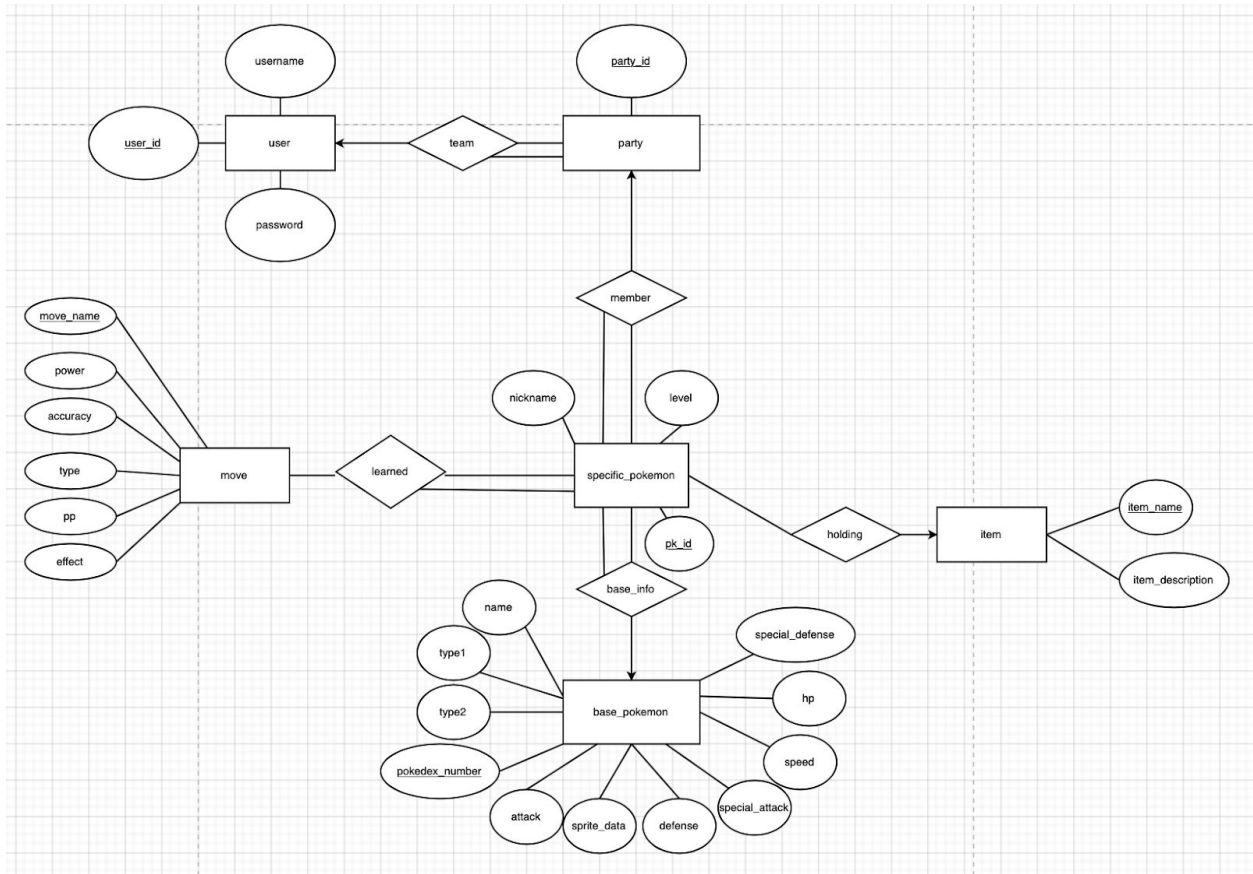


CS 4750 Databases: Final Project Report

Larry Cai (ljc2sh), Charles Fang (csf8mb), Eldon Luk (eb19rd), Jammie Wang (jw8kc)

Database Design

ER Diagram



Schema Statements

party(party_id)

user(user_id, username, password)

moves(move_name, power, accuracy, type, pp, effect)

item(item_name, item_description)

specific_pokemon(pk_id, level, nickname)

base_pokemon(pokedex_number, name, type1, type2, family, attack, defense, special_attack, special_defense, speed, hp, sprite_data)

team(party_id, user_id)

base_info(pk_id, pokedex_number)

holding(pk_id, item_name)

```
learned(move_name, pk_id)
member(pk_id, party_id)
```

Database Programming

We are hosting our database on the UVa CS server. During development, we used XAMPP's mysql. Due to the CS server's need for credentials, our instructions below describe how to set-up the local MYSQL server. Our React app will run locally.

Application Architecture

The frontend of our web application is built with ReactJS and TypeScript. AXIOS allows us to use asynchronous javascript and API calls to our backend. **Src\axios\api.tsx** contains our AXIOS requests. That file also contains an `apiURL` variable that represents the main endpoint and can either point to the CS servers or localhost.

Our backend server consists of PHP files located in **src\php**. `connectDB.php` contains the database credentials and `dbQuery.php` contains the functions that run the actual SQL statements. Additional PHP files such as `item.php`, `move.php`, or `pokemon.php` contain switch statements and serve as the actual endpoints that our AXIOS requests visit.

The SQL file used to import the database is called `database.sql` and is located at **src\php**. If you are having trouble importing the SQL file on MACs due to an error about requiring an update to MYSQL and your computer cannot find mysql, try running these commands.


```
sudo nano ~/.bash_profile

export
PATH=/opt/local/bin:/opt/local/sbin:/Applications/xampp/xamppfiles/bin:$PATH


sudo mysql_upgrade
```

Steps to run our application:

1. Download XAMPP 7.2.3

 XAMPP for **OS X** 7.2.34, 7.3.23, 7.4.11, 7.2.34, 7.3.23 & 7.4.11

Version	Checksum	Size
7.2.34 / PHP 7.2.34	What's Included? md5 sha1	Download (64 bit) 161 Mb

 XAMPP for **Windows** 7.2.34, 7.3.23 & 7.4.11

Version	Checksum	Size
7.2.34 / PHP 7.2.34	What's Included? md5 sha1	Download (64 bit) 153 Mb

2. Max: Open up the terminal, Windows: Open up the Command Prompt

```
larrycai — -bash — 80x24
Last login: Fri Nov 13 12:34:18 on console

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
larrys-mbp:~ larrycai$
```

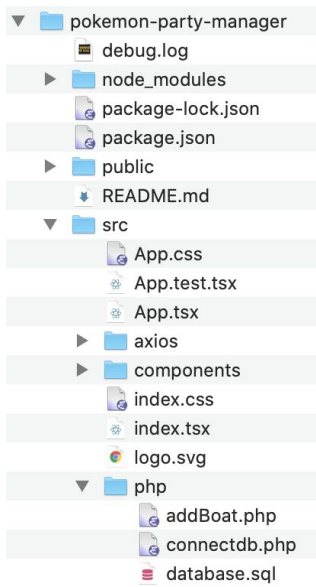
3. Run: git clone <https://github.com/fang-charles/pokemon-party-manager>

Make sure it is in directory htdocs/cs4750/pokemon-party-manager and to pull from the “development” branch!

```
[larrys-mbp:~ larrycai$ git clone https://github.com/fang-charles/pokemon-party-manager
```

4. Path to database:

```
larrys-mbp:php larrycai$ pwd database.sql
/Applications/XAMPP/xamppfiles/htdocs/cs4750/pokemon-party-manager/src/php
```



5. Upload sql file to local database. Make sure XAMPP is installed, and Apache and MySQL are running.

Importing into the database "ebl9rd_c"

File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in `.[format].[compression]`. Example: `.sql.zip`

Browse your computer: no file selected (Max: 40MiB)

You may also drag and drop a file on any page.

Character set of the file:

6. Find the path to connectdb.php:

```
larrys-mbp:php larrycai$ pwd connectdb.php
/Applications/XAMPP/xamppfiles/htdocs/cs4750/pokemon-party-manager/src/php
```

7. Change credentials in connectdb.php to your local database credentials. Again, make sure XAMPP is installed, and Apache and MySQL are running.

```
$username =
$password =
$host = 'loc
$dbname = 'e
```

The default credentials can be used to connect to the CS server database:

```
$username = 'jw8kc';
$password = 'Helloworld1!';
$host = 'usersrv01.cs.virginia.edu';
$dbname = 'jw8kc';
```

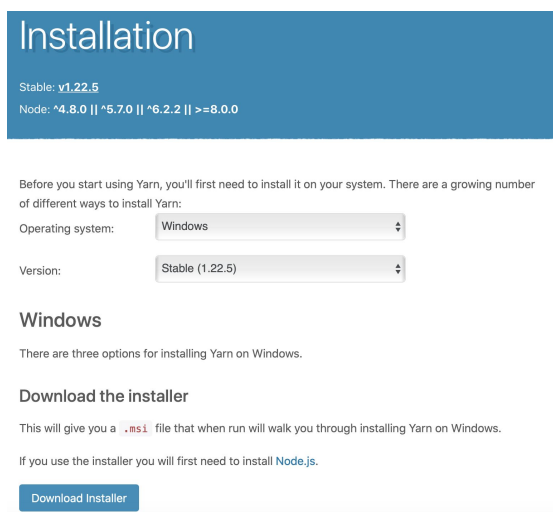
8. Run: cd pokemon-party-manager

```
larrys-mbp:~ larrycai$ cd pokemon-party-manager
```

9. Install yarn

9a. Window install yarn: follow instructions at

<https://classic.yarnpkg.com/en/docs/install/#windows-stable>



9b. Mac install yarn: brew install yarn

```
[larrys-mbp:pokemon-party-manager larrycai$ brew install yarn
```

10. Run: install npm

```
[larrys-mbp:pokemon-party-manager larrycai$ npm install
```

11. Run: yarn. This will download all needed packages and dependencies.

```
[larrys-mbp:pokemon-party-manager larrycai$ yarn
```

12. Run: yarn start. This will now open the application in your default web browser.

```
[larrys-mbp:pokemon-party-manager larrycai$ sudo yarn start
yarn run v1.22.10
$ react-scripts start
```

Advanced SQL Commands

Advanced SQL commands incorporated in our application were constraint checks and stored procedures. Constraint checks are used in many of our tables to ensure that the data contained in our database matches the constraints we have set up for our application. An example of constraint check from milestone 2 was:

```
CREATE TABLE specific_pokemon(
pk_id int,
level int,
nickname varchar(255),
PRIMARY KEY (pk_id),
FOREIGN KEY (pk_id) REFERENCES member(pk_id)
ON DELETE CASCADE,
CHECK (level >= 1 && level <= 100 )
);
```

This constraint checks to make sure that each pokemon added to a user's party has a level between 1 and less than 100. This makes sense as a Pokemon cannot be level 0 or greater than 100, per defined by the game itself.

We also wrote many stored procedures to simplify the many steps that our application required. A good example of this is:

```
DELIMITER $$
CREATE PROCEDURE addPokemon(
```

```

    IN pokedex_number INT,
    IN level INT,
    IN nickname varchar(255),
    IN party_id int,
OUT pk_id int
)
BEGIN
INSERT INTO member (party_id) VALUES (party_id);
SELECT @@IDENTITY INTO pk_id;
INSERT INTO specific_pokemon (pk_id, level, nickname) VALUES (pk_id, level,
nickname);
INSERT INTO base_info (pk_id, pokedex_number) VALUES (pk_id, pokedex_number);
SELECT pk_id AS `pk_id`;
END
$$
DELIMITER ;

```

The addPokemon stored procedure is called when a user wants to add a pokemon into their party. The user will need to specify the pokedex number for the pokemon, its level, and nickname. Additionally, the user must provide the specific party ID they are adding the pokemon to. This procedure is called in our AddPokemon.tsx component in the front-end. The component contains text fields that set the level, base pokemon, and nickname as well as a save button.

Due to the foreign key dependencies in our table, adding a new pokemon requires a very specific ordering of INSERT statements. First, a new row containing the party ID is added to the member relationship table that links parties and specific pokemon. The auto-incrementing primary key of member (pk_id) is generated and stored as a variable.

With this pk_id (the unique identifier for a specific pokemon), we can insert values into the specific_pokemon table, which describes the level and nickname of the specific pokemon, and the base_info relationship table, which provides the connection to base_pokemon statistics that are identical for all instances of a pokemon.

The final select statement allows the API call to retrieve the pk_id of the recently generated pokemon. This stored procedure simplifies all of these insertions into one SQL call from our PHP API. Many other similar stored procedures are used across our application to provide this ease of usage. For example, generateParty(IN user_id, OUT party_id) and setMoves(IN pk_id, IN move1, IN move2, IN move3, IN move4) both simplify our API calls by abstracting away the individual steps.

Export Data

As part of the requirements, we added an export data feature. When viewing a pokemon card, there is an option to download the Pokemon information as a JSON object. This JSON object describes all the attributes about the specific pokemon downloaded.

Database Security

Database Level

At the database level, end users of the application do not have accounts for the database at all and thus, no permissions. We have implemented access control for developers by creating sub-accounts with certain permissions. Developers have permission to:

- CREATE new tables and databases
- INSERT new rows into tables
- SELECT items throughout the database
- UPDATE table rows

Developers do not have permission to:

- DROP tables or the database
- DELETE rows from tables

Developers do not have access to the ‘user’ table since it contains usernames and hashed passwords, which poses a huge security risk if it is accessible. However, they do have access to all other tables and all routines. The SQL commands used to limit and set developer privileges for both routines and tables are as follows:

Routines

```
GRANT CREATE ROUTINE ON jw8kc_.* TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.addPokemon TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.clearParty TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.clearUser TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.generateParty TO
'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getBasePokemonInfo TO
'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getParty TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getPokemonitem TO
'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getPokemonmoves TO
'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getSpecificPokemon TO
'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getUser TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.getUserID TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.loseItem TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.removePokemon TO
'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.setitem TO 'jw8kc_a'@'localhost';
GRANT ALTER ROUTINE, EXECUTE ON PROCEDURE jw8kc_.setmoves TO 'jw8kc_a'@'localhost';
```

Tables

```
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.base_info TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.base_pokemon TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.holding TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.item TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.learned TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.member TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.move TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.party TO 'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.specific_pokemon TO
'jw8kc_a'@'localhost';
GRANT CREATE, INSERT, SELECT, UPDATE ON jw8kc_.team TO 'jw8kc_a'@'localhost';
```

Since our database is hosted on the CS server, we were unable to reference entire databases by doing `jw8kc_.*`, hence why some of the commands are repetitively applied to multiple tables within the same database. All of the commands above were used for sub-account `jw8kc_a`. The usernames were respectively switched out for `jw8kc_b`, `jw8kc_c`, and `jw8kc_d` to give permissions to each sub-account.

Application Level

Passwords are hashed using a hash function before storing it in the 'user' table. Doing so prevents plaintext passwords from being stored in a database. Note that although the password is sent in plaintext over the network to the server, HTTPS (the application layer protocol) is encrypted, which guards against attacks within the network. The password hashing at the server is shown with the below code:

```
global $db;

$hash = hash('md5', $password);
$query = "INSERT INTO user (username, password) VALUES
(:username, :hash);";
$stmt = $db->prepare($query);
$stmt->bindValue(':username', $username);
$stmt->bindValue(':hash', $hash);
try {
    $stmt->execute();
} catch (Exception $exception) {
    $stmt->closeCursor();
    return false;
}
```

Some additional application level security measures we had intended on implementing include not allowing the user to access any part of the application without logging in. Currently, we have a user table in our database where each user has a unique ID that is automatically generated and incremented upon user creation. The hashed passwords are stored in this table.