Pokebuilder Project Report

Chase Jamieson

Vuochlang Chang

December 7, 2021

1 Introduction

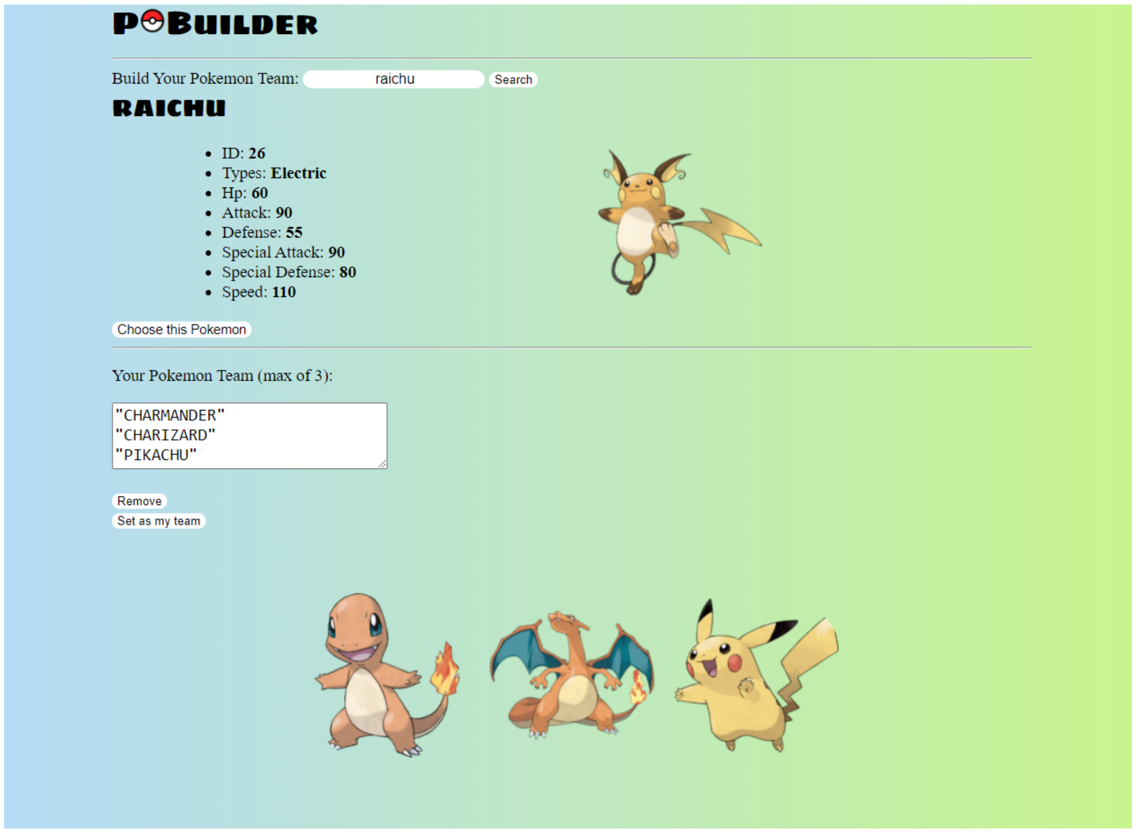
From our Project Proposal, we were tasked to create a Pokemon Team builder web app. Where users could search for a pokemon and add it to their team. Specificiations were to implement using a Mongo Database, a Javascript frontend, and a Node and Express backend.

2 Implementation

First We extracted data from PokeApi (An API interface containing Pokemon data) into a useable JSON of the form:

```
    {"id": 1,
    "name": "bulbasaur",
    "types": ["Grass", "Poison"],
    "hp": 45,
    "attack": 49,
    "defense": 49,
    "special-attack": 65,
    "special-defense": 65,
    "speed": 45,
    "source": "http://pokeapi.co/api/v2/pokemon/1/"},
```

Using this data we created an interface for searching for pokemon, adding it too your team, viewing their stats and types. Upon startup, all 898 pokemon data is added to the Mongo database, the user will be able to search and query for the pokemon they want.



RAICHU

- ID: **26**
- Types: **Electric**
- Hp: **60**
- Attack: **90**
- Defense: **55**
- Special Attack: **90**
- Special Defense: **80**
- Speed: **110**

Your Pokemon Team (max of 3):

```
"CHARMANDER"
"CHARIZARD"
"PIKACHU"
```

As seen on the page users can:

- Search for a pokemon in the database
- Add and remove pokemon from their team
- save their team for viewing.

3 MongoDB JSON

On starting the server, our JSON with all the Pokemon data is automatically imported into the mongo database. Additionally entering ctrl+c will be caught and drop the database.

```
async function startServer() {
  // Set the db and collection variables before starting the
     server.
  db = await MongoClient.connect(MONGO_URL);
  collection = db.collection('pokemon');
  console.log('Created database <pokebuilder> and collection <
     pokemon>');

  // added all 898 pokemon data
  let pokemonData = fs.readFileSync('pokemon.json');
  let pokemon = JSON.parse(pokemonData);
  await collection.insertMany(pokemon);
  console.log("Finished loaded data");

  // Now every route can safely use the db and collection objects
     .
  await app.listen(3000);
  console.log('Listening on port 3000');

  process.on('SIGINT', function() {
    console.log("Caught ctrl-C signal, clearing data on Mongo..")
        ;
    db.dropDatabase();
    console.log("Dropped database");
    db.close();
    process.exit();
  });
}
startServer();
```

4 GET/POST routes

We have 1 get route and 2 Post routes in our Server.js file. The GET route is for the search function, while the post routes are for adding and removing pokemon to your team. The code below demonstrates that:

```
app.get('/lookup/:word', findPokemon);

async function addToTeam(req, res) {
  const definition = req.body.definition;
  console.log("debug: requested to add ", definition.word, " to
     the team");

  myPokemonList.addPokemon(definition);
  myPokemonList.printPokemon();

  res.json({ success: true });
}
app.post('/set/', jsonParser, addToTeam);

async function removePokemon(req, res) {
  myPokemonList.remove();
  myPokemonList.printPokemon();
  res.json({ success: true });
}
app.post('/remove/', jsonParser, removePokemon);
```

5 Front-end ES6 Classes and Back-end Classes

On the Front-end we have two ES6 classes, one for each Pokemon you search, and one for the teams. We considered these necessarily to separate because the information stored for the individual Pokemon differs from the team you create. These classes are in our fetch.js file which is what we use on the client side to communicate with the server.

```
module.exports = class ServerPokemon {
   constructor() {
      this.pokemonList = [];
      this.index = 0;
   }

   printPokemon() {
      console.log("debug <class> printing..");
      for(var i = 0; i < this.pokemonList.length ; i++){
         console.log(this.pokemonList[i].word);
      }
```

```
    }

    addPokemon(pokemonData) {
        if (this.index > 2) return;
        this.pokemonList.push(pokemonData);
        this.index += 1;
        console.log("debug <class>: added <" + JSON.stringify(
            pokemonData.word));
    }

    remove() {
        this.pokemonList.pop();
        console.log("debug <class>: removed last pokemon");
    }

}
```

6 Lessons learned

1. When making the project we had several lessons learned on both the front end and backend. On the front end side, its very important to outline whether or not its necessary to create new classes to hand certain functions. We also gained experience using fetch as well as async/await functions to work with our backend. Another aspect which is less technical on the front end but very important is formatting the displays so the interface is useable on different resolutions. On the backend side, we learned how useful Monogdb can be to query and automatically import our data on server startup.

2. If someone was considering using our system and framework, I would tell them first clearly outline how they'd like to implement classes to handle certain pieces of data. I'd also say its important to work on the specific structural pieces of code before trying to make it look nice and presentable, functionality first.