**Design and Development of a Dynamic Web Application**

Submitted by:

Jean Karlo S. Sioson

Gian Jerome Pascua

Submitted to

PROF. Paul Jason Perez

In partial fulfilment of the requirements for the course

LIS 161 - Information and Communications Technology in LIS II

2ND Semester, A. Y. 2022-2023

School of Library and Information Studies

University of the Philippines Diliman

University of the Philippines System

JUNE 2023

Documentation:

**Intro**

For this project, our group decided to recreate a generation 1 Pokedex similar to the manga and the videogames of Pokemon. We decided that it was a fun take on designing and developing a dynamic web application that utilized a database. We also had to limit the pokemon considering there are more than thousands of Pokemon in the latter generations, so we limit ourselves with the original 151 Pokemon that you could find in games Pokemon Blue and Red. Both of the members of this group are very new to Python and coding in general so we decided to focus on function while keeping a minimalistic style since we are both inexperienced but we still wanted to give a stylish looking webpage. We took great guidance from the example that was provided which was named “Adopt a Pet” so you will see a lot of similarities in the structure of the code.

**Creation of app.py and data.py**

As stated before, we modeled our website in a way that resembles the Adopt a Pet example given. A brief explanation of all the functions and features of the code are as follows:

First off we will discuss the **app.py** file which contains the code for a Flask web application. Flask is a popular Python web framework used for developing web applications. This file is responsible for importing necessary modules which it does by importing the required modules from the Flask framework and the data module. It then creates an instance of the Flask application as instructed in the module of flask templating wherein an instance of the Flask class is created and stored in the app variable. Now the code defines routes and associated functions:

* The '/' route is associated with the index() function, which renders the index.html template.
* The '/pkdex/<pkmn\_type>' route is associated with the pkdex() function, which takes a parameter pkmn\_type and retrieves a list of Pokémon based on the provided type from the data module. It then renders the pkdex.html template, passing the pkmn\_type and Pokemon list as variables.
* The '/pkdex/<int:pkmn\_id>' route is associated with the pkmn() function, which takes a parameter pkmn\_id and retrieves a specific Pokémon based on the provided ID from the data module. It then renders the pkmn.html template, passing the pkmn object as a variable.
* The '/register' route is associated with the register() function, which renders the register.html template.
* The '/processed' route is associated with the processing() function, which is triggered when a form is submitted via the POST method. It extracts the form data (Pokémon type, name, description, and URL) and inserts it into the database using the insert\_pkmn() function from the data module. After successful insertion, it redirects to the pkdex route, passing the pkmn\_type as a parameter.
* The '/modify' route is associated with the modify() function, which is triggered when the modify form is submitted. If the form contains the value "edit," it retrieves the Pokémon based on the provided ID and renders the update.html template, passing the pkmn object as a variable. If the form contains the value "delete," it retrieves the Pokémon based on the provided ID, deletes it using the delete\_pkmn() function from the data module, and redirects to the pkdex route, passing the pkmn\_type as a parameter.
* The '/update' route is associated with the update() function, which is triggered when the update form is submitted. It retrieves the form data (Pokémon ID, type, name, description, and URL) and updates the corresponding Pokémon in the database using the update\_pkmn() function from the data module. After a successful update, it redirects to the index route.

Lastly, when we running the application: The if \_\_name\_\_ == "\_\_main\_\_" condition ensures that the application is only run if the script is executed directly, not imported as a module. Finally, the app.run(debug=True) line starts the Flask development server with debugging enabled.

Now we move on to the data.py file which is a script that interacts with a SQLite database containing Pokémon data. It includes functions for connecting to the database, reading Pokémon based on type or ID, inserting new Pokémon, updating existing Pokémon records, and deleting Pokémon from the database. It does this by the following codes:

* The connect\_to\_db() function establishes a connection to the SQLite database specified by the db\_path variable. It returns the connection (conn) and cursor (cur) objects, allowing further interaction with the database.
* The read\_pkmn\_by\_pkmn\_type() function retrieves Pokémon from the database based on their type. It executes a SELECT query with the provided type as a parameter and returns the results.
* The read\_pkmn\_by\_pkmn\_id() function retrieves a single Pokémon from the database based on its ID. It executes a SELECT query with the provided ID as a parameter and fetches the first result.
* The insert\_pkmn() function inserts a new Pokémon into the database. It takes a dictionary (pkmn\_data) containing the Pokémon's type, name, description, and URL. The function connects to the database, executes an INSERT query with the provided values, and commits the changes.
* The update\_pkmn() function updates an existing Pokémon record in the database. It takes a dictionary (pkmn\_data) containing the updated Pokémon information, including the type, name, description, URL, and ID. The function connects to the database, executes an UPDATE query with the provided values, and commits the changes.
* The delete\_pkmn() function deletes a Pokémon record from the database based on its ID. It connects to the database, executes a DELETE query with the provided ID as a parameter, and commits the changes.

These functions provide basic CRUD (Create, Read, Update, Delete) operations for interacting with the Pokémon database using SQLite as per the parameters of this project. They abstract away the database connectivity and query execution, allowing other parts of the application to easily interact with the Pokémon data.

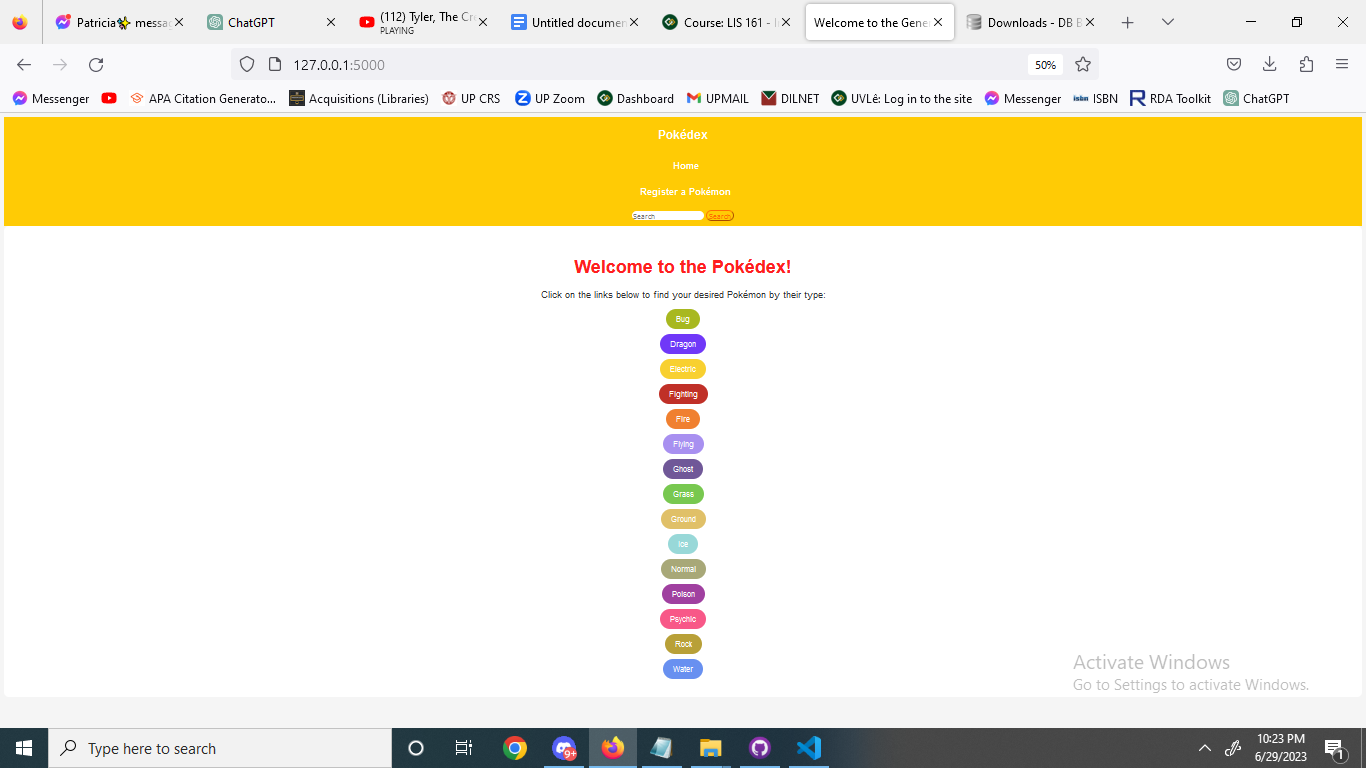
**Explanation of the HTML FILES:**

1. base.html: This is the base template that serves as the layout for all other pages. It includes a navigation bar, a content block, and the necessary Bootstrap CSS and JavaScript files.
2. index.html: This template extends the base.html template and represents the homepage of your PoKedex application. It displays a list of links for different Pokemon types.
3. pkdex.html: This template extends the base.html template and is used to display a list of Pokemon based on their type. It uses a for loop to iterate over the Pokemon data and generate a list.
4. pkmn.html: This template extends the base.html template and is used to display information about a specific Pokemon. It shows the Pokemon's name, image, and description. It also includes a form for modifying or deleting the Pokemon.
5. register.html: This template extends the base.html template and is used to register a new Pokemon. It includes a form with fields for selecting the Pokemon type, entering the name, image URL, and description.
6. update.html: This template extends the base.html template and is used to update an existing Pokemon's information. It is similar to the register.html template but includes a hidden input field for the Pokemon ID.

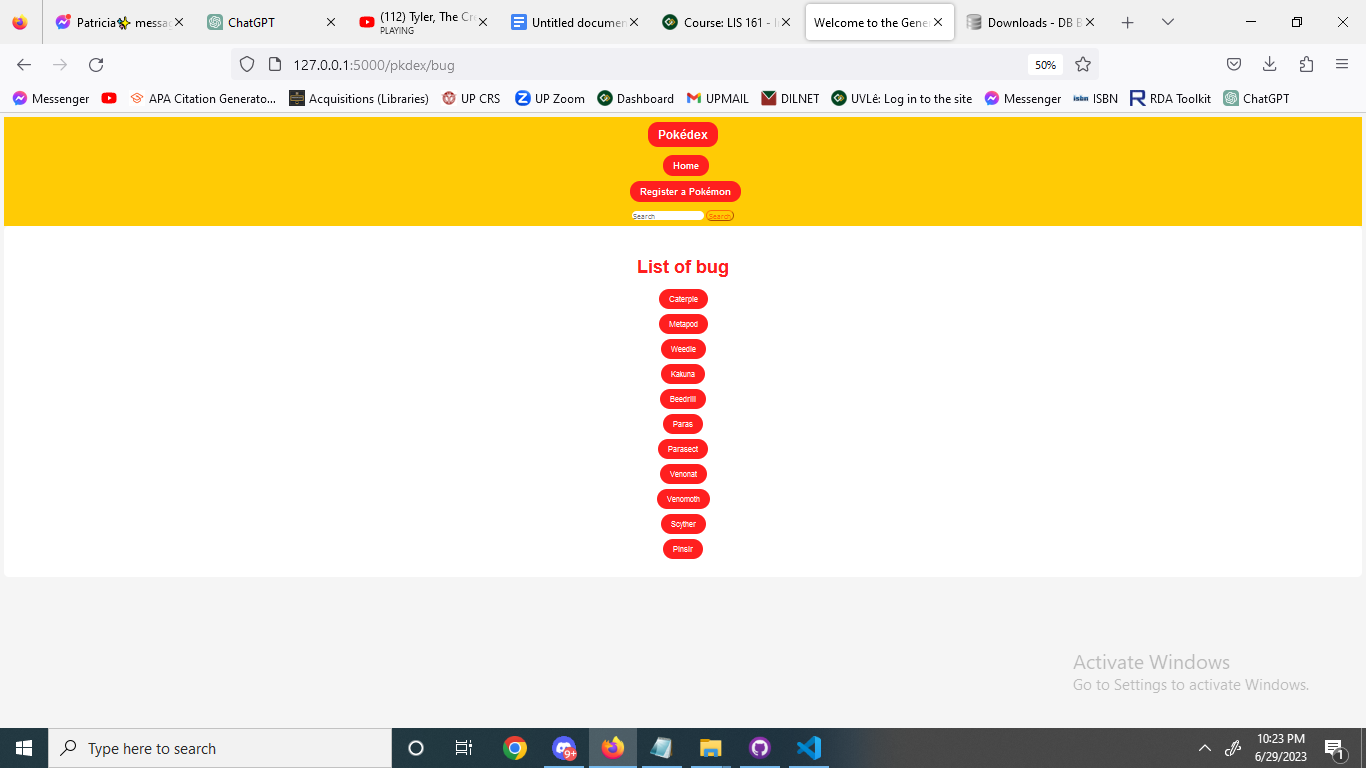
Note: We utilized AI, specifically ChatGPT to help us design the html files and we modified them so that it would be logical and fit the theme of Pokemon and give you a feel of what a digital Pokedex might look like.

**DEMO: showcase of all the features of the webpage**

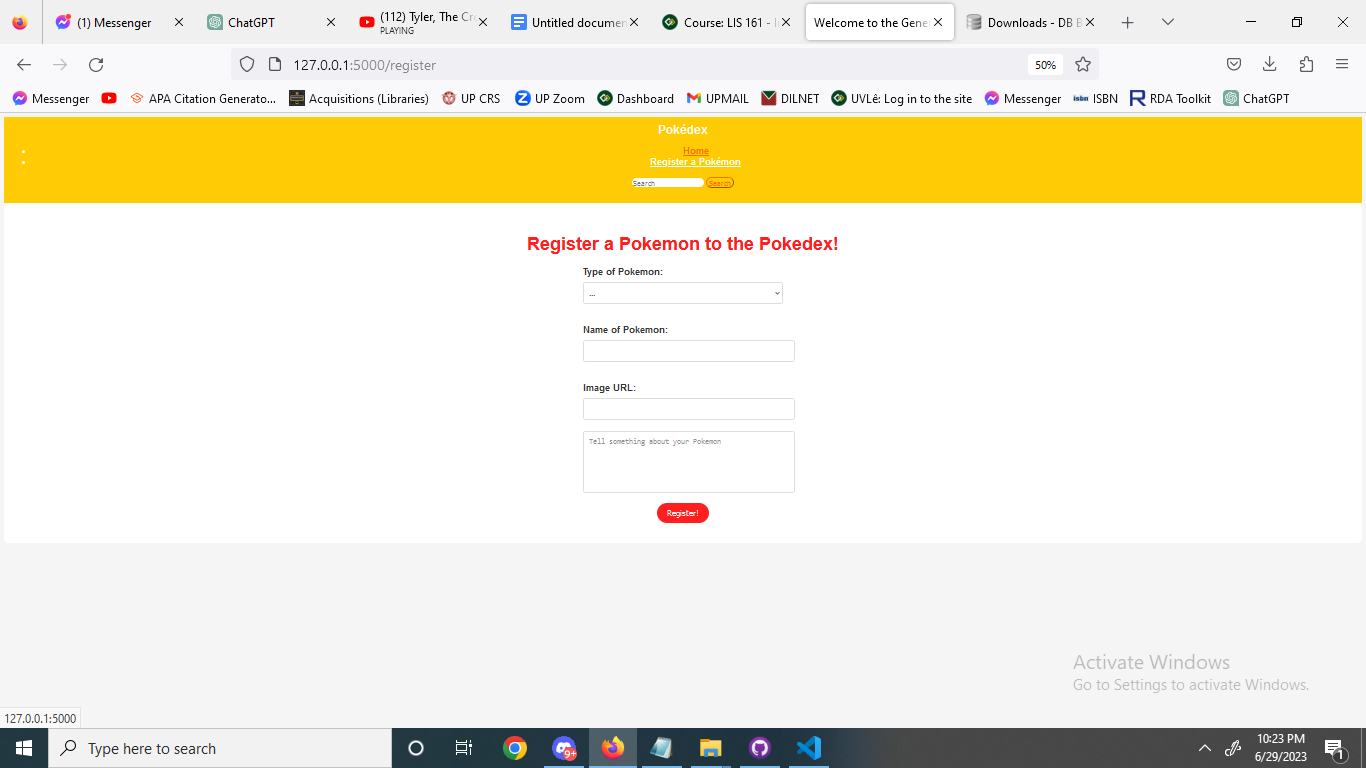
1. The homepage/index page of the application, showcasing the database which could be categorized by the different pokemon types. From this page you can also access the register feature, a decorable search bar that does not work but only serves as design, and a home button. We were also able to assign colors to the different pokemon types in a logical sense (water = blue) for ease of identification and navigation of the user.

****

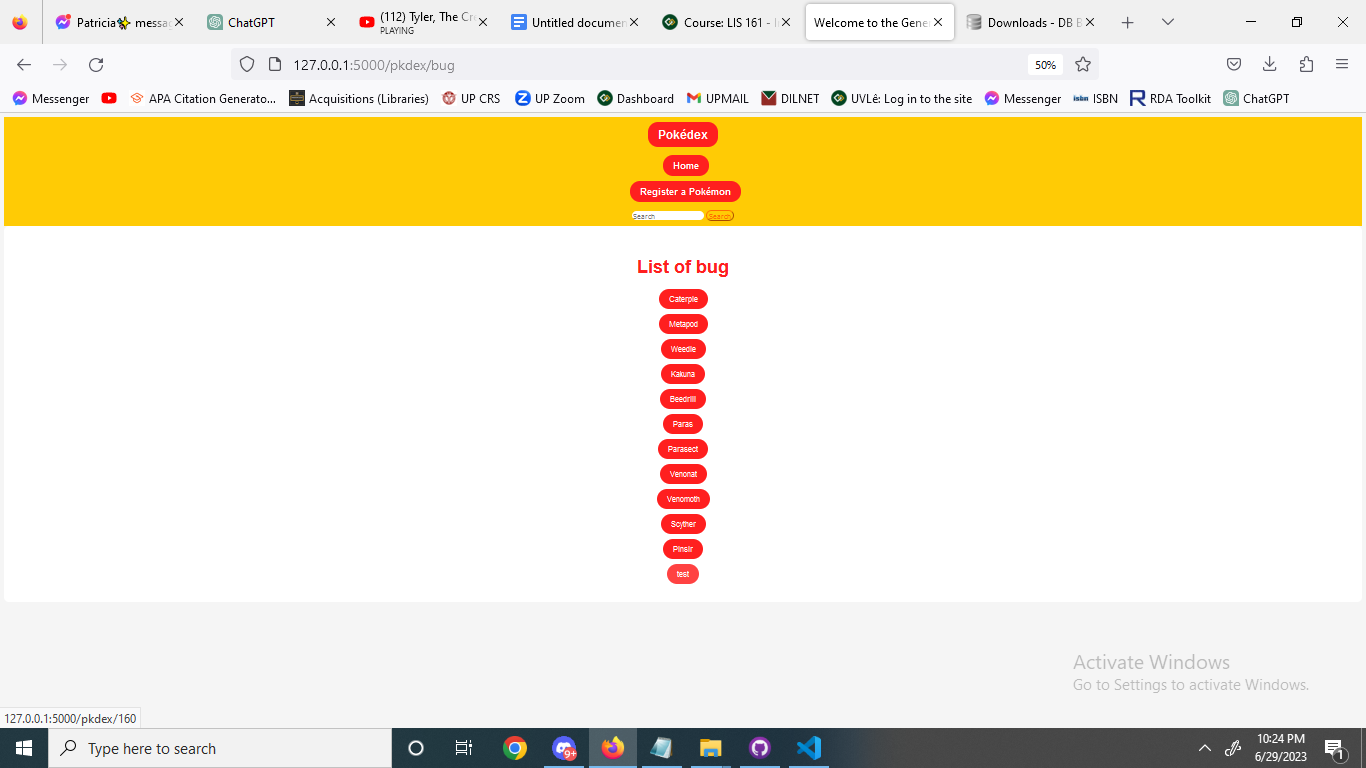
1. Upon clicking the pokemon type “Bug” you will be redirected to the pokedex.html file and will be presented with all the similar type Pokemon listed in the database.



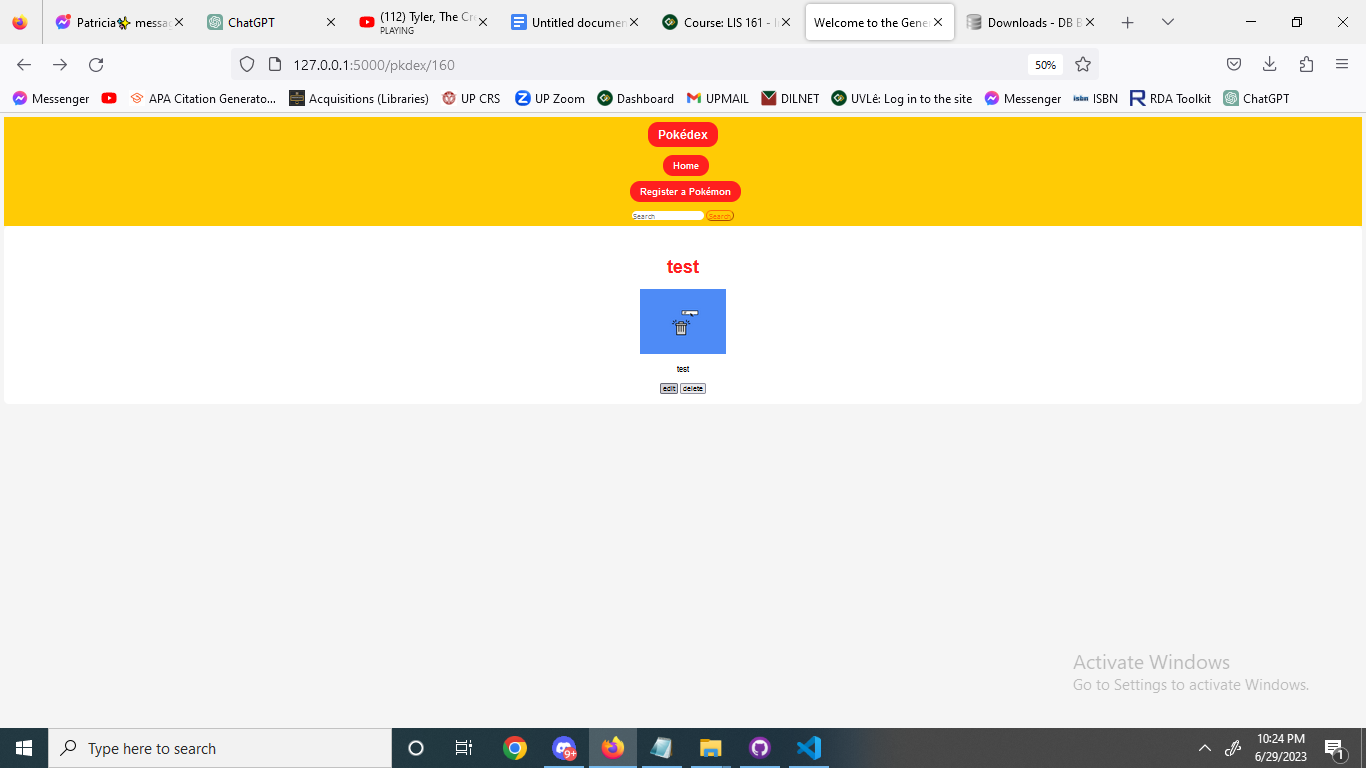
1. When clicking the register a pokemon button located at the navigation bar, you will then be redirected to this page which is the register.html file and will be prompted with fields necessary to be filled out. For this example, we will input the name of the pokemon as “test,” input its type as bug, field a random image url, and write “test” in the description.



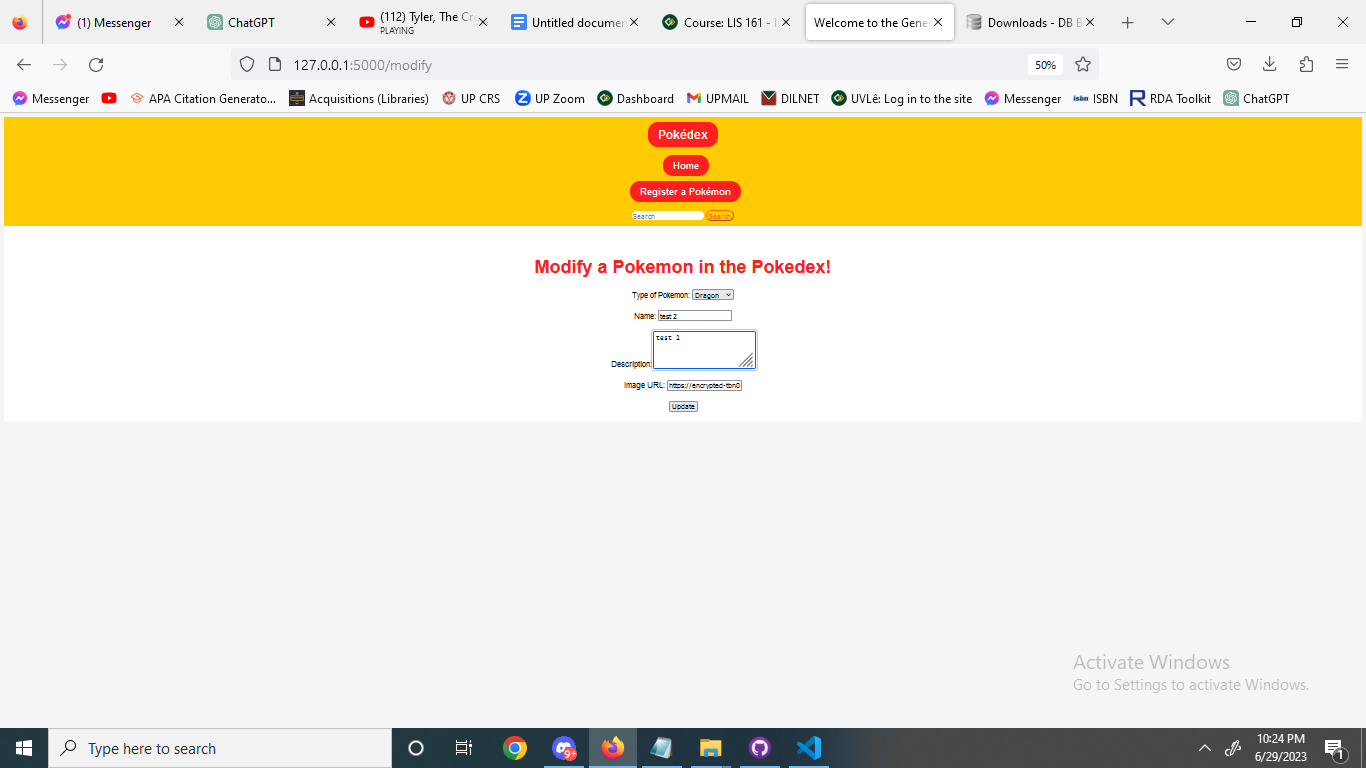
1. After inputting the registered pokemon, in this case “test,” we will now navigate back to the list of bug type pokemon and the most recent pokemon registered will appear at the bottom, as seen in the image



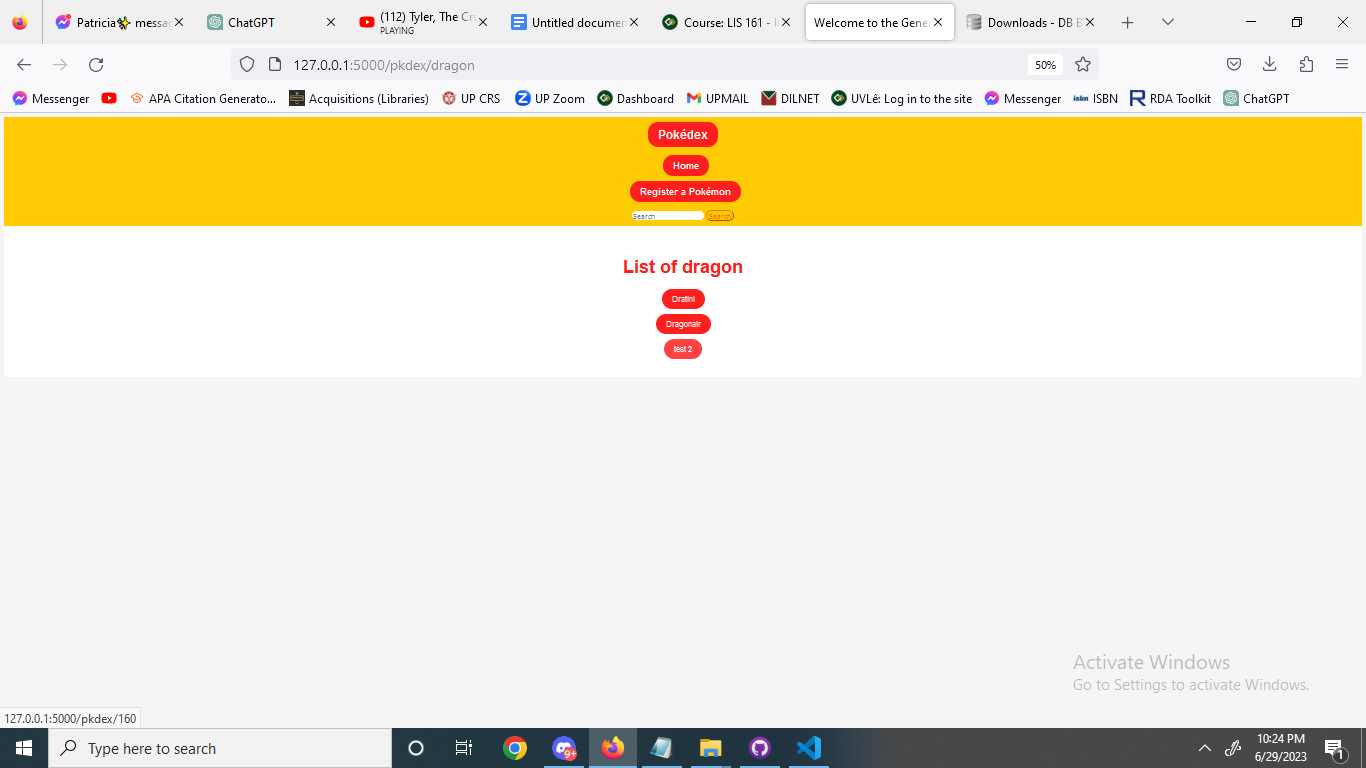
1. Now if we click the pokemon “test” you will then be redirected to the pkdex.html file and be prompted with all the data registered in the database, aptly named “pokemon.db” and the table “Pokemon” as shown in the image.



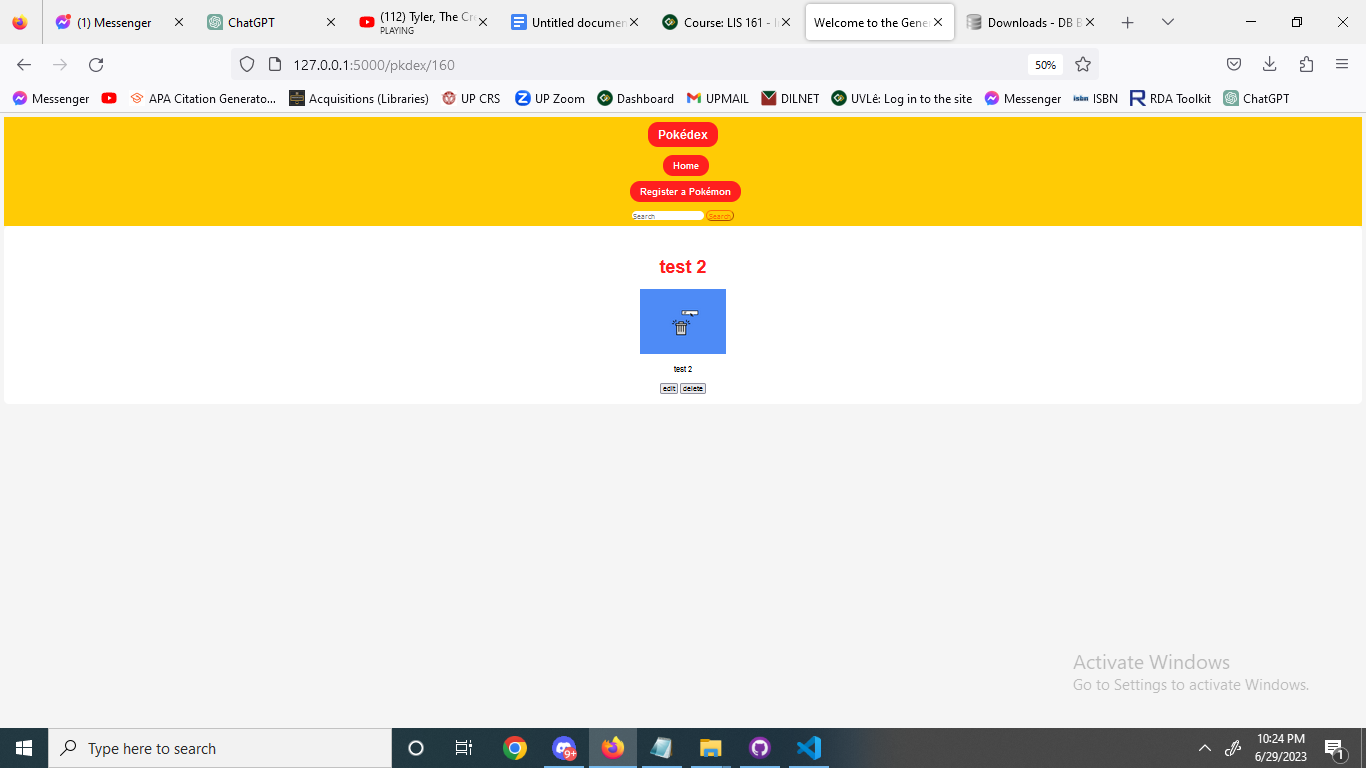
1. Now that we have “test” registered in the database, let’s say we want to modify its data fields, we want to make him a dragon type, rename it as “test 2”, have the same image, but change the description as “test 2,” we can click on the edit tab and fill the following fields up



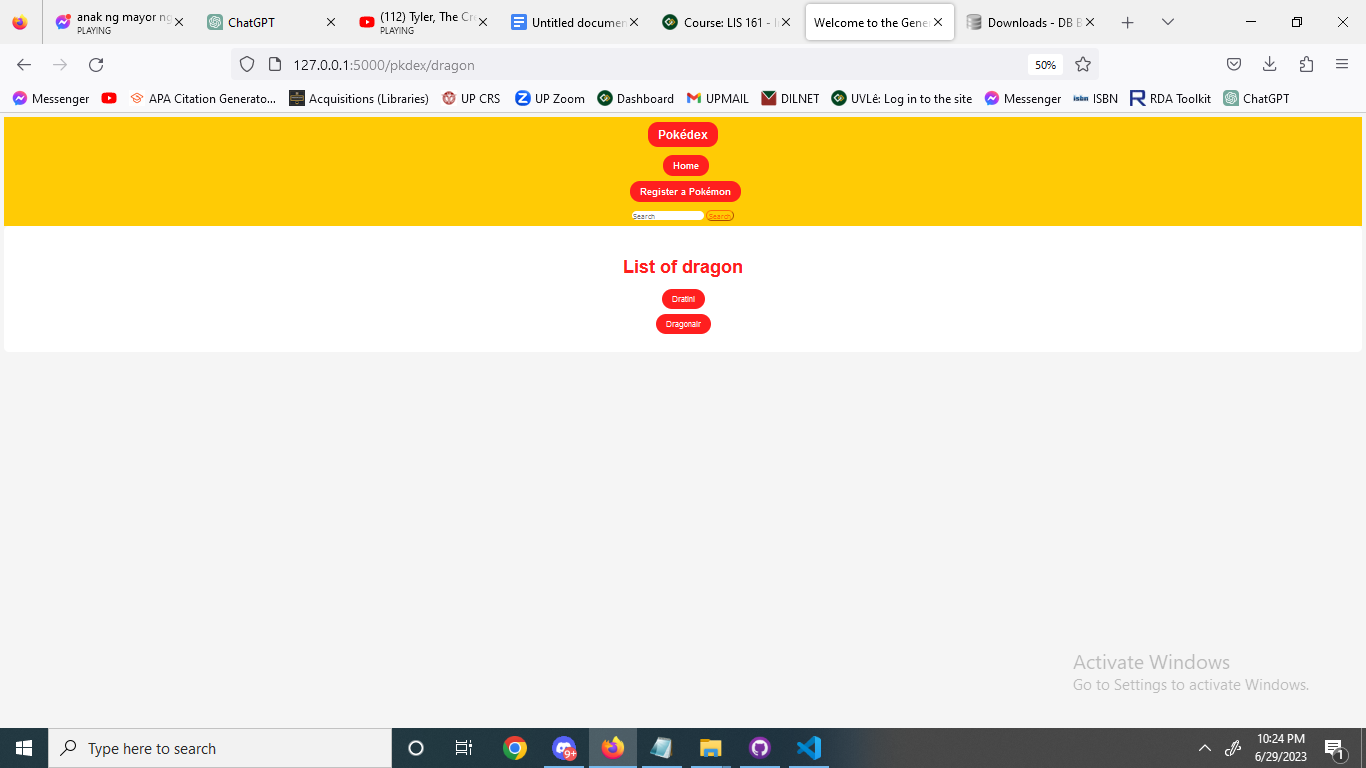
1. After inputting these and clicking the update button, we will be redirected back to the home page, we will then navigate to the dragon type page to see if our changes have reflected. We can confirm that the change has happened as we can now see a new button and Pokemon named “test 2” in the webpage, again, located at the bottom.



1. Now if we click “test 2” we will now be redirected to the pkdex.html file again and now we can see that the modification was successful. Now let’s click delete to see if it will indeed remove the record from the database.



1. If we navigate back to dragon type, we will then see that the delete did work and the Pokemon named “test 2” is now gone!



That concludes the demo and showcases all the features of the webpage.