. The title has a drop shadow to mimic a stroke on the font. The button and picture also have drop shadows. The button changes color when the user hovers it and upon pressing, it redirects to the app. The index page background has a gradient, changing from white to blue.

# **Navigation**

Navigation is executed via large buttons. There is no traditional nav bar. The buttons fill most of the space on small screens. They’ve been arranged with a flexbox. The buttons are links that navigate to the app’s various sections. By design, the corners of the buttons are rounded, they also have a drop shadow to pop out more to the user and make them look less dull. The color of the buttons changes on hover.

On larger screens, the layout changes. The arrangement becomes a grid, with the buttons becoming more rectangular. The container is placed horizontally at the center, and it gains a white border.

Lowercase text is displayed on the navigation buttons. I believe it looks more stylish than capitalizing the first letters. The theme persists across other sections. The color of the font is white, this contrasts and compliments the blue background, whilst being easy to read. The font used is Karantina and the background is the same as on the index page.

The navigation still contains debug buttons I used whilst making the app. They are currently hidden as I want to continue developing the app after the project has been submitted.

# **Sections**

All sections have the same base layout. A header is present at the top of the page. Its background color is blue. The font color of the header is white, and the section name is in lowercase. The header also contains a button to close the section. There are two buttons actually, however one of them is hidden according to the display size.

The header is meant to stick to the top of the section but is not currently doing so. The feature worked in earlier versions, changes in the way the layout works being the cause. I intend to fix this bug in the future.

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 bold, and the color dark gray. The text after the title is using a Sans Serif typeface and the same font color as the title. The section’s background color is light gray.

On larger screens, the sections can be repositioned by dragging them. When the user opens a new section, it pops out as a smaller window within the app. This feature causes 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 is comprised of a picture, the name of the species, the hunt method, the start date and encounter total. The species name uses Karantina as the font and has a blue drop shadow. The color of the font is white.

The encounter section features two buttons on either side of it for increasing or decreasing the total. Between the buttons is the total number of encounters.

There are two further buttons after the encounter section. The first shows notes about the hunt. After pressing the **show notes** button, the notes become visible, and the show button disappears. It is replaced by **hide notes**.

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

The font used for all the buttons is Karantina. The buttons are blue, their corners rounded, and the font color is white. In keeping with the recurring theme, I decided to display the **show** and **hide notes** buttons in lowercase. This matches the navigation buttons. The **PAUSE** and **CAPTURED** buttons are in uppercase as they’re *function* buttons. They run a more important task than the others on the page. In the future, I plan to add an **EDIT** button, allowing the user to edit their hunt information.

The **current hunt** section is not fully functional at present. It doesn’t display the correct picture of the hunt, a placeholder displaying the pokémon Bellossom is used instead. In the future I plan to implement pictures of all pokémon so that it matches the selected hunt.

Something else to note – When there is no active hunt (current hunt), the button for 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 as this is currently hard coded).

# **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 similar to the buttons in the current hunt section.

The first field is a regular text field. The user enters their chosen pokémon species. In the future, I want this field to display a selection menu containing all the currently known pokémon.

The second field is a selection menu with 3 options. I plan to expand this menu with more options later. The third field is for the start date of a hunt and the fourth is for any notes the user wishes to record for the hunt. The last field is larger than the others.

I plan to add more fields in the future, for example one where the user can record what Pokémon game they are hunting in.

# **Change current hunt**

The **change hunt** section has a list of the 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 the **current hunt** section. Simultaneously, the current hunt is set to the pokémon that the user chose from the list.

# **Target and shiny lists**

Both the target and shiny lists function the same. They have similar kinds of lists as seen in the **change hunt** section.

# **Showcase**

The buttons in both the target and shiny lists redirect the user to the **showcase** section. The **showcase** section does not have a title like the other sections. Instead, it shows the name of the pokémon species the user selected from either one of the lists. The species font is styled similarly to the font in the **current hunt** section.

Additional information is displayed under the species name in white text fields, these 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 contains information regarding the app, planned future features and currently known bugs.

# **The footer**

Currently the footer of the app is not in use. While using the footer, it causes a bug that disturbs the flow of the app. Currently there is nothing in the footer, so it being missing is not a big deal.

After fixing the bugs I will add some copyright information in the footer. It will be displayed at the bottom of the page after all the other content on smaller screens. On larger screens the position of the footer is also at the bottom, but it’s visible all the time and the position is fixed.

# **Styling the app**

Tailwindcss framework was utilized instead of vanilla CSS to style the application. Using Tailwind aided with development speed as you can style your website or application without leaving your HTML files. Tailwind was installed via their CLI tool. More information regarding Tailwind can be found on their website.

Whilst using Tailwind’s style classes, I tried to avoid repetition. Every HTML element was inspected individually to make sure there wasn’t any. While this was my intention, there is a possibility that some exist. I would put this down to human error. Upon reflection, I realized that I could have placed sections’ font style classes in the main container, under the body tag.

# **JavaScript**

JavaScript was used to make the app’s interactivity. JS is widely known as a scripting language for web pages, however nowadays it is also utilized outside of them. JS contains frameworks and libraries to assist developers with frontend development. I didn’t use any in my project but briefly considered 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 the browser’s memory. The data is stored as a JSON string. From now on, I will refer to 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. The method then 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 could possibly result in 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 and sections’ visibility. At first there was supposed to be only the **toggleSelections** function, hence the function’s if statement. For fear of creating additional bugs, I made two similar functions to avoid this.

**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 the 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 also 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 total 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 have to 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. I’ve then hardcoded what could have been done with **displayCurrentHunt** with a 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 the user selected from the **targets** array. It then 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 forEach 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, it displays 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 yet another function. In the first function the position values are set 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.