University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 3

Date: 2023-03-14

Group Number: 108

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Sahil Thind	92272954	l4c0t	thindsahil3@gamil.com
David Sopheap	24296634	g0g3b	david.sopheap@yahoo.ca
Zach Taylor	48297956	l1n5s	taylorzachary8@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Description:

Our project attempts to allow users to look-up information on all ~1000 Pokémon, and most game-relevant data points pertinent to them, namely, their names, stats, moves, items and abilities. These data points will be categorized based on data relevant to them, namely power, type, and effect. It will also be possible to categorize them into teams that users can then save and load in the future.

Task Breakdown:

1) **(David,** Sahil**)**

First, we have to find a fast way to populate the data into tables in the first place, because there are just so many data-points. My first priority will be scraping data from various sources online (scraping HTML files essentially) and then automating the entries for the major databases; data-points like Pokemon, learnsets, abilities, stats, items, and all their descriptions can be done this way.

2) **(All)**

This leaves user interaction, which is where the entirety of the query-related spec guidelines must be met.

First, we must design a (admittedly shallow) account system which users create that can track teams under that account; adding teams to accounts and removing accounts will take care of the INSERT and DELETE requirements respectively.

Next, we need a system that allows users to add up to six Pokemon to a team. They will be able to choose their moves, items, and abilities accordingly by selecting them via various queries to the database. This feature meets the UPDATE requirement. Then, implementing sorting and filtering features on data will entertain many other requirements. For one, sorting attributes as the user desires to be alphabetical or numerical should be possible for all data-points in the main database, and certain broad filters should also be able to be applied to sorts. For example, one user might want to find all grass and poison types and sort them by highest defense stat. Another may want to order all Pokemon, and only view their speed stats. Another user might want to find the largest special attack stat of a Pokemon that learns the move Sludge Bomb. Queries like the first one will meet the SELECT, and JOIN queries. Queries like the second will meet the PROJECT requirement, while queries like the third meet the DIVISION and AGGREGATE requirements.

Special attention must be paid to meet the NESTED AGGREGATE requirement, since adding too much user freedom for every aggregate combination will likely take too long.

3) **(Zach,** Sahil**)**

For the frontend, we will make a web app using React. The web app will include a menu window, which will allow users to log into their account. Upon logging in, users will be met with a list of existing teams (names) stored in their account as well as view, 'create' and 'update' buttons to perform actions on those teams. Clicking on the 'view' button displays all the Pokemon including moves, abilities and items in the selected team. Clicking the 'create' button brings the user to a new page where they can perform all actions related to creating a new team. This page will contain:

I. an 'add' button to add a Pokemon to the team. Clicking the add button brings a dropdown list of all the Pokemon from which the user can choose, either by scrolling through the list or searching.
   A. Clicking the 'filter' button next to add will bring a list of filters that can be applied to the Pokemon list. Filters include things like sorting the list alphabetically or by base stat total, including only Pokemon of certain type, learn certain moves, etc. A corresponding query is run in the database to update the results.

II. four input boxes to choose moves for the Pokemon. Only moves which the added Pokemon can learn are shown in the dropdown list of moves.
   A. Next to moves input boxes, a button exists to apply filters to the moves dropdown list. Filters include things like only specific type moves, ordering by base power, etc.

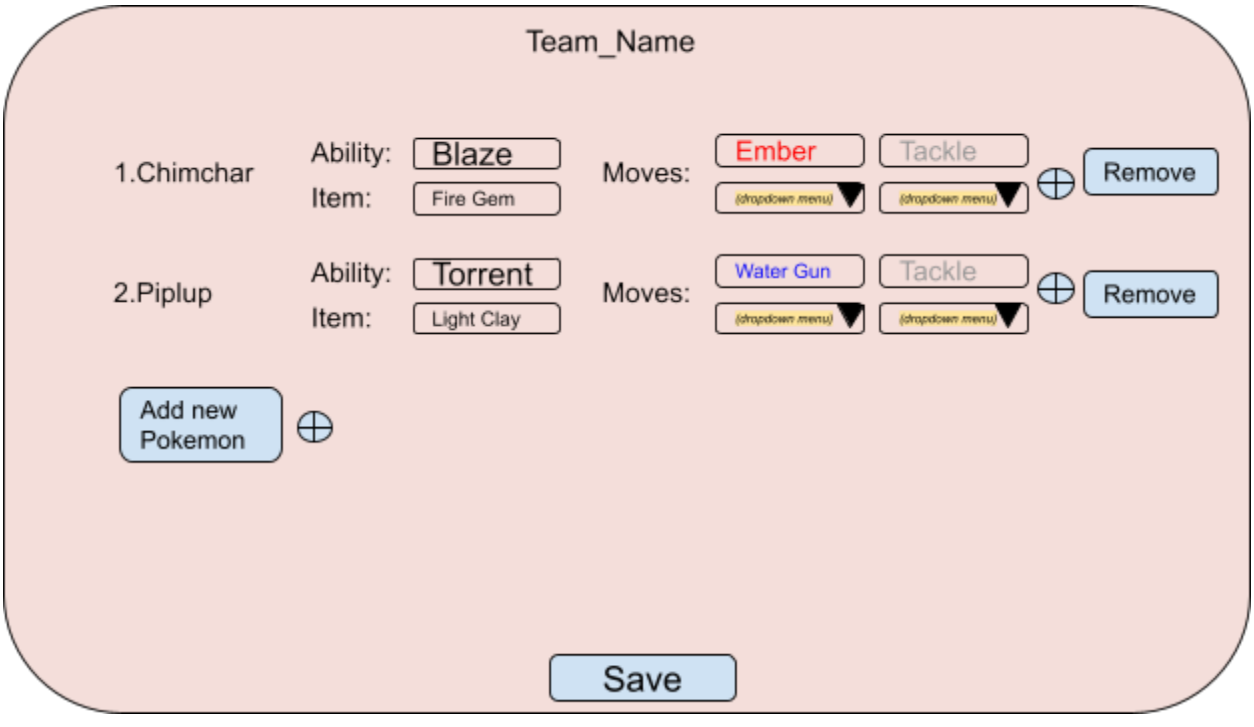
III. an input box to add an ability. Only abilities which the added Pokemon can learn are shown in the dropdown list of abilities.

IV. an input box to add a held item for the Pokemon,

V. a 'remove' button to remove the newly added Pokemon. The user can add up to 6 Pokemon.

VI. a 'save' button to save the current team and add to the user's account. To save, the user must have at least 1 Pokemon in the team, as well as every Pokemon must have at least one move and a single ability. A warning is displayed if these conditions aren't met.

The Add Team page can look something like this (very rough draft):



Clicking the ‘update’ button brings the user to the Add Team page but with the existing Pokemon in the team automatically populated.

Timeline:

1) **(Less than a week, started concurrent)**

This should not take considerable time; we expect it to take about three or four days to set up the repo, build the scraper, and automate the database populations. This can also obviously be done concurrently with the UI.

2) **(3 weeks, started concurrent)**

This will take up the majority of construction time; pretty much the remainder of the three weeks will be devoted to this and the UI creation. Fortunately, this can be started while 1) is in progress, again with mock data. The creation of the account and team frameworks should be fairly easy, but building a comprehensive system of query and testing it from the data stored will likely take a lot of wrangling, and is difficult to formally test (so we likely won’t go overboard on comprehensive tests ala say, the CPSC 310 term project that involved hundreds of tests).

3) **(2-3 weeks, started concurrent)**

This can take place during 2) and use mock data, so this will take as much or as little time as we require. We plan to put everything fancy on the backburner; it’s probably best to focus on text, dropdowns, search bars, and tabs for team interaction. Sprites and graphics will only happen if we have enough time to automate that too. Altogether, this may take the entire three and a half weeks, or it may take half that.