Sistemes i Tecnologies Web

Gotta Catch 'Em All: Part 1 - Backend

Contents

1 Objectives

The main objectives of this exercise are:

- Consolidate the Javascript concepts seen in theory classes.
- Work with the modules NodeJS [1] offers.
- Implement the back-end side of a web application using Javascript and NodeJS.

2 Introduction

In this exercise we will develop a back-end application that will be used by a Vue front-end implemented on the second part of the project.

The front-end will display a simplified version of Pokemon Red's Route 1. The player will be able to walk it, enter the grass to find Pokemons, catch them, visualize them on the Pokedex, and save the game.

The back-end application will be implemented using *Express.js* [2]. For the front-end to be able to properly work, it must provide the following five endpoints:

- **`/initial_info`**: An endpoint that returns a dictionary with the initial info needed to start the game. If **`/save`** endpoint has been called before, it returns the dictionary stored in a .txt file; if not, it returns a dictionary with default values. The returned dictionary must have the following mandatory attributes:
  - **`x`**: A positive integer representing the player 'x' position on the map.
  - **`y`**: A positive integer representing the player 'y' position on the map.
  - **`direction`**: An string representing the direction the player is facing ('north', 'south', 'west' or 'east').
  - **`pokedex`**: A dictionary where the key represents a Pokemon identifier, and the value it's a boolean representing if it has been caught (true) or just seen (false).
- **`/enter_grass`**: An endpoint called when the player enters a patch of grass. It returns 400 if a wild encounter (A wild encounter is when a Pokemon appears) must trigger inside the grass, and 200 otherwise.
- **`/leave_grass`**: An endpoint called when the player exits the patch of grass. As we are outside the grass, a wild encounter must not trigger, thus it must provoke **`/enter_grass`** endpoint to return 200 instead of 400.
- **`/capture`**: An endpoint that returns 200 if a Pokemon has been caught, and 400 otherwise.
- **`/save`**: An endpoint that stores in a .txt file a provided dictionary in JSON format that will later use **`/initial_info`** to return the initial information of the game.

3 Project structure

We list and explain below the purpose of each file/folder the project contains:

- **package.json**: This file contains the list of npm [3] dependencies the back-end server needs. In this case, we will need the *express* package to be able to start an *Express.js* server, and the *cors* package to be able to accept requests coming from localhost. Before starting up our server, we need to install those dependencies using command **`npm install express && npm install cors`**.
- **app.js**: This file contains the code that starts up the *Express* server on port 8081. All the code of this exercise must be implemented in this file. To start up our server, we must execute the file with NodeJS using command **`node app.js`** (or **`nodemon app.js`** if you want automatic refresh).
- **public**: This folder contains a compiled version of the Vue front-end application that you will implement on the second part of the project. Once the server is running, you can access this front-end through *http://localhost:8081*.

4 Endpoints

In this exercise you will need to implement some endpoints using *Express.js* routing [4]. We list below the endpoints to implement, specifying for each one the required method type, and the task it must perform.

4.1 `/initial_info`

This endpoint must be a GET method. This endpoint is called by the front-end just at the beginning. It must return a dictionary with the format presented above. This endpoint must consider two use cases depending on whether the file 'save_file.txt' exists:

- **The file does not exist**: This means the **`/save`** endpoint has not been called yet. We can check if a file exists using *existsSync* function from NodeJS's *fs* package [5]. In this case, the response must be a JSON representing a dictionary with default values. Be cautious of not returning 'x' and 'y' values that place the player on a section of the map without access to grass tiles (because you will not be able to test the incoming endpoints). An example of default values could be:

```
{
    "x": 15,
    "y": 6,
    "direction": "south",
    "pokedex": {}
}
```

- **The file exists**: This means the **`/save`** endpoint has been called at least once. The content of the file should be a JSON string with identical format to the one presented above. We MUST use streams to pipe the content of the file directly to the response of the request. We can create a read stream on the file using *createReadStream* function from NodeJS's *fs* package [5].

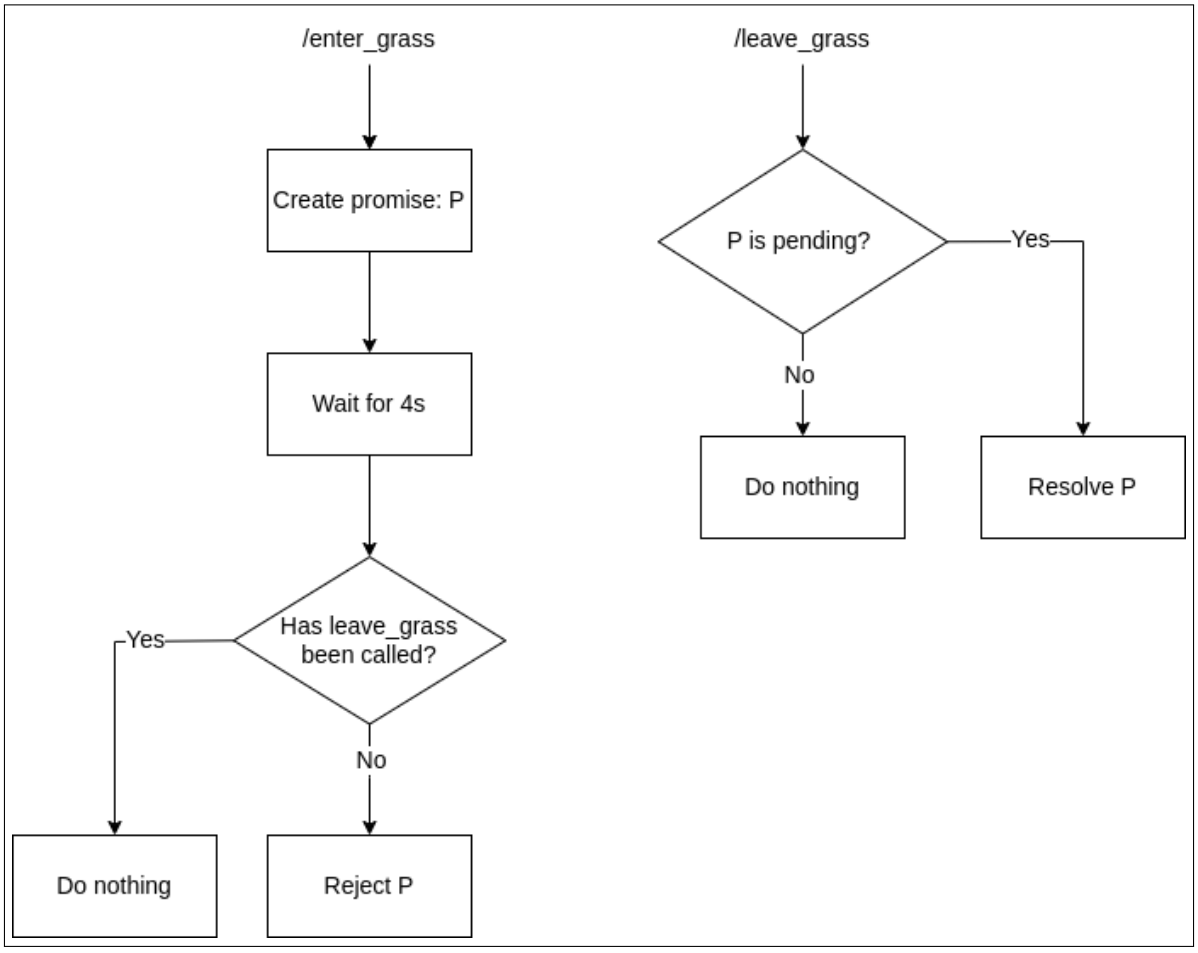

Figure 1: Flux diagram of endpoints **`/enter_grass`** and **`/leave_grass`**

4.2 `/enter_grass`

This endpoint must be a GET method. This endpoint is called by the front-end every time the player enters a patch of grass tiles on the map (Figure 2).

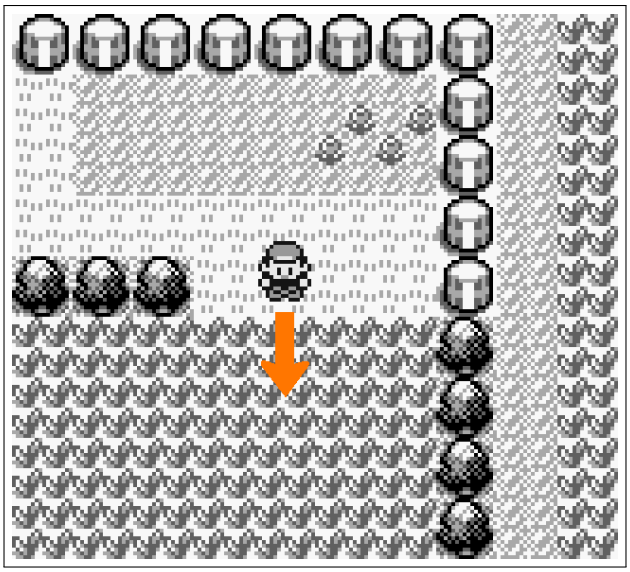

Figure 2: Player about to enter a patch of grass

On the provided app.js skeleton, we can see a commented bit of code similar to the following:

```
const Game = function() {
    this.enterGrass = function() {
        .....
    }
    this.capturePokemon = function() {
        .....
    }
};
const gameObj = new Game();
```

We must implement the **`enterGrass`** function. This function must return a promise, that must reject without parameters in case a wild encounter has occurred, and resolve without parameters otherwise.

`/enter_grass` endpoint must use the **`gameObj`** object to call this function. In case **`enterGrass`** promise resolves, **`/enter_grass`** endpoint must return an status code [6] of 200; and in case it rejects, must return an status code of 400.

Let's take a look at Figure 1. To decide if a wild encounter has occurred, the first thing **`enterGrass`** function must do is create the promise and wait for 4 seconds. During that timelapse, **`/leave_grass`** endpoint must be able to resolve the **`enterGrass`**'s promise at any moment (meaning we have exited the grass, and thus a wild encounter must not occur). Once that time has elapsed, we must check if the promise has been already resolved by **`/leave_grass`**; and if it hasn't, we must reject it ourselves (meaning the player is still inside the patch of grass, and the front-end must trigger a wild encounter).

On summary, the **`enterGrass`** function must create a promise, wait 4 seconds, and reject the promise just if it has not been resolved while waiting.

From the perspective of the front-end, it will call **`/enter_grass`** endpoint once the player enters a patch of grass; and then, if the player manages to exit the patch before 4 seconds, it will call **`/leave_grass`** endpoint, which will provoke **`/enter_grass`** to return 200 immediately after, making the front-end not trigger a wild encounter.
On the other hand, if the player does not exit the patch of grass during 4 seconds, **`/enter_grass`** will return 400 after 4 seconds, and the front-end will trigger a wild encounter.

4.3 `/leave_grass`

This endpoint must be a GET method. It is called by the front-end every time the player exits a patch of grass tiles (Figure 3).

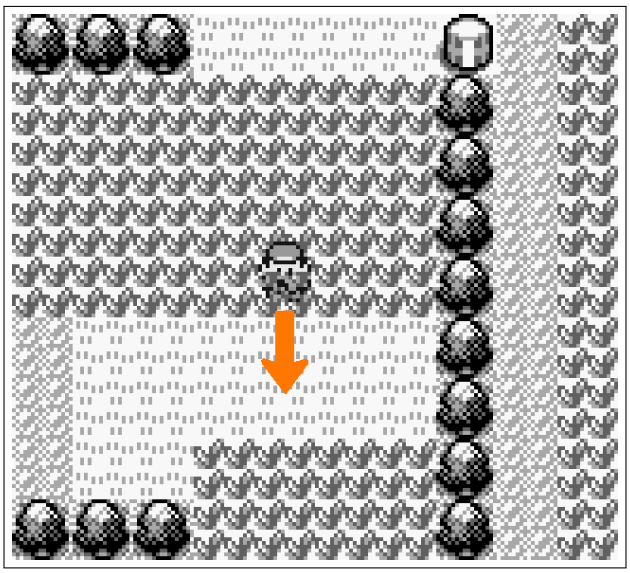

Figure 3: Player about to exit a patch of grass

Let's take a look again at Figure 1. This endpoint must check if there is a 'pending' promise created by the **`/enter_grass`** endpoint. Two possibilities could happen:

- **The promise is still pending**: Meaning **`/enter_grass`** has still not rejected it. In that case, we must resolve the promise immediately.
- **The promise has been rejected**: Meaning 4 seconds have passed since the promise was created, and **`/enter_grass`** has already rejected the promise and returned an status code of 400 (triggering a wild encounter on the front-end). If this is what happened, we must simply ignore it and continue.

In either case, after all that, we must respond with an status code of 200.

Given the fact that we must check if a promise created by another endpoint it's still pending, it is infered that **`/enter_grass`** endpoint must store something that later **`/leave_grass`** will check/interact with. It is PROHIBITED to store any global variable in **app.js** file. All variables or functions that you need, must be contained inside the **`Game`** class.

4.4 `/capture`

This endpoint must be a GET method. This endpoint is called by the front-end when, inside a wild encounter, we choose the option to capture the Pokemon (Figure 4). Inside the **`Game`** class, just below the **`enterGrass`** function, we can see a function called **`capturePokemon`**. We must implement that function.

`capturePokemon` function must return a promise that resolves without parameters if the Pokemon was caught; or rejects without parameters if the Pokemon escaped the Pokeball. **`/capture`** endpoint must use the **`gameObj`** object to call this function, and, if the promise resolves, it must respond with an status code of 200. On the other hand, if the promise rejects, it must respond with an status code of 400.

In **`capturePokemon`** function, to decide if the Pokemon must be caught or escape the Pokeball, we must perform the following steps:

1. Wait for one second.
2. Calculate a boolean which it's value must be **`true`** in 80% of the cases. On the provided **app.js** file we can see a commented bit with a boolean that will be **`true`** on 80% of the cases: **`Math.random() < 0.8`**. Once we got the boolean result:

(a) If it's `false`, we must reject the promise immediately.

(b) If it's `true`, we must go back to step 1. If the boolean has been `true` three times in a row, instead of going back to step 1 again, we must resolve the promise.

Hint: Using recursion could be a good way keep your code short and simple.

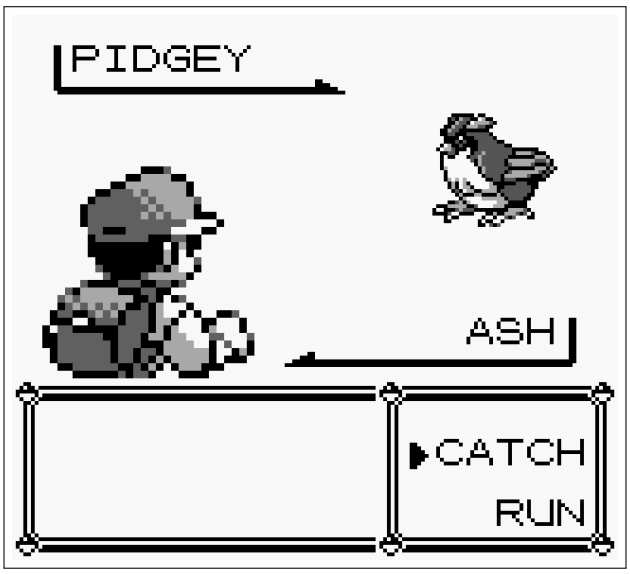

Figure 4: Wild encounter main screen

From the front-end perspective, once you choose the capture option, it will perform the call to the `/capture` endpoint, and you will see the animation of the Pokeball moving. Once the endpoint responds, two things can happen:

- **The endpoint responds status code 200**: You will see a text informing that the Pokemon has been caught (Figure 5), and you will be able exit the wild encounter.
- **The endpoint responds status code 400**: The Pokemon will appear again, and you will simply pop back to being able to choose between capturing, or running away (Figure 4).

Figure 5: Pokemon captured in a wild encounter

4.5 `/save`

This endpoint must be a POST method. As you might have inferred by now, the back-end is not keeping track in real time of either the player's position on the map, the facing direction, or the captured/seen Pokemons. This is all being stored on the front-end. But if we reload the page, all this information is lost and we go back to the default values **`/initial_info`** endpoint provides to us. To be able to preserve our position, facing direction and Pokedex through page reloads, there is an option on the menu of the front-end to be able to save the game (Figure 6).

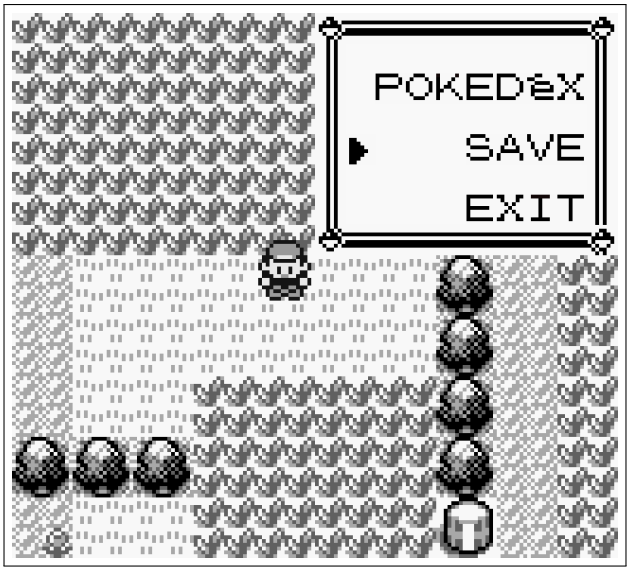

Figure 6: 'SAVE' option on the menu

If, at any point, we choose this option, the front-end will perform a request to the **`/save`** endpoint, sending a dictionary on the body of the request with the 'x' and 'y' position of the player on the map; the 'direction' is facing, and a dictionary representing the 'pokedex'.

The **`/save`** endpoint must convert the request body to a JSON string (for that purpose, you can use JSON.stringify function [7]) and store it in a file called *save_file.txt*. If it's not the first time the **`/save`** endpoint has been called, and the file already exists, we must simply override the content of the file with the new information. Once the file is written, we must respond with status code 200.

Despite being mandatory on **`/initial_info`** to pipe the information of the save file using streams, it is not mandatory to use it in this endpoint. Instead, as a way of practicing promises, we recommend using the promise API from *fs* package (In the skeleton we already provide you the require "fs/promises").

4.6 `/???`

The mysterious endpoint. Before implementing it, make sure you have implemented the rest of endpoints, and that you have passed the tests presented in Section 5.

You do not know the endpoint's name, you do not know the method type, and you do not know what you must respond. Here is the riddle: Once you own all the Pokemons in the grass, reading is the only path to the third.

Only the first group to upload the implemented endpoint to the Github Classroom repository will get 0.5 extra points on the final grade.

5 Testing

5.1 Internal tests

To check if you have correctly implemented the endpoints and the functions inside **`Game`** class, we provide a function called **`internalTests`** at the bottom of **app.js**. This function only runs a few tests to cover the most obvious use cases, but it can be a good starting point to see if you are on the right track. To use it, simply uncomment the function call on **`app.listen`**'s callback.

Startup the server using command **`node app.js`**, and, after a while, if the tests have passed correctly it should appear something like the following in the console:

```
test_enterGrassReject finished.
test_enterGrassResolve finished.
test_capturePokemon finished.
Tests finished.
```

5.2 Manual testing

First, execute your back-end with command **`node app.js`**. Once the server is running, you can go to the browser of your choice and navigate to **`http://localhost:8081`**. If the endpoints are correctly implemented, you should see something similar to the image in Figure 7.

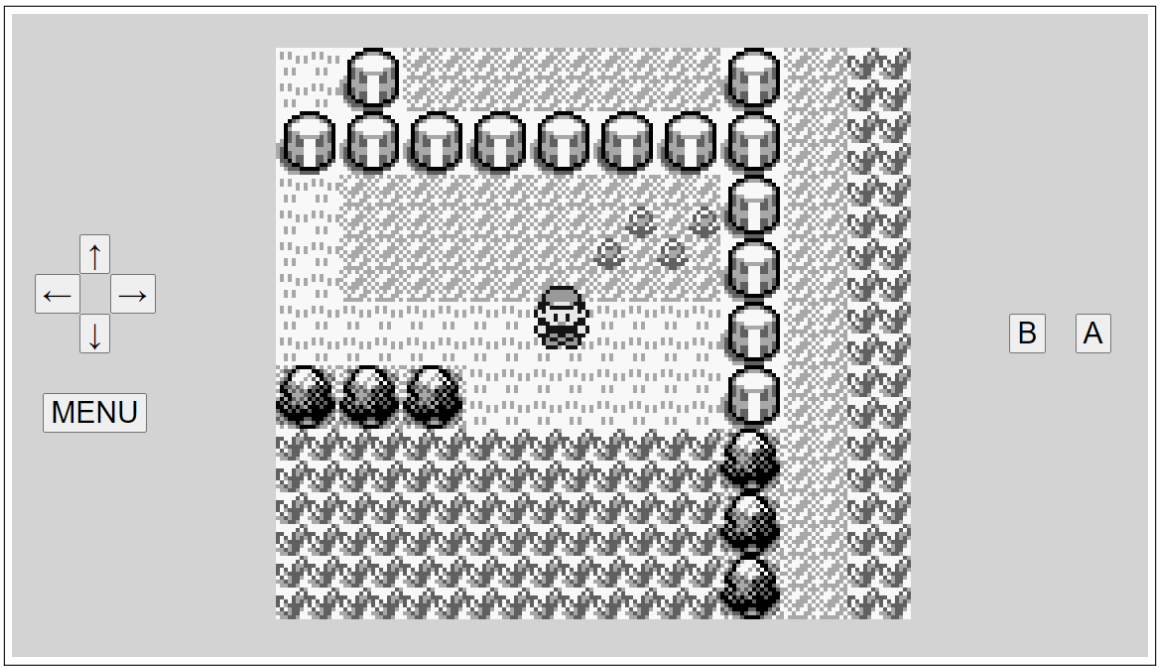

Figure 7: Example of the front-end page. Game at the center, and buttons on each side

If, instead of that, you see something similar to Figure 8, most definitely means the **`/initial_info`** endpoint is not correctly implemented. Error descriptions should help you understand what is going on.

Once no more error screens appear, you can move your player through the map using the arrow buttons; open and close the menu using the MENU button below the arrows; cancel or close dialogs using the B button; and accept or choose options using the A button.

5.2.1 Manual Test 1: Initial location and facing direction

Changing manually the 'x', 'y' and 'direction' attributes on the default dictionary that **`/initial_info`** sends should make the player change his position and facing direction accordingly.

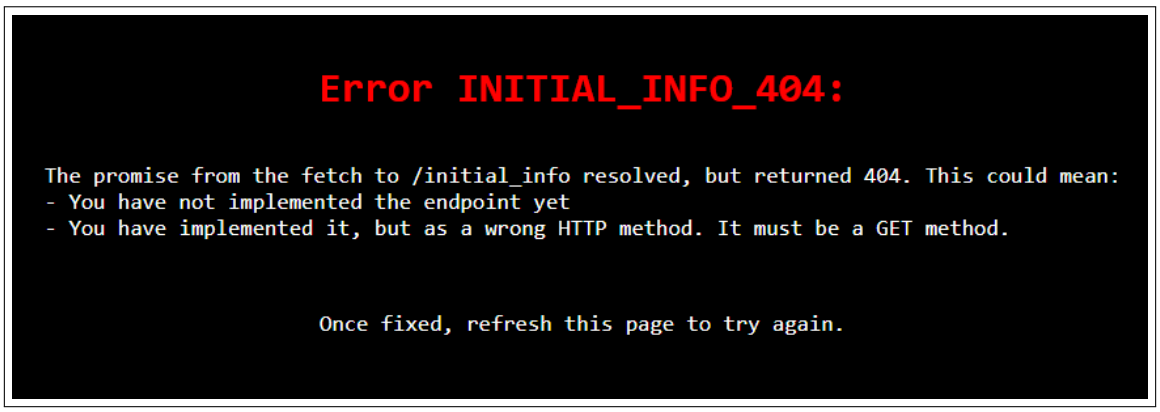

Figure 8: Example error screen. Error identifier above, and explanation below

5.2.2 Manual Test 2: Entering grass and entering a wild encounter

If we move the player towards a patch of grass and enter it, we should see that the front-end performs a call to **`/enter_grass`** endpoint. If something wrong happens with the call, more error screens will appear. Once no more error screens appear, wait 4 seconds for the **`/enter_grass`** endpoint to respond, and you should see the front-end changes to a wild encounter screen. Run away from the encounter and move on to the next test.

5.2.3 Manual Test 3: Exiting the grass in two different ways

Once you run away from the wild encounter, you should still be inside the patch of grass. Once you exit the patch, you should see that the front-end performs a call to **`/leave_grass`** endpoint. If something wrong happens with the call, more error screens will appear. Once no error screens appear, try entering the patch of grass again, but this time exit the patch before a wild encounter happens (a.k.a, before 4 seconds). Check if more error screens appear, and if not, move on to the next test.

5.2.4 Manual Test 4: Capturing a Pokemon

Enter a patch of grass again to provoke a wild encounter. This time, instead of running away, choose the option to capture the Pokemon. The front-end will perform a call to **`/capture`** endpoint, and if something wrong happens with the call, more error screens will appear. If the Pokemon escapes the Pokeball, repeat the process until the Pokemon is captured.

5.2.5 Manual Test 5: Saving the game

At this point, with a captured Pokemon, and maybe with another one seen (from Manual Test 2), it's time to save the game. Click the MENU button, and select the 'SAVE' option. The front-end will perform a call to the **`/save`** endpoint. If something wrong happens with the call, more error screens will appear. If no error screens appear, you should see the front-end informing you that the game has been saved.

5.2.6 Manual Test 6: Reloading the page

At this moment, if you reload the page, **`/initial_info`** endpoint should, instead of responding with the default dictionary, respond you with the information saved on the previously called **`/save`** endpoint. If you appear on the same spot as you were, facing the same direction, and with the same seen/owned Pokemons as you had before (you can check seen/owned Pokemons selecting the 'POKEDEX' option on the menu), **`/initial_info`** and **`/save`** endpoints seem to work correctly.

5.2.7 Manual Test 7: Checking save overwrite

Finally, you can move a little bit, or capture another Pokemon, and try saving the game again to see if the **`/save`** endpoint correctly overrides *save_file.txt* with the new information. Reload the page to check if the **`/initial_info`** endpoint correctly provides the front-end with the updated information.

References

[1] **NodeJS website**: https://nodejs.org

[2] **Express framework.**: https://expressjs.com

[3] **npm website**: https://www.npmjs.com

[4] **Express basic routing.**: https://expressjs.com/en/starter/basic-routing.html

[5] **fs package**: https://nodejs.org/dist/latest/docs/api/fs.html

[6] **Response status property**: https://developer.mozilla.org/en-US/docs/Web/API/Response/status

[7] **JSON.stringify**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify