# CSE 154: Web Programming — Spring 2018

## Homework Assignment 7: Pokedex V2

**Due Date:** Wednesday, May 30, 11pm

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

## Overview

This is the first assignment where the primary focus is not a user interface but only the web service which will connect to a database to retrieve and modify data.

## Learning Objectives

- Continue to practice all of the learning objectives from Homeworks 1-6, including:
  - Carefully reading a specification.
  - Reducing redundancy in your code while producing expected output.
  - Producing quality readable and maintainable code with unobtrusive PHP.
  - Clearly documenting your code as specified in the CSE 154 Code Quality Guide.
- Building an API that responds to GET and POST requests using the PHP language.
- Using the PHP language to read information from a database.
- Using the PHP language to write information to a database.

## Files Provided

To begin with you will not be provided any Graphical User Interface files for testing. On Monday we will release client-side files for a small web app you can (optionally) use to test your API with. Details about the provided 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`

## Files to Submit

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 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.

## Web Service 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 your web service will be using to keep track of which Pokemon you have caught. Storing your Pokemon in a database (instead of in a DOM element as you did in HW5) allows us to maintain the state after refreshing or exiting the web page.

Write a SQL file `setup.sql` that creates a table called "Pokedex" to store your collected Pokemon. This file should meet the following requirements:

- The "Pokedex" 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")
  - "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).
- To represent the date and time, use the `DATETIME` data type. In MySQL, this type represents a date and time in the format `YYYY-MM-DD HH:MI:SS` (e.g., 2018-05-15 13:54:00 to represent 1:54 PM on May 15th, 2018).
- Your database name (hw7), table name (Pokedex), column names (name, nickname, datefound) **must match exactly those 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`

**Request Format:** `getcreds.php`
**Request Type:** `GET`
**Returned Data Format:** `plain text`
**Description:** 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-2/18sp/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 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.

**Example Request:** `getcreds.php`
**Example Output** (`bricker` is the example PID):

```
bricker
poketoken_123456789.987654321
```

## Fetching Pokedex Data - `select.php`

**Request Type:** `GET`
**Returned Data Format:** `JSON`
**Description:** `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. This PHP web service does not take any query parameters (ignore any parameters passed).
**Example Request Output:**

```
{
  "pokemon": [
     { "name" : "bulbasaur",
       "nickname" : "Bulby",
       "datefound" : "2018-05-15 13:54:00" },
     { "name" : "charmander",
       "nickname" : "Charmy",
       "datefound" : "2018-05-16 08:45:10" },
       ... ]
}
```

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

**Request Type:** `POST`
**Query Parameters:**

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

**Returned Data Format:** `JSON`
**Description:** `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.

**Expected Output Formats:**
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." }
```

Nothing should change anything in your Pokedex if there is an error due to a name collision. However, in 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`

**Request Type:** `POST`
**Query Parameters:**

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

**Returned Data Format:** `JSON`
**Description:**

- 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.
- If passed `mode=removeall`, all Pokemon should be removed from your Pokedex table.

**Expected Output Formats:**
Upon success in using the `name` parameter, 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 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>` in the message should be replaced with the value of the passed `name` (maintaining letter-casing).

If `mode` is passed as a POST parameter with the value `removeall`, and all Pokemon are successfully removed from your Pokedex table, 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`

**Request Type:** `POST`
**Query Parameters:**

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

**Returned Data Format:** `JSON`
**Description:** `trade.php` takes a Pokemon to remove from your Pokedex `mypokemon` (case-insensitive) and a Pokemon to add to your Pokedex `theirpokemon`.

When adding `theirpokemon` to your Pokedex, the Pokemon name should be in all lower case and the Pokemon should have the default nickname format (i.e. the name in all UPPERCASE).

**Expected Output Formats:**
Upon success, you should print a JSON result in the format:

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

If you do not have the passed `mypokemon` in your Pokedex table, you should print a 400 error header with the following message in JSON format:

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

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 (maintaining letter-casing).

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

**Request Type:** POST
**Query Parameters:**

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

**Returned Data Format:** JSON
**Description:** 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)
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.

**Expected Output Formats:**
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 you are not passed a nickname, your success message should then use the uppercase version of the pokemon's name for the nickname (i.e. 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 so STRONGLY encourage you not to use this as a testing program that we are not even going to release it until day 3 of this assignment.
Instead we 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.
- As you work, 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 Code Quality Guide on the course web site and follow a similar format to that of lecture examples.
- Your SQL file should have a header comment with your name and section at a minimum. You may choose to comment your SQL file more than that but it is not required.
- Your PHP files should have adequate documentation. The top of the file should have a descriptive header describing the assignment. Complex sections of code should be documented. Your PHP variables and functions should be documented in a style like JSDOC documenting the method's description, parameters and return values.
- Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and method names and follow the naming conventions as outlined in the Code Quality Guide. Lines of code should be fewer than 100 characters long.
- You must use the ensure that a users can not inject malicious SQL into your database by `PDO` `prepare/exec` set of statements for insert, delete and update queries.
- Your PHP code should not cause errors or warnings. Add `error_reporting(E_ALL);` to the top of your `.php` file so errors are thrown and not kept silent. Hint: If you set error handling in `common.php` it will be set in any file that includes `common.php`.
- Variables should be localized as much as possible.
- Do not use the `global` keyword.
- Utilize PHP functions for good readability. Capture common operations as functions to keep code size and complexity from growing.
- You should not be creating more than one database (PDO) object for any request. Consider factoring out your code that opens the database connection.
- Alternatively, you should not create a database (PDO) object when you do not need one.
- You should make an extra effort to minimize redundant code. Capture common operations as functions to keep code size and complexity from growing.

## Grading Rubric

This assignment will be out of 20 points. The key areas we will be looking at assess directly relate to the learning objectives, and your matching the specification for the external behavior as well as the internal correctness of your code. **NOTE:** While we can not guarantee the same distribution of points, past rubrics have been split with 60% of the points allocated to external correctness and the 40% for internal. Thus a **potential** rubric **might be** summarized as:

External Correctness (12 pts)

- Database setup
- Web Service
  - player credentials
  - getting pokedex data
  - adding pokedex data

  - removing pokedex data
  - trading pokedex data
  - renaming pokemon
  - error handling
- Service client (JavaScript)

Internal Correctness (8 pts)

- PHP
  - follows class code quality guidelines
  - avoids redundancy, uses function to encapsulate functionality
  - functions are well documented including parameters and return values
  - strict error reporting turned on
  - sets JSON_encode properly
- setup.sql works as expected.
- Otherwise good quality code - a catch all for things like indentation, good identifier names, long lines, large anonymous functions, etc.

## Academic Integrity

As with any CS homework assignment, you may not place your solution to a publicly-accessible web site, neither during nor after the school quarter is over. Doing so is considered a violation of our course academic integrity policy. As a reminder: The University of Washington has an entire page on Academic Misconduct on their Community Standards and Student Conduct Page. Please acquaint yourself with the University of Washington's resources on academic honesty, and in particular how academic misconduct will be reported (which has been changed for 2017).