# CSE 154: Web Programming — Autumn 2017

## Homework Assignment 7: Pokedex V2

**Due Date:** Monday, December 4th, 11pm

This assignment is about using PHP together with SQL to create, modify, and query information in a database.

## Overview

You are provided the following files:

- `main.html` - the main page of the application, which lets a user choose to start a game or trade Pokemon with another user and the pokedex/game view of the application, which lets a user choose a Pokemon to play with and then play a Pokemon card game with another player.
- `main.min.js` and `lib.min.js` - the JS for `main.html`
- `styles.css` - the styles for `main.html`

You will turn in the following files:

- `setup.sql` - a small SQL file that sets up your personal Pokedex table.
- `getcreds.php` - a web service for retrieving your player credentials (PID and token).
- `select.php` - a web service for retrieving the Pokemon from your Pokedex table.
- `insert.php` - a web service for adding a Pokemon to your Pokedex table.
- `update.php` - a web service for naming a Pokemon in your Pokedex.
- `delete.php` - a web service for removing a Pokemon from your Pokedex table.
- `trade.php` - a web service for updating your Pokedex list after a Pokemon "trade".
- `common.php` - shared PHP functions for your other PHP files.

## Behavior

### Creating Your Own Database - `setup.sql`

Before starting the PHP files, you will need to set up your own SQL database. In Cloud 9, you can do so by starting the mysql terminal and entering the following commands:

```
CREATE DATABASE hw7;
use hw7;
```

This is the database you'll be querying into throughout the assignment. The provided front-end code will use your web services and the data in your database to keep track of which Pokemon you have caught. Storing your Pokemon in a database (instead of in a JS array as suggested in HW5) allows us to maintain the state after refreshing or exiting the web page.

Write a SQL file that creates a table called "Pokedex" to store your collected Pokemon. This table should have three columns: "name" for each Pokemon's name which also serves as the table's `PRIMARY KEY` (e.g., "bulbasaur"), "nickname" for each Pokemon's nickname (e.g., "Bulby"), and "datefound" for the date and time you collected the Pokemon. The name and nickname columns should have `VARCHAR` data types and allow

string lengths of 30 characters (you can allow longer if you wish).

We can represent date and time in mysql using the `DATETIME` data type. This type represents a date and time in the format `YYYY-MM-DD HH:MI:SS` (e.g., 2017-05-15 13:54:00 to represent 1:54 PM on May 15th, 2017).

Your database name (hw7), table name (Pokedex), column names (name, nickname, datefound) must match exactly those here in the spec. Your columns must match exactly those described here in the spec.

Note the following SQL commands that will probably prove useful:

```
SOURCE setup.sql;                -- runs your setup.sql file
SHOW databases;                  -- lists databases in your mysql
USE hw7;                         -- tells mysql to use your hw7 database
SHOW tables;                     -- list tables in your currently active (hw7) database
DESCRIBE <tablename>;            -- gives information about the columns of a table
DROP TABLE <tablename>;          -- careful with this one, it deletes a table entirely
```

## Fetching Player Credentials - `getcreds.php`

This PHP file returns the user's player ID (PID) and token. For this assignment, your PID will be your UW netid. These PID and token values will be used by the front end code and the games webservice for verifying that players are who they say they are when they play moves in battle mode, and trade with one another.

You will need to generate your token to play games and trade with other students on our server. To do so, visit `https://webster.cs.washington.edu/pokedex-2/17au/uwnetid/generate-token.php`. The PID and token values displayed should be carefully copy/pasted in your `getcreds.php` file.

In this PHP file, you should print a plain text response (set the Content-Type in the header) with the body containing your PID followed by your token, each on their own line. Note that there are no query parameters for this file, so you print these values whenever the web service is called. Below is an example response for Kyle's credentials:

```
kthayer
poketoken_123456789.987654321
```

## Fetching Pokedex Data - `select.php`

`select.php` should output a JSON response of all Pokemon you have found (your Pokedex table), including the name, nickname, and found date/time for each Pokemon. Below is an example response:

```
{
  "pokemon": [
     { "name" : "bulbasaur",
       "nickname" : "Bulby",
       "datefound" : "2017-05-15 13:54:00" },
     { "name" : "charmander",
       "nickname" : "Charmy",
       "datefound" : "2017-05-16 08:45:10" },
       ... ]
}
```

This PHP web service does not take any query parameters (ignore any parameters passed).

## Adding a Pokemon to your Pokedex - `insert.php`

Query Parameters (`POST`):

- `name` - name of Pokemon to add
- `nickname` (optional) - nickname of added Pokemon

`insert.php` adds a Pokemon to your Pokedex table, given a required `name` parameter. The `name` should be added to your Pokedex in all-lowercase (for example, `name=BulbaSAUR` should be saved as `bulbasaur` in the Pokedex table).

If passed a `nickname` parameter, this nickname should also be added with the Pokemon (don't modify the anything to upper or lower case for the nickname, just store it as it was given). Otherwise, the nickname for the Pokemon in your Pokedex table should be set to the Pokemon's name in all uppercase (e.g., `BULBASAUR` for `name=BulbaSAUR`). You should also make sure to include the date/time you added the Pokemon. In PHP, you can get the current date-time in the format for the previously-described SQL `DATETIME` data type using the following code:

```
date_default_timezone_set('America/Los_Angeles');
$time = date('y-m-d H:i:s');
```

You should add the result `$time` variable to add to your `datefound` table column.

Upon success, you should output a JSON result in the format:

```
{ "success" : "Success! <name> added to your Pokedex!" }
```

If the Pokemon is already in the Pokedex (as determined by a duplicate name field), you should print a message with a 400 error header in the JSON format:

```
{ "error" : "Error: Pokemon <name> already found." }
```

where you should not change anything in your Pokedex as a result. For both success and error cases, `<name>` should be replaced with the value of the passed `name` (maintaining letter-casing).

## Removing a Pokemon from your Pokedex - `delete.php`

Query Parameters (`POST`):

- `name` - name of Pokemon to remove, or
- `mode=removeall` - removes all Pokemon from your Pokedex

If passed `name`, `delete.php` removes the Pokemon with the given name (case-insenstive) from your Pokedex. For example, if you have a Charmander in your Pokedex table and a request to `delete.php` with `name` passed as `charMANDER` is made, your Charmander should be removed from your table.

Upon success in this case, you should print a JSON result in the format:

```
{ "success" : "Success! <name> removed from your Pokedex!" }
```

If passed a Pokemon name that is not in your Pokedex, you should print a message with a 400 error header in the JSON format:

```
{ "error" : "Error: Pokemon <name> not found in your Pokedex." }
```

Your table should then not change as a result.

For both success and error cases, <name> should be replaced with the value of the passed name (maintaining letter-casing).

Otherwise if passed mode=removeall, all Pokemon should be removed from your Pokedex table.
Upon success in this case, you should print a JSON result in the format:

```
{ "success" : "Success! All Pokemon removed from your Pokedex!" }
```

If passed a mode other than removeall, you should print a message with a 400 error header in the format:

```
{ "error" : "Error: Unknown mode <mode>." }
```

where mode is replaced with whatever value the user passed for this query parameter.

### Trading Pokemon - trade.php

Query Parameters (POST):

- mypokemon - name of Pokemon to give up in trade
- theirpokemon - name of Pokemon to receive in trade

trade.php takes a Pokemon to remove from your Pokedex mypokemon (case-insensitive) and a Pokemon to add to your Pokedex theirpokemon.

Upon success, you should print a JSON result in the format:

```
{ "success" : "Success! You have traded your <mypokemon> for <theirpokemon>!" }
```

If your you do not have the passed mypokemon in your Pokedex table, you should print a 400 error header with a message in the same format as the error message specified in delete.php (using mypokemon for the Pokemon's name).

Otherwise, if you already have the passed theirpokemon in your Pokedex, you should print a 400 error header with a message in the JSON format:

```
{ "error" : "Error: You have already found <theirpokemon>." }
```

If either error occurs, your table should not be changed as a result. For any case, <mypokemon> and <theirpokemon> should be replaced with the respective query parameter values.

### Renaming a Pokemon in your Pokedex - update.php

Query Parameters (POST):

- name - name of Pokemon to rename
- nickname (optional) - new nickname to give to Pokemon

update.php updates a Pokemon in your Pokedex table with the given name (case-insensitive) parameter to have the given nickname (overwriting any previous nicknames).

Upon success, you should print a JSON result in the format:

```
{ "success" : "Success! Your <name> is now named <nickname>!" }
```

As in the previous files, `name` and `nickname` should be printed in the same format as the respective query parameters.

If you do not have the Pokemon with the passed `name` in your Pokedex, you should output the error behavior as in the same case for `delete.php`. If missing the `nickname` query parameter, the Pokemon's nickname should be replace with the UPPERCASE version of the Pokemon's name (similar to the case in `insert.php`). So for example, if passed `name=bulbasSAUR` (given you have a Bulbasaur in the table) and no `nickname` parameter is given, any previous nickname should be replaced with `BULBASAUR`. Your success message should then use `BULBASAUR` as the format for `<nickname>`.

#### common.php

You should factor any shared code into `common.php` and turn it in with the rest of your PHP files. Recall that you can use `include('common.php')` at the top of a PHP file to include all functions that are found in a file called `common.php` (requiring it is in the same directory as the file including it).
For any PHP web service with `GET` or `POST` parameters, if the user does not provide a required parameter, a 400 error message should be output in the JSON format:

```
{ "error" : "Missing <parametername> parameter"}
```

if only one required parameter is missing, and

```
{ "error" : "Missing <parameter1> and <parameter2> parameter"}
```

if multiple parameters are required and missing. In the case that one of a number of parameters should be provided, and none is, the error message should be of the form:

```
{ "error" : "Missing <parameter1> or <parameter2> parameter"}
```

These error responses should take precedence over any other error for each web service.

## Development Strategy

### SQL

This homework should give you a lot of experience using the `mysql` program to keep track of what changes are being made to your database.

- Run basic versions of your queries from the `mysql` terminal before putting them into your PHP.
- Use `try/catch(PDOException $pdoex)` to trap SQL exceptions in your PHP code, and print them for debugging.

### PHP

The provided front end for this homework is NOT a good testing program. It assumes that your code works, and makes many calls against your code in quick succession. We STRONGLY encourage you to call your PHP functions over the web before trying to use your code in concert with the provided front end.

For `GET` requests (`getcreds.php` and `select.php`) the easiest thing to do is simply use a browser to visit the URL and pass the query params.

For the other PHP files that you implement as `POST` requests, you'll need to do something a little bit more complicated. This is because it's harder to simulate `POST` requests than `GET` requests, but you have some options:

- Make a dummy HTML page that lets you write JS fetch commands for POSTS, or use the JS console.

- Make a dummy HTML page with a form that submits to your PHP program.
- Use a program like Postman https://www.getpostman.com/ to craft `POST` requests against your API. (Note: you need Postman Interceptor if you are developing on Cloud9 – this lets Postman use your browser login information to authenticate your request to Cloud9).
- One other way is to test with GET, and change to `POST` after you are satisfied that it works. However, you should still test that the `POST` works before you turn your homework in, and for this reason, we encourage you to use another testing strategy to get into the flow of actually testing `POST`s.

### General

- Get your database setup, implement `setup.sql` and practice making some database SELECT, INSERT, UPDATE, DELETE queries from the `mysql` terminal
- Implement `getcreds.php` to get going on the PHP part of the assignment.
- Be on the lookout for common code to factor into `common.php`
- Implement `select.php` using data that you have manually inserted into the DB from the `mysql` terminal
- Implement `insert.php`, and verify that it works first in the database, and then with `select.php`
- Implement `update.php` and `delete.php`
- Implement `trade.php`
- Review all of your files to make sure you've factored out any shared code into `common.php`

## Implementation and Grading

For full credit, your SQL and PHP code should follow the rules listed in the Style Guide on the course web site and follow a similar format to that of lecture examples. Remember to comment each file you create with a brief overview of what the file does, and include comments for each function you define.

Your PHP code should not cause errors or warnings. Do not use the `global` keyword. Use good indentation/spacing, and avoid long lines over 100 characters. Avoid redundant code, and use parameters and return values properly. Capture common operations as functions to keep code size and complexity from growing.

Additionally, for this assignment, your PHP code must pass a PHP strict standards check. To check for this, you should execute the following line of code at the beginning of every PHP file that you write:
`error_reporting(E_ALL);`

Note that it might not make sense for this line of code to be physically present in every PHP file.

Do not place your solution on a public web site. Submit your own work and follow the course misconduct policy.