Advanced Web Technologies - The Pokedex Web App

Zsolt Varga
40212393@napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Abstract

The Pokedex is a small web app created as a coursework assignment using python and the python Flask framework. It provides a fully functional web API with a number of routes to serve data about monsters from the Pokemon franchise, moves they can learn, and a way to search and filter the data. A large emphasis has been put on the server side logic and the optimization of data, as well as the front-end development. A concise user interface, in the form of a Pokedex device that appeared in the franchse, and a well thought-out routing hierarchy aim to provide functionality and a pleasant user experience.

Keywords – coursework, Edinburgh, napier, university, zsolt, varga, web, technologies, front-end, back-end, server, flask, js, javascript, html, css, Pokedex, app, website, dynamic

1 Introduction

The Pokedex project is a small web app for browsing the Pokedex - a collective term for all Pokemon from the video game and cartoon series of the same name. The web app serves as a tool to check specific information about these Pokemon and moves, and perhaps as a learning tool to refresh ones knowledge of the Poke world.

This exact Pokedex version reflects the contents of the first generation of the games and contains 151 creatures, as well as their details, and 12 Pokemon types. It also serves as a collection of moves/attacks these creatures are able to learn. There are 165 of these, and similarly to Pokemon, these are too divided between the 12 types and additionally, each have one of three categories assigned.

While creating this application, originality and great user experience were kept in mind. Our goal was to create a web app that is straightforward to use, looks aesthetically pleasing, contains accurate and easily readable information, and contains nostalgic elements from the Pokemon games and TV series. For simplicity, only the first generation (151 as opposed to the current 802 creatures as of 2018) was implemented.
The whole website resembles the actual pocket device for identifying the monsters (see figure 11). The screen of this device serves as the window for displaying content.

All images of Pokemon and their sprites are the property of The Pokemon Company. These images were downloaded from veekun.com [1] and fandom.com.[2]
To obtain textual information about Pokemon, several web-scraper tools were created using the BeautifulSoup python library. As an ethical approach to this form of obtaining data, timing of several seconds was set between each request not to overload any servers, and only publicly available data was scraped [3] - mainly from the Bulbapedia Pokemon wiki page.
Considering web crawling is not the focus of the assignment, we will not be examining this part of the project in this report.

2 Design

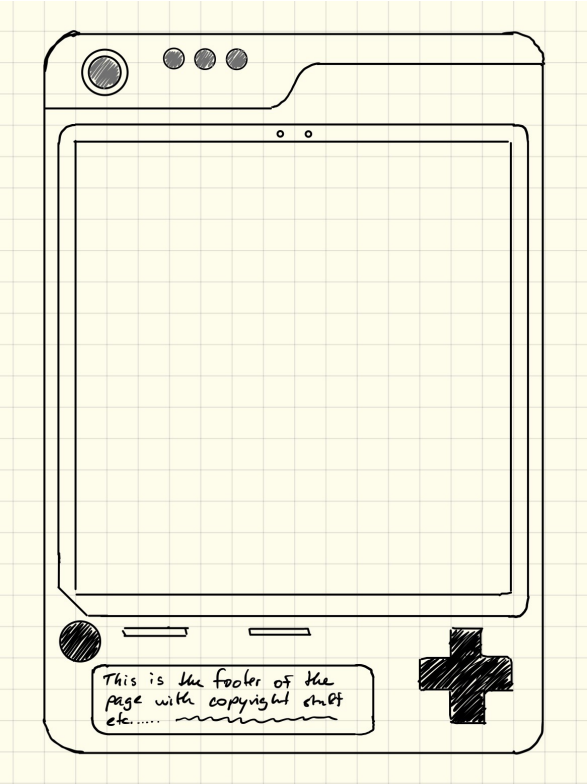

Figure 1: Initial sketch of the Pokedex frame

2.1 User Interface

As has been previously mentioned, the central design element is the window frame in the form of a simplified Pokedex device from the games. In addition to this, a blurred background of an outside setting has been added to give the feeling of using the device in nature - to catch Pokemon. The bright hues of the green and the saturated red of the frame are very resembling of the colours used in the games. The whole frame is more or less responsive, to the extent of responding to the window width, generally speaking however is a very

static element within the dynamic environment.
Despite this fact, it took extensive planning and sketching to reach the desired effect - see Figures 1 and 2.

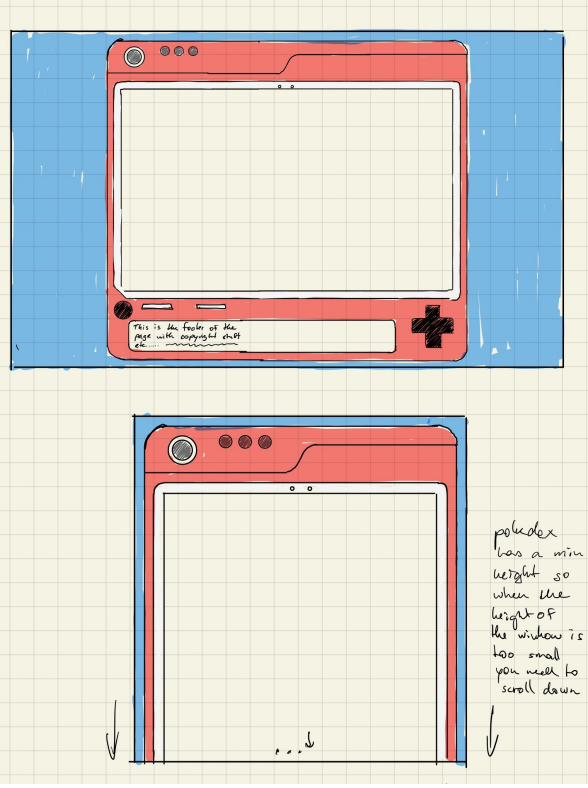

Figure 2: Initial sketch of the Pokedex frame in different window sizes

Regardless of the possibility to use CSS libraries for the project, one of the main goals was to create an original experience. While bootstrap is highly customizable, our challenge was set to create an environment using plain CSS, HTML, and javascript, and to achieve details and nostalgic design elements.

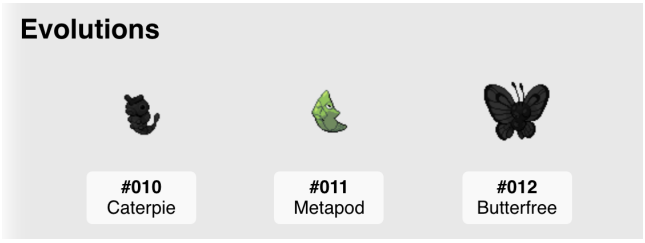

Figure 3: The Evolutions section of a Pokemon page

In this context, any feature that reminds one of the Pokemon franchise could be considered nostalgic. Besides the obvious red frame with buttons and diodes, a few other less obvious elements were introduced. For example, within a Pokemon's page (see Figure 12), the evolutions section indicates the number of evolutions and a direct link to the page of the evolution. However, the linked Pokemon's sprites have been desaturated and darkened (Figure 3 so as to resemble the "Who's that Pokemon?" section of the TV series, where the viewer is prompted to guess the name of the monster. Similarly, we can only see the silhouette of the Pokemon before following the link.

There are many other similar visuals, some of the more notable ones being the Poke-center symbol used as the link to the index page (again, created using plain CSS), sprites taken from the games, and type labels colored the same way they are in the games.

Another main functionality of the web app is the ability to view the whole Pokedex (Figure 4) and to filter it down based on Pokemon attributes. Along with the function of generating a random Pokemon further support the learning aspect of the app and give the user a better understanding of the way the Pokedex is structured.

Figure 4: View of full Pokedex

2.2 Routing and URL's

Coming up with the API's routing hierarchy was very dependent on the structure of the data. At first, only the Pokemon were implemented. Each of them has several attributes that are at the same level. These include Pokedex number that serves as a primary key, and type (most importantly), then an alias name, measurements, description and a list of learnable moves. This lead to the basic routes "/Pokemon" that branches out to "/Pokemon/name", "/Pokemon/types", and "/Pokemon/types/ type"

The structure for moves takes on a similar structure, as each move also has its number, name, type, as well as a category. Regardless, there are some differences due to the fact some information is less important. Since we are displaying the type of each move within the moves table in the the "/moves" route (see Figure 17) it would be abundant information to include a "/moves/types" route. The fact that move types are identical to Pokemon types also supports this conclusion. Therefore, the branch within the hierarchy has been 'skipped' and only the specific "moves/types/type" was implemented. Same applies for the move categories. See figure 18

See Figure 10 for a table of all routes with a short description.

For this stage of the assignment, only GET routes have been implemented, since we're mainly retrieving data. In order to

implement functionality of adding a new Pokemon or move to the dataset we would develop support for PUT and POST methods.

Figure 5: Rendered template for a 404 error code

Besides building a base routing hierarchy, several redirects have been developed in order to make navigation through the URLs more efficient. As there are several interchangeable terms in the Pokemon franchise, an effort has been put into reflecting this within the routes and make all synonymous words usable. For instance, the collective name for all Pokemon is Pokedex, as opposed to Pokemons (which is also grammatically incorrect), hence the route "/pokedex". However, the route for a single Pokemon is "/pokemon/name". Considering the Pokedex number of a Pokemon and the name are both primary keys, they can both be used to refer to a specific creature. Therefore, all of the following combinations redirect to the specific "/pokemon/name" route:

- /pokemon/dex_number
- /pokedex/name
- /pokedex/dex_number

This way we ensure routes are very interconnected and prevent 404 status code errors

2.3 Error Handling

Some basic error handling has been developed in the form of custom templates viewing the error code within the web-page alongside a custom apologetic message (Figure 5).

2.4 Data Storage

One of the very initial stages of development was the planning of necessary data structures and their inner format as well. Since the scope of the whole data is in the hundreds and the website does not have user support, storing in json was an easy pick. A more important decision was how this data should be organized in order to remove as many duplicates as possible and to optimize it to a certain extent. For this reason, a decision has been made to create several json files, where each mimics a relation (from relational database theory) - hence the Pokemon, moves, and types files. Within each file the data is stored as an array of objects with their own primary keys (see example on Figure 6).


```
[{
    "dexnum": 1,
    "name": "Bulbasaur",
    "type": ["Grass", "Poison"],
    "alias": "Seed Pok\u00e9mon",
    "description": "Bulbasaur is a small, quadruped Pok\u0
    "moves": ["Tackle", "Growl", "Leech Seed", "Vine Whip"
    "evolutions": ["Bulbasaur", "Ivysaur", "Venusaur"],
    "height": 70,
    "weight": 6.9,
    "catchrate": 45,
    "hatchtime": [5140, 5396]
}, {
    "dexnum": 2,
```


Figure 6: A look inside the pokemon.json file - dexnum and name both serve as primary keys in this case

To connect this data to the flask app, we have created a class called PokeData inside a dataconnection.py file, that we then imported and instantiated within the Pokedex.py file. The PokeData class contains all functions for processing the data and returns only the necessary data in the form of dictionaries or arrays of thereof. For an example see listing 1.
This way, we enforce separation of concerns, and make the source code cleaner in overall. [4]

Listing 1: Code snippet of a method making use of the PokeData class to retrieve a name of a Pokemon

```
1  pokedata = PokeData()
2
3  @app.route('/pokemon/random')
4  def random_pokemon():
5      number_of_pokemon = len(pokedata)
6      dexnum = random.randint(1,number_of_pokemon)
7      pokemon_name = pokedata.nameByNum(dexnum)
8      return redirect("/pokemon/{}".format(pokemon_name))
9
```

3 Enhancements

There are numerous ways of improving the web app, some of which are minor changes, and others huge alterations of functionality. Here we will explore these options as possibilities for future versions of the app.

3.1 Routing and Redirects

The routes quite clearly reflect the way our data is structured; however, some smaller additions could fill the gaps. For instance, as we mentioned before the routes for "/moves/-types" and moves/categories are missing since this is abundant information. Regardless, we could use these routes to redirect to a more logical location.

3.2 Search Function

The ability to search the data was one of the initial plans for the app, however had to be scratched due to time constraints.

This way, we could interact with our Pokemon information in a whole new way.

3.3 User Support

For user support, we would most probably need to change our data storage and introduce a database with sufficient hashing. However, having the ability to recognise distinct users would introduce a plethora of new functionality. For instance, saving favourite Pokemon, a tool for building one's teams of Pokemon and comparing Pokemon, a pet Pokemon mini-game, a chat for interacting with others could all form the purpose and usefulness of the web app. Implementing all these features would be outside the scope of the module, but are a possibility for further direction.

3.4 Security

Strongly tied to the user accounts - security is a big concern for a web app. Since our website only really supports GET requests no measures were taken for securing our flask app. However, for any PUT/POST/DELETE etc. requests, we would need to make sure to take action against code injections and similar attacks.
One of the current vulnerabilities at this current stage of the project is the filter function for the Pokedex. This was done by storing the arguments inside a dictionary which is then passed along to the PokeData object to retrieve a filtered Pokedex in form of an array. Since we are not protecting the passed in data, this could potentially pose as a security threat. Instead, the session library should be used to create a cookie with the filter attributes.

3.5 Design Enhancements

Since the Pokedex frame already serves as a very dominant design element, it would be great to functionaliise it. The fake buttons on the device could serve as an fun alternative way of navigating throught the web app.
Also, despite the frame being a little responsive, there is a lot of space for improvement. For mobile devices for instance, the whole Pokedex device should be replaced with only the inner white screen bezel, as was originally planned (see Figure 7).

3.6 Error Handling

Some amount of error handling has been implemented. For instance, 404 error codes have been considered and on top of having a custom Jinja templates that seamlessly fit within the design, each route is designed to abort with a 404 error when necessary - see Figure 5.
However, other error codes do not have similar handling as of the current state of the project.

4 Critical Evaluation

Objectively speaking, the flask app has several very useful features that work well, however, there is space for improvement in several areas, and these areas will most likely be addressed in future versions.

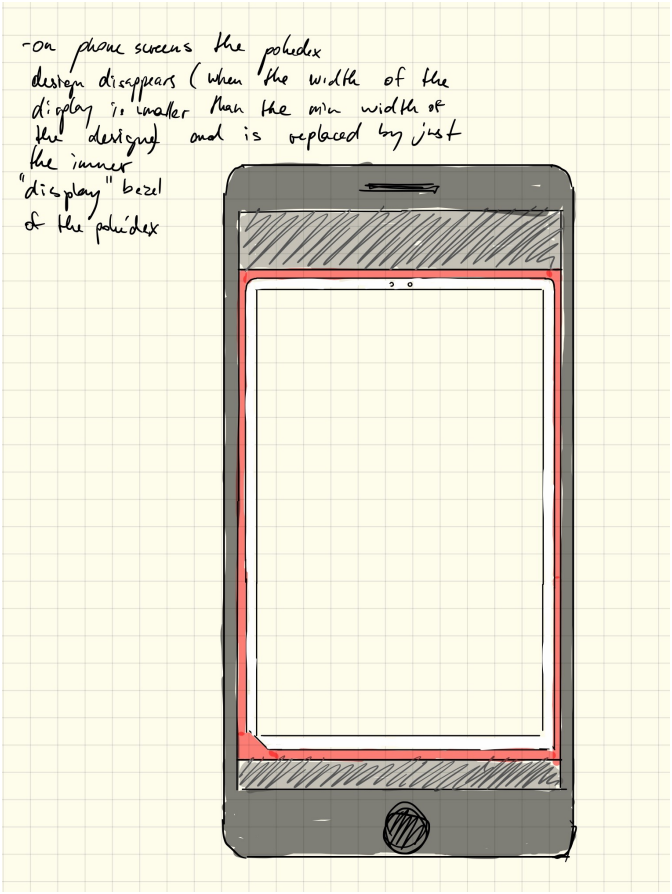

Figure 7: Initial sketch of the Pokedex frame reduction on a mobile screen

4.1 The Pokedex Page and Filter

One of the most important routes and functionalities of the whole app is the Pokedex page itself. This page provides an overall view of all Pokemon, with some of the most important information about the Pokemon - such as Pokedex number, sprite, type, and name.
Since the number of Pokemon is rather large, we have decided to place them in a tile-like table, where each cell contains the compressed data about the Pokemon. Considering how small these areas are, a clever way to simplify some of the data was to replace the typing with small colored dots. These colors are the type colors Pokemon players are familiar with and at the same time prevent the overflood of text. See figure 4.
Furthermore, each of these 'tiles' provides a direct link to the Pokemon's page where the user is able to find out more information about the monster.
The more ways to interact with the data the more types of users we might be able to satisfy. Therefore, a filter option has been included within the Pokedex page (Figures 8 and 9). This way, the Pokedex may be narrowed down based on any combinations of attributes the user pleases.

One of the possible improvements to the filter might be to replace the type selector with a button for each type, so the filter may take any number of types (1-12) as opposed to just 1 and 12(all).

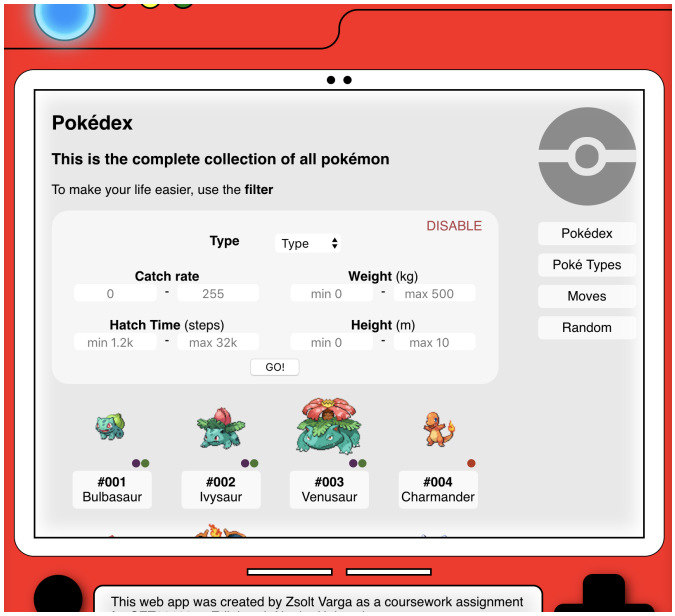

Figure 8: Opened filter in the pokedex page

Figure 9: Filter results for Electric type of maximum 10kg weight and maximum 2m height

4.2 Concise Design

The overall design came out looking very pleasant and nearly identical to what had been planned. There were several initial sketches that can be seen in figures 1, 2, 7, 12, and 13. The initial intent was to use a broken down image that would overlap and be able to respond to the window size, but ended up writing everything in CSS.
Every page of the web app makes use of the Pokedex frame and thanks to Jinja templating we managed to avoid a lot of duplicate code. There were other instances of templates being put to use, for example only one Pokedex page was created, and several other pages that include a Pokedex grid make use of this template.
Small alterations between similar elements of pages are what makes the web app interesting, such as the "Who's that Pokemon?" darkening of sprites, for example.

The overall design is not break-through, but provides a great user experience. As we mentioned before, adding more interactivity to the main elements would be a great way to engage our target audience, which is primarily those young at heart.

4.3 Data Hierarchy and Routes

The data currently stored and interacted with in our web app is rather simple in terms of hierarchy deepness. There are no cases of the data being nested deeper than three layers, therefore it would be a sensible and great addition to "grow this tree". We could do this by adding a generation layer, which would essentially multiply the size of our data seven-fold, as there are currently seven Pokemon generations.
We could do this by having seven different versions of json files (some Pokemon types change, monsters and moves, completely new features and complications are added etc.). As for the UI, this could be solved by creating a simple 7-stop slider on the very top of our web app interface.

There are a few other things that could be corrected with the routes, especially the 'moves' part of the hierarchy. For instance, all three routes "/moves", "/moves/types/type", and "moves/categories/category" use the same template, so it might not be obvious to the user where in the tree they currently are. This distinction could be drawn by simply adding additional data to provide more information than the "/moves" route as opposed to just narrowing down the table of moves.

5 Personal Evaluation

The usual workflow for this particular assignment consisted of working on the app locally inside a virtual environment of a python version compatible with the university's development server. Making frequent git commits, once a semi-releasable version has been created, this version was pulled onto the remote server to check if everything works as it should. This process was repeated until the current version of the software, and occasional remote repo pulls made sure all changes work on both the local device and the server.

5.1 Obstacles and Takeaways

This project not being my first attempt at creating a functional API, the takeaways and learned skills were surprisingly high in volume.
First of all, using git after every atomic change to the source code became a habitual task, which is a good habit to have in case of system failures when working on a personal project locally. Other than that, being able to jump back to a very specific version of the web app is a great way to compare functionality and pick the best solutions.

There were a few challenging aspects to the project, and not every immediate solution turned out to be effective at a later stage of development.
For instance, when creating the json files, the first approach was to create an object of nested objects, where the key

served as a primary key. This turned out as a bad idea, since python dictionaries don't guarantee the objects to stay in the same order[5], meaning the list of Pokemon seemed to be random once we tested the app on the remote server. Therefore, at a later stage the format was changed to an array of objects which solved the issue, simplified the logic of retrieving ordered data, and removed substantial python syntax from Jinja templates.

Other obstacles included the routing hierarchy itself. Several changes were made to the naming of routes and the way these routes relate to each other. The filter functionality turned out to be more challenging than expected.
All these kinds of issues provided a closer and more in-depth understanding of the data itself and how python data types work.

The last personal challenge was to avoid using any styling libraries and create a UI using pure CSS, HTML, and javascript. This small challenge actually taught some valuable lessons, mainly about how CSS works, but also served as a way to make the project more unique and worthy of adding to the portfolio.

6 Conclusion

In overall, we have managed to create a successful web application using the Flask framework through creating a functional web API and connecting it to the front-end. The API uses several routes organised in a logical hierarchy, to provide data about Pokemon and Pokemon moves. The front-end uses several technologies including HTML, javascript, and CSS, and utilises the Jinja templating engine to render the dynamic content provided by the API.
Several lessons were taken away from this project, including a better understanding of python, the Flask library, routing, and git.

References

[1] Veekun, "Downloads." https://veekun.com/dex/downloads. Accessed: 2018-10.

[2] Fandom, "First generation pokÃľmon images." http://nintendo.wikia.com/wiki/Category:First_generation_PokÃľmon_images. Accessed: 2018-10.

[3] C. Steele, "The modern python 3 bootcamp." https://www.udemy.com/the-modern-python3-bootcamp/learn/v4/overview. Accessed: 2018-07.

[4] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. U.S.: Pearson Education, Inc., 2009.

[5] J. Portilla, "Python and flask bootcamp: Create websites using flask." https://www.udemy.com/python-and-flask-bootcamp-create-websites-using-flask/learn/v4/overview. Accessed: 2018-09.

REDIRECT	ROUTE	NOTE
	/pokedex	Requesting all Pokémon (Pokédex = all Pokémon)
	/pokemon	
	/pokedex/<number>	Requesting a single Pokémon
	/pokemon/<number>	
	/pokedex/<name>	
	/pokemon/<name>	
	/pokemon/types	Requesting all types for Pokémon
	/pokemon/types/<type>	Requesting a single type of Pokémon
	/pokemon/random	Requesting a single random Pokémon
	/moves	Requesting all moves
	/moves/<move>	Requesting a single move
	/moves/types/<type>	Requesting moves of a certain type
	/moves/categories/<category>	Requesting moves of a certain category

Figure 10: List of routes with description

Figure 11: Comparison of Pokedex frame and Pokedex device as appeared in the TV show

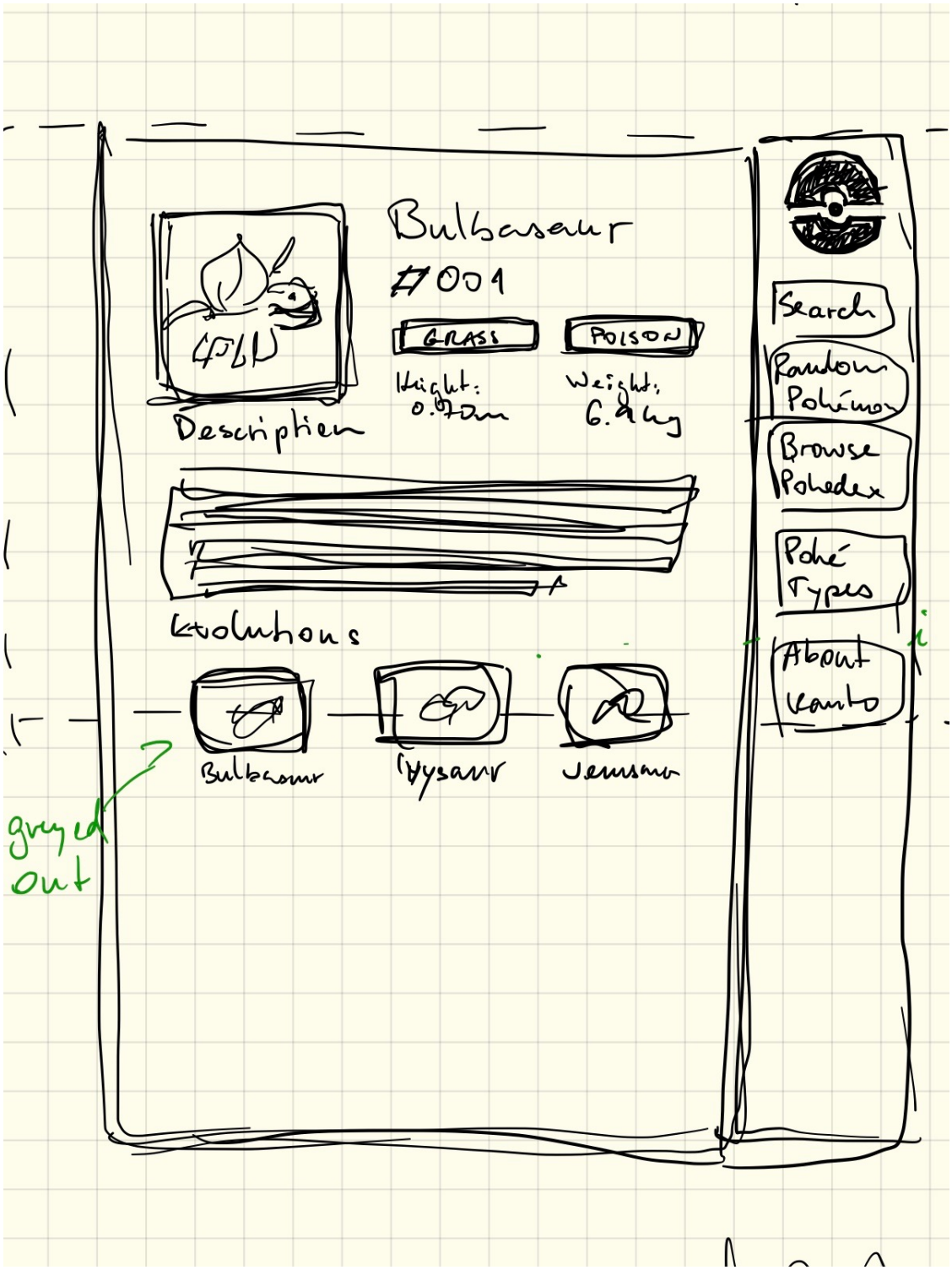

Figure 12: Sketch of a Pokemon's page layout

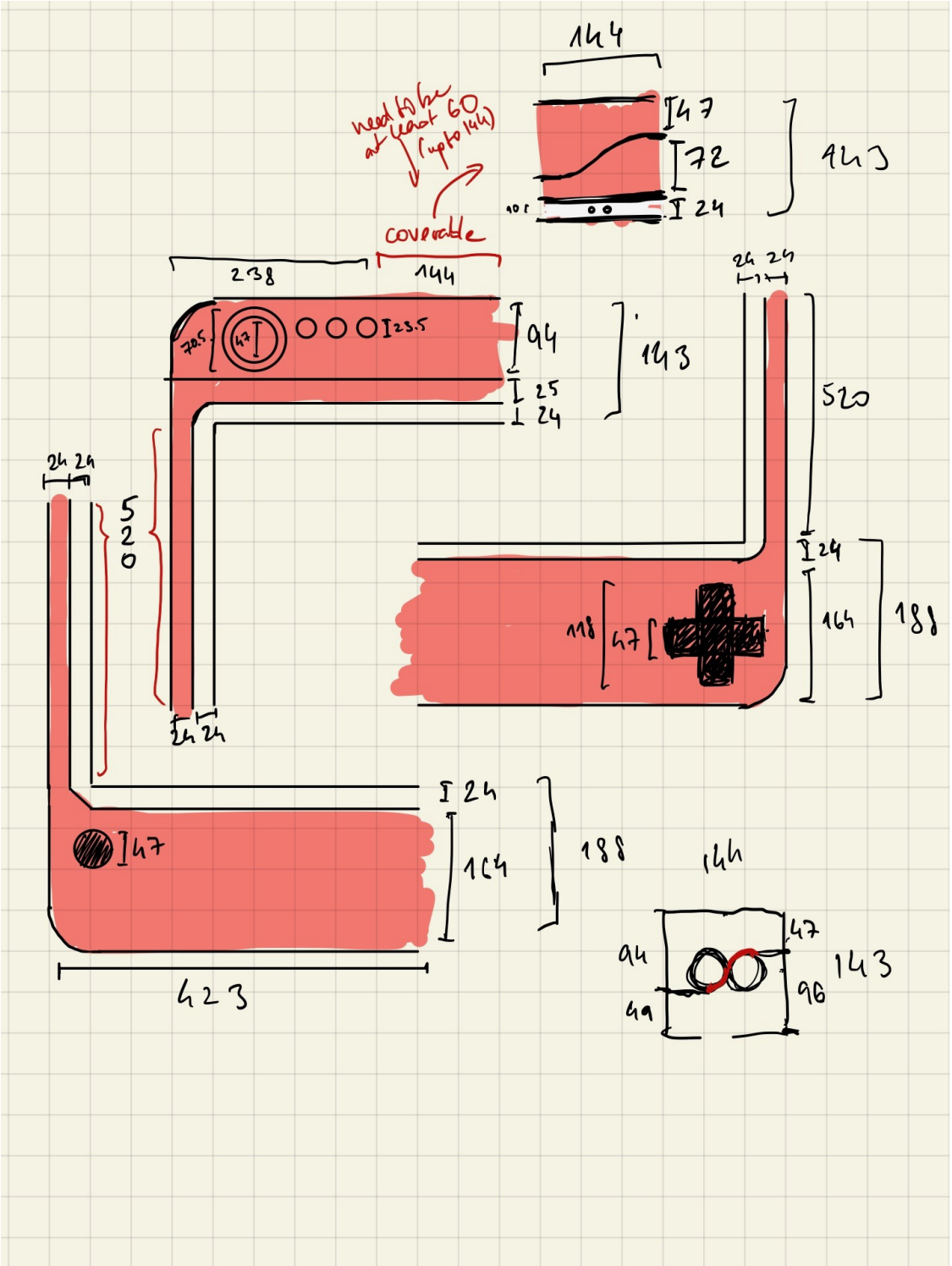

Figure 13: Plan of how to create the Pokedex frame with pixel calculations

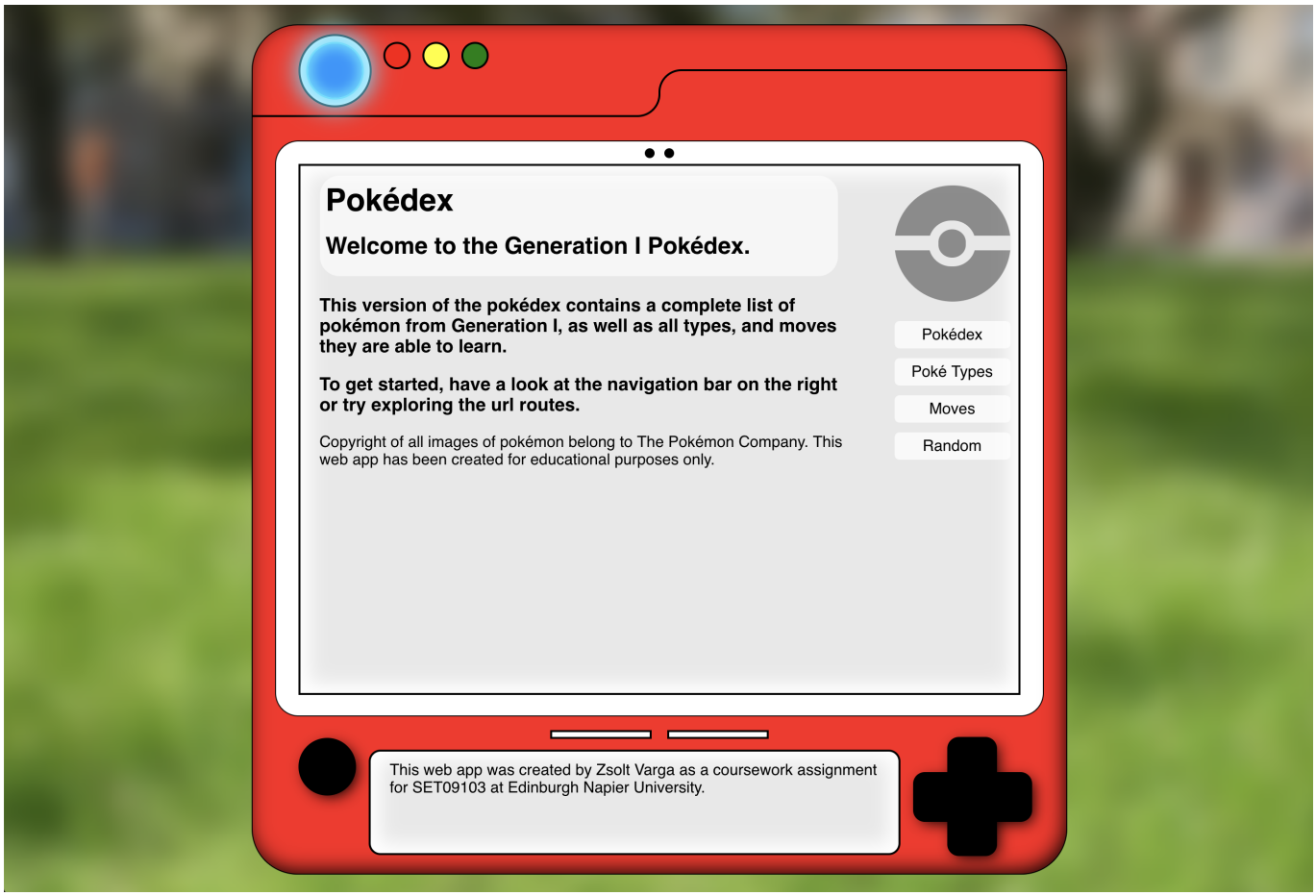

Figure 14: The welcome/index page

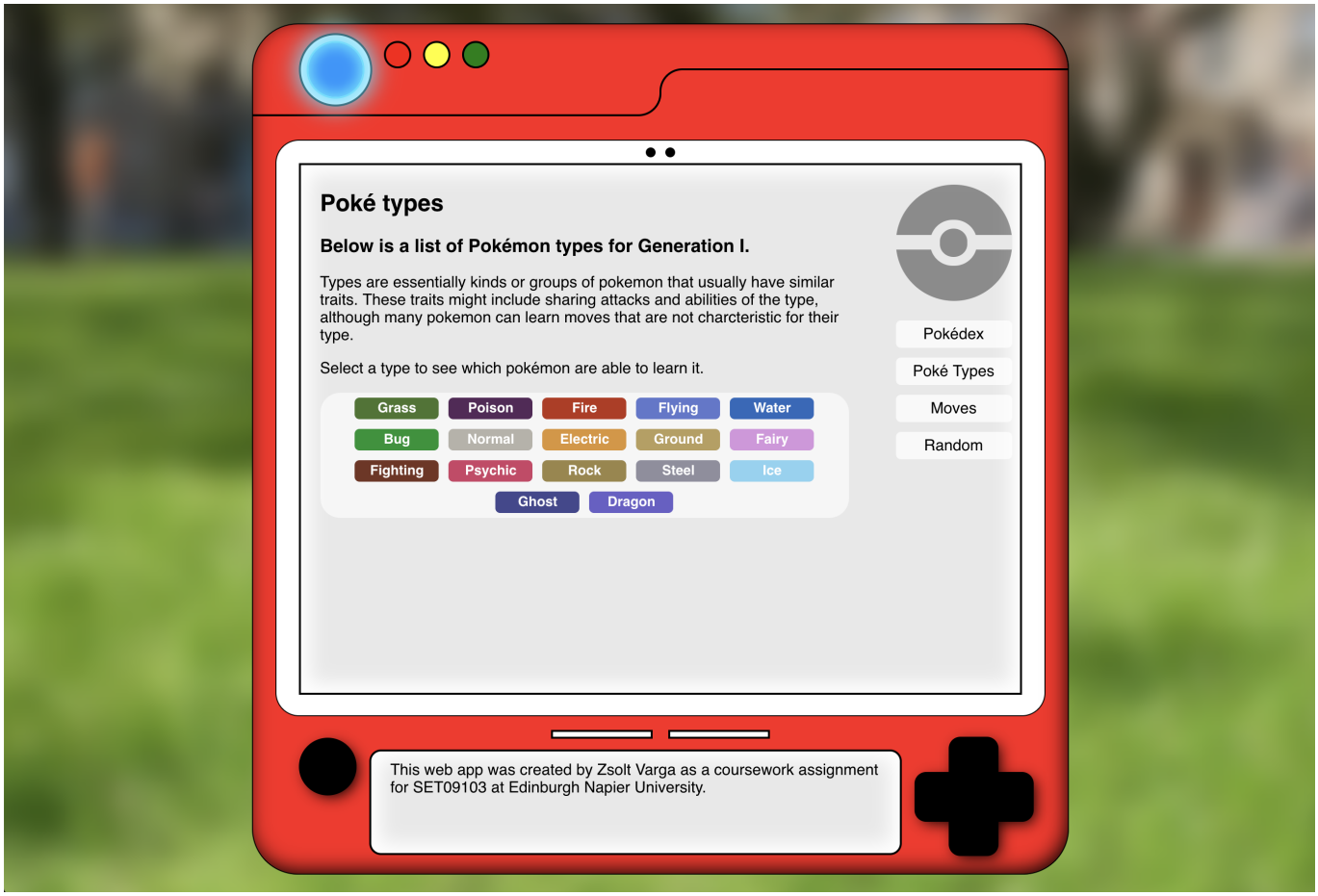

Figure 15: Page for the types route

Figure 16: Rendered template for the Ice Type Pokemon

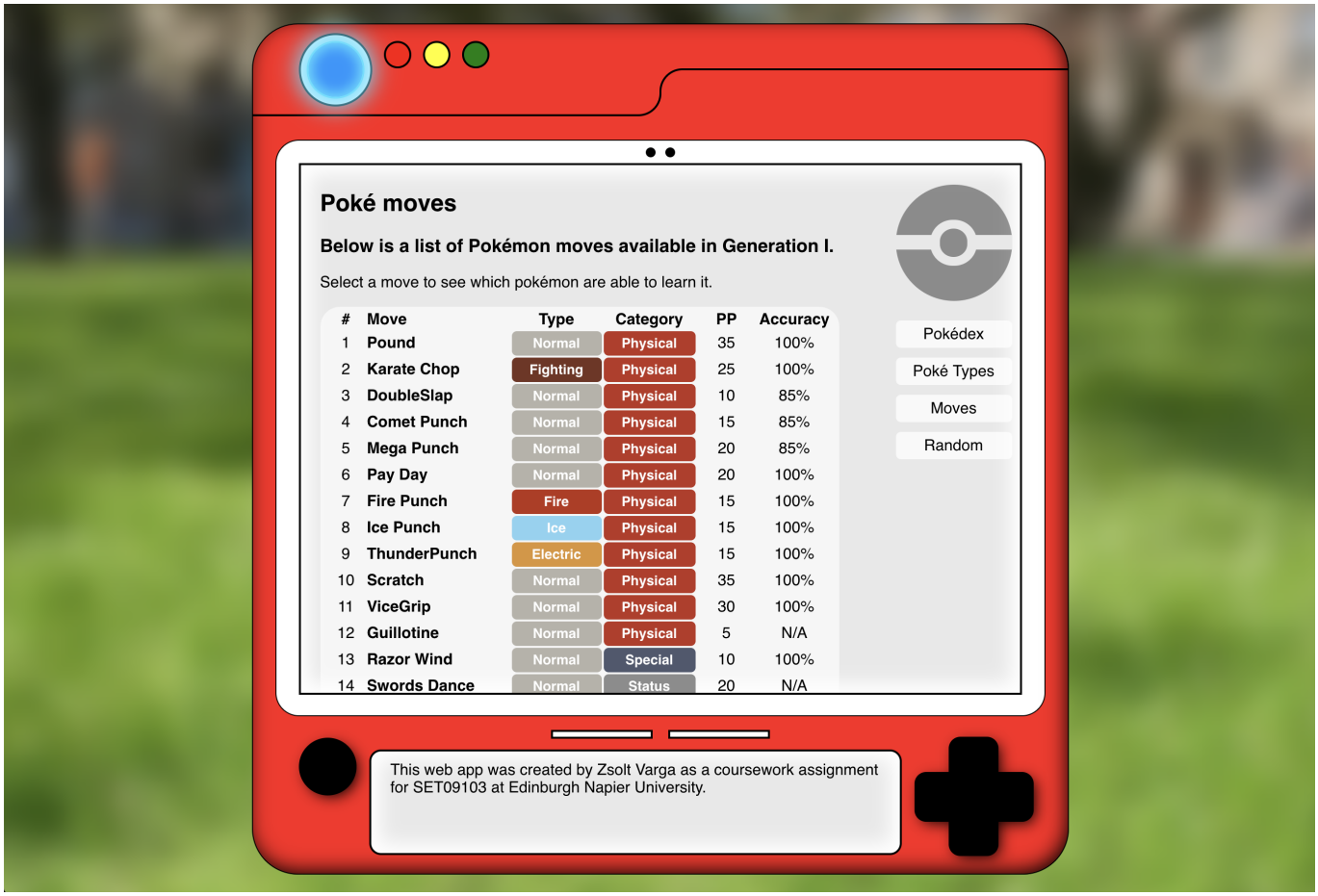

Figure 17: Page for the moves route

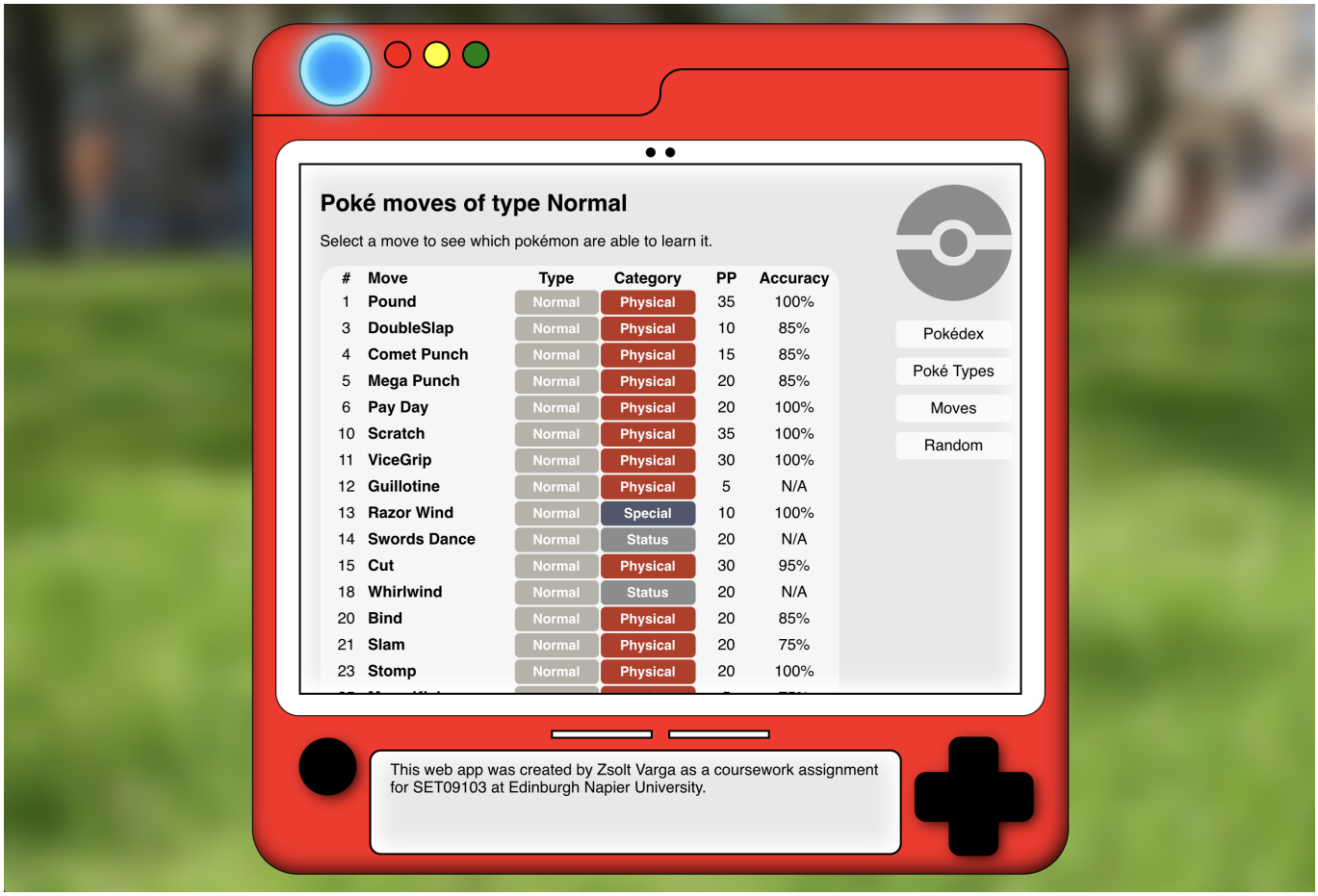

Figure 18: Page for the Normal type moves