In the 2007 game *Lego Star Wars: The Complete Saga*, information regarding characters is stored in various plain-text files, using a non-standard data format. This information includes data such as the character’s name, their icon, and their price to buy in the in-game shop. If people want to add new characters, they have to edit these files by hand. By moving this data to standard JSON data, it would be possible to easily edit this information using a GUI. This web application simulates such a GUI, allowing users to create, modify, and delete characters and their data without tediously editing files by hand.

When a user first looks at the page, he or she sees a new character button as well as a list of characters, each with three adjacent buttons labeled show info, edit, and delete. If any of the buttons are clicked, an additional section appears which updates dynamically depending on which buttons are pressed. When you click the new character button, it brings up some text boxes for the user to enter the localized character name (the actual name of the character in English), the internal name (used for the model file name and in this application to refer to the JSON entry), the price, and URL for the icon image. If the user hits submit, the program checks to ensure that the input is valid and that the internal name is not a duplicate. If those checks are passed, the character list gets updated, and the “new character” pop-ups disappear. If the user clicks the edit button next to a character, a very similar behavior occurs, but the text boxes are automatically filled with the information from the JSON data, and successfully submitting without any errors, such as a duplicate name or invalid input, changes the specified character instead of adding to the list. Clicking the show info button displays the character’s name, internal name, and price. Clicking the delete button brings up a prompt that says which character the delete button was pressed for and asks the user to verify that that character should be deleted. Deleting characters removes them from the character list.

|  |  |
| --- | --- |
| Interface is Authentic, Professional, Balanced Interface design is authentic, looks professional, is balanced across the web page | Interface has been formatted to have unobtrusive thematically-appropriate styling. |
| Interface links to functional jQuery jQuery is included in application source | Links to a locally stored shared jQuery library, to avoid dependencies upon external websites. |
| Events are Effective, jQuery jQuery is used to bind events to event handlers<br/> The event handling mechanism works to achieve goals. | jQuery is used when possible, such as in Init() here: $("#button\_create").on("click",ShowCreate); |
| DOM Access, jQuery The DOM is accessed with jQuery | jQuery is the only way the DOM is accessed. |
| DOM Dynamically Built, jQuery The DOM is dynamically updated using jQuery calls | jQuery is the only way the DOM is updated. |
| Documentation: Program Code and Project Description .js file is fully annotated, explaining the code, not just stating that a call is being used.<br/> There is a document describing how the application works and how to use it, and the goal of the applicat | Comments thoroughly explain the developer’s thoughts behind each design decision. Documentation is provided in the paragraphs above explaining the purpose and intended usage. |
| Style, jQuery In at least one instance, style is modified or applied using jQuery | jQuery is used to color the text of the error messages.  For example, in UpdateChar(): $("#popup\_invalid").css("color","red"); |

/\*

Functions: PascalCase

Variables: snake\_case (mostly)

\*/

/\*

global vars and consts

\*/

let invalid\_popup\_enabled = false;

/\*

functions

\*/

const ListChars = function()

{

    let output\_text = `<table id="char\_table">`; // start table

    if (true) // limit varaible scope

    {

        let found\_char\_ghg = null; // initalizing to null prevents garbage data (or at least it does in ancient 90s versions of C)

        let found\_char\_icon = null;

        let found\_char\_name = null;

        characters.forEach((character) =>

        {

            found\_char\_name = character["char\_name"];

            found\_char\_ghg = character["char\_intn\_name"];

            found\_char\_icon = character["char\_icon"];

            // this shouldn't be a table, because tables can't be dynamic.

            output\_text += `<tr>`

                        + `<td style="text-align:center"><img src="${found\_char\_icon}" alt="${found\_char\_name}" width="64px" height="64px"><br>${found\_char\_name}</td>`

                        // if we write onclick directly, we can define an arg. For some inexplicable reason,

                        // functions sometimes have to be called without parentheses in js, which is the case for

                        // jQuery's event handler binder. There may be a way to work around it, but if there is,

                        // I didn't learn how in class. This would be much cleaner if functions always had

                        // parentheses like in C.

                        + `<td><button onclick="ShowRead('${found\_char\_ghg}')">Show Info</button><br>`

                        + `<td><button onclick="ShowUpdate('${found\_char\_ghg}')">Edit</button><br>`

                        + `<td><button onclick="ShowDelete('${found\_char\_ghg}')">Delete</button><br>`

                        + `</tr>`;

        });

    }

    output\_text += `</table>` // end table

    $("#char\_list").html(output\_text)

    $("#char\_list").show();

    $("#popup\_ui").hide(); // HACK HACK HACK. Likes to show up every time we update the list. Fortunately, it never needs to, so we can hide it here.

}

const CheckIfCharExists = function(char2find)

{

    let found\_char = characters.find((elem) => elem["char\_intn\_name"] == char2find);

    if (found\_char != undefined) // undefined == not found

    {

        // character exists

        return true;

    }

    else

    {

        // character doesn't exist

        return false;

    }

}

const ShowCreate = function()

{

    let ui\_html = `<table><tr><td>Character name (English): </td><td><input id="char\_name"/><br></td></tr>`

                + `<tr><td>Internal name (GHG filename): </td><td><input id="char\_intn\_name"/></td></tr>`

                + `<tr><td>Price: </td><td><input id="char\_price"/></td></tr>`

                + `<tr><td>Icon URL (should be square): </td><td><input id="char\_pic\_url"/></td></tr></table>`

                + `<button id="submit\_char">Submit</button>`;

    $("#popup\_ui").html(ui\_html);

    $("#popup\_ui").show();

    $("#submit\_char").on("click",CreateChar)

}

const CreateChar = function()

{

    // Grab inputs

    let new\_name = $("#char\_name").val();

    let new\_intn\_name = $("#char\_intn\_name").val();

    let new\_price = parseInt($("#char\_price").val());

    let new\_pic\_url = $("#char\_pic\_url").val();

    // like DOS, we default to uppercase.

    new\_intn\_name = new\_intn\_name.toUpperCase();

    if (new\_price == NaN || new\_name == "" || new\_intn\_name == "" || new\_price == "" || new\_pic\_url == "") // this stopped triggering for NaN for some reason

    {

        // don't attempt to add invalid input; return early

        // not going to verify the img is valid; that's out of scope

        if (!invalid\_popup\_enabled)

        {

            $("#popup\_ui").append(`<p id="popup\_invalid">Invalid input</p>`)

            invalid\_popup\_enabled = true;

            $("#popup\_invalid").css("color","red"); // color me red

        }

        else

        {

            $("#popup\_invalid").text("Invalid input");

        }

        // return one because user error

        return 1;

    }

    else if (CheckIfCharExists(new\_intn\_name) == true)

    {

        // don't create dupes of chars

        if (!invalid\_popup\_enabled)

        {

            $("#popup\_ui").append(`<p id="popup\_invalid">Already exists!</p>`)

            invalid\_popup\_enabled = true;

            $("#popup\_invalid").css("color","red"); // color me red

        }

        else

        {

            $("#popup\_invalid").text("Already exists!");

        }

        // return one because user error

        return 1;

    }

    invalid\_popup\_enabled = false; // reset global var to avoid issues next time

    let new\_char =

    {

        char\_name : new\_name,

        char\_intn\_name : new\_intn\_name,

        char\_price: new\_price,

        char\_icon: new\_pic\_url,

    }

    characters.push(new\_char);

    $("#popup\_ui").hide();

    ListChars();

    // return zero because everything worked correctly (hopefully)

    return 0;

}

const ShowRead = function(char2read)

{

    let found\_char = characters.find((elem) => elem["char\_intn\_name"] == char2read)

    if (found\_char != undefined)

    {

        let ui\_html = `<table><tr><td>Character name (English): </td><td>${found\_char["char\_name"]}</td></tr>`

                    + `<tr><td>Internal name (GHG filename): </td><td><code>${found\_char["char\_intn\_name"]}</code></td></tr>`

                    + `<tr><td>Price: </td><td>${found\_char["char\_price"]} studs</td></tr>`;

        $("#popup\_ui").html(ui\_html);

        $("#popup\_ui").show();

        return 0;

    }

    else

    {

        console.log(found\_char);

        console.log("Character not found. Perhaps this function was called incorrectly?")

        return 1;

    }

}

const ShowUpdate = function(char2read)

{

    let found\_char = characters.find((elem) => elem["char\_intn\_name"] == char2read)

    if (found\_char != undefined)

    {

        let ui\_html = `<table><tr><td>Character name (English): </td><td><input id="char\_name" value="${found\_char["char\_name"]}"/><br></td></tr>`

                    + `<tr><td>Internal name (GHG filename): </td><td><input id="char\_intn\_name" value="${found\_char["char\_intn\_name"]}"/></td></tr>`

                    + `<tr><td>Price: </td><td><input id="char\_price" value="${found\_char["char\_price"]}"/></td></tr>`

                    + `<tr><td>Icon URL (should be square): </td><td><input id="char\_pic\_url" value="${found\_char["char\_icon"]}"/></td></tr></table>`

                    + `<button id="submit\_char" onclick="UpdateChar('${found\_char["char\_intn\_name"]}')">Submit</button></p>`;

        $("#popup\_ui").html(ui\_html);

        $("#popup\_ui").show();

        return 0;

    }

    else

    {

        console.log(found\_char);

        console.log("Character not found. Perhaps this function was called incorrectly?")

        return 1;

    }

}

const UpdateChar = function(char2update)

{

    // grab inputs

    let new\_name = $("#char\_name").val();

    let new\_intn\_name = $("#char\_intn\_name").val();

    let new\_price = parseInt($("#char\_price").val());

    let new\_pic\_url = $("#char\_pic\_url").val();

    // like DOS, we default to uppercase.

    new\_intn\_name = new\_intn\_name.toUpperCase();

    if (new\_name == "" || new\_intn\_name == "" || new\_price == "" || new\_price == NaN || new\_pic\_url == "") // apparently "" =/= null? why?

    {

        // don't attempt to add invalid input; return early

        // not going to verify the img is valid; that's out of scope

        if (!invalid\_popup\_enabled)

        {

            $("#popup\_ui").append(`<p>Invalid input</p>`)

            invalid\_popup\_enabled = true;

            $("#popup\_invalid").css("color","red"); // color me red

        }

        // return one because user error

        return 1;

    }

    else if (CheckIfCharExists(new\_intn\_name) == true)

    {

        // don't create dupes of chars

        if (!invalid\_popup\_enabled)

        {

            $("#popup\_ui").append(`<p id="popup\_invalid">Already exists!</p>`)

            invalid\_popup\_enabled = true;

            $("#popup\_invalid").css("color","red"); // color me red

        }

        else

        {

            $("#popup\_invalid").text("Already exists!");

        }

        // return one because user error

        return 1;

    }

    invalid\_popup\_enabled = false; // reset global var to avoid issues next time

    for (let i = 0; i < characters.length; i++)

    {

        if (characters[i]["char\_intn\_name"] == char2update)

        {

            characters[i]["char\_name"] = new\_name;

            characters[i]["char\_intn\_name"] = new\_intn\_name;

            characters[i]["char\_price"] = new\_price;

            characters[i]["char\_icon"] = new\_pic\_url;

        }

    }

    $("#popup\_ui").hide();

    ListChars();

    // return zero because everything worked correctly (hopefully)

    return 0;

}

const ShowDelete = function(char2del)

{

    // extract character to make our code less convoluted

    let found\_char = characters.find((elem) => elem["char\_intn\_name"] == char2del)

    if (found\_char != undefined) // undefined == doesn't exist

    {

        let ui\_html = `<p>You want to delete ${found\_char["char\_name"]} (<code>${found\_char["char\_intn\_name"]}</code>). Are you sure?</p>`

                    + `<button id="submit\_char" onclick="DeleteChar('${found\_char["char\_intn\_name"]}')">Confirm Delete</button></p>`;

        $("#popup\_ui").html(ui\_html);

        $("#popup\_ui").show();

        return 0;

    }

    else

    {

        console.log(found\_char);

        console.log("Character not found. Perhaps this function was called incorrectly?")

        return 1;

    }

}

const DeleteChar = function(char2del)

{

    for (let i = 0; i < characters.length; i++)

    {

        if (characters[i]["char\_intn\_name"] == char2del)

        {

            characters.splice(i,1);

            console.log(`deleted ${char2del}`)

        }

    }

    $("#popup\_ui").hide();

    ListChars();

}

const Init = function()

{

    $("#popup\_ui").hide(); // will show error message if somehow doesn't get hidden

    $("#char\_list").hide(); // hide while loading, in case cpu is slow

    ListChars();

    $("#button\_create").on("click",ShowCreate);

}

$(document).ready(Init);