# Web Application Documentation

## General

### Libraries and Other Technologies

This web application uses PHP, JavaScript, JQuery, ThreeJS and a number of related Javascript packages.

PHP: <https://www.php.net/>

JavaScript: <https://www.javascript.com/>

JQuery: <https://jquery.com/>

WebGL: <https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API>

ThreeJS: <https://threejs.org/>

dat.gui: <https://github.com/dataarts/dat.gui>

Unity: <https://unity.com/>

### General navigation

The navigation bar at the top of the page has three links.

The logo is a link to user.php on the user.php and editor.php pages.

The words “My Portals” is a link to user.php on the user.php and editor.php pages.

The words “Log Out” is a link to logout.php on the user.php and editor.php pages, functionality to be expanded upon in the User System section below.

### User System

For a user to log in, they must enter their username and password into index.php.

Upon submission, the PHP script on the same page checks it against the database of users in users.json in the users folder. If the username and password match an entry within the database, a ten character alphanumeric session key is generated. Both this and the user ID are stored as cookies in the browser.

The username is entered into the input field named username and the password is entered into the input field named password.

The cookies are userid for the user id and sesID for the session key.

Each page will then check these cookies to ensure that they are valid and simultaneously update the session key with a new, randomly generated 10 characters alphanumeric session key.

The session keys are generated with the cryptographically secure random\_int(int $min, int $max) function available in php7.

Keys will last 30 days before being naturally expiring.

Pressing the log out button on any page sends the user to logout.php, which simply clears the cookies and redirects the user to the login page.

## User Page

user.php is the main user page. It contains the list of all the worlds created by the user and allows the creation of new worlds, editing the metadata of existing world, deleting worlds and provides the proper format for opening the world editor.

### World Listings

Each of the worlds is listed using PHP. The script on the user.php page reads the cookies, and after validating the userID, checks the users.json file to find the worlds the user owns.

Each of these worlds is queried using a function to gather their data which is then displayed, using a series of concatenations to produce a string of a template that is echoed to create a world listing. This listing also automatically fills in certain specific data in the function calls made by the buttons labelled Edit, Metadata, Share and Delete.

This will be expanded upon below.

### User Page Panels

Clicking one of the buttons labelled New World, Metadata, Share or Delete reveals a floating panel above the page and a semi-transparent backdrop. These work with the JQuery .hide() and .show() functions.

Clicking one of these buttons will show the backdrop and the corresponding panel. Clicking on the backdrop or the cancel button on the panel will attempt to hide all panels and the backdrop. This is done by hiding all items with the class worldEditPanel as well as specifically targeting the backdrop which has the ID panelBack.

Hiding the panels is done the function hideENW().

Showing the panels are done with specific functions that will be detailed in the corresponding section.

### World Creation

To create a world, the user must click the button labelled new world.

Clicking this will run the function showNewWorld(). This function simply runs the JQuery .show() function on the elements with the IDs panelBack and newPanel.

The panel contains a simple form, consisting of four input fields and two buttons. The form is not an HTML form, but rather, powered by Javascript and JQuery.

Clicking the button Create World will run the function createWorld().

The function first reads the textarea with the ID NW\_Tags, the tags section of the form. It splits the tags at every comma (“,”) using the .split(string splitter) function of JavaScript. It then iterates though this list of tags, first removing white space using the .trim() function and checks if it exists in a second array. If it does not, this tag is added to the array. If not, the tag is discarded. This creates a list of unique tags.

This list of tags as well as the data from tags are used to construct an object as follows.

var newWorld = {

"name" : $("#NW\_Name").val(),

"description" : $("#NW\_Description").val(),

"tags" : uniqueTags,

"private": $("#NW\_Private").val(),

"objects" : [],

"owner" : userID

}

This creates the base for the world JSON that acts as the main constructor for any scenes.

The elements with the IDs NW\_Name, NW\_Description and NW\_Private are the remaining elements from the new world form, providing the name of the world, the description of the world.

This file is sent via three synchronous ajax requests to other pages in order to create the world. The requests are done synchronously in order to allow the requisite files to be created before the function forces the page to refresh in order to display the newly created world. The refresh is done using the JQuery location.reload() function.

The first sends the data, along with to newWold.php in the worlds folder.

$.ajax({

type: 'POST',

url: "worlds/newWold.php",

data: {

json : JSON.stringify(newWorld),

wold: worldID

},

async:false

});

The newWold.php script first checks that the string is a valid JSON before creating a JSON file with the world ID as a name and writes the JSON string into the file.

The second AJAX requests send the world ID and the user ID to the newWorld.php page in the users folder.

$.ajax({

type: 'POST',

url: "users/newWorld.php",

data: {

name: worldID,

user:userID

},

async:false

});

After checking that the inputs are set, that they are strings and that they are not empty, the script will parse the users.json file and add the world ID to the list of worlds owned by that user before saving the changes made to the file.

The final AJAX request is made to the newWorld.php page in the folder Theater at the local address AssetBundleBuilder/Assets/Theater/newWorld.php. This sends the world ID, the user ID and the world password.

$.ajax({

type: 'POST',

url: "AssetBundleBuilder/Assets/Theater/newWorld.php",

data: {

name: worldID,

user: userID,

pass:$("#NW\_Private").val()

},

async:false

});

This information is used to create a simple standardised filesystem for storing uploaded files.

After checking that the inputs have been made and that they are not empty, the system first creates a folder with the world’s name. Then, two more folders, one named fbx and the other named assetbundle are created inside. Then a file called metadata.json is created, with a simple JSON string saved within it stating the owner of the world and the password for the world.

### World Deletion

Pressing the button labelled delete for a world runs the function showDelete(World ID, World Password). The world id and the world password are added by the PHP script and are specific to the world.

The function runs the .show() function on the elements with the IDs panelBack and deletePanel but it also changes two additional values.

The variables editWorldId and editWorldPass are both set to the world ID and the world password respectively.

The panel revealed requires that the user enter their password in order to delete the file.

When the password is entered and the form is submitted, the deleteWorld() function is run.

This function performs three AJAX requests before reloading the page in order to refresh the list of worlds.

The first is to deleteWorld.php in the users folder.

$.ajax({

type: 'POST',

url: "users/deleteWorld.php",

data: {

name:editWorldId,

user:userID,

pass:$("#DWPassword").val()

},

async:false,

success: function(value){

valid = value;

}

});

In addition to sending the userID and the user’s password, it also receives confirmation from the server that the action has been carried out.

The seleteWorld.php script first checks the inputs before using the array\_diff(array1, array2) function of PHP to remove the world from the list of worlds the user owns. It then responds, signalling that the task is complete.

If this flag is set, then the other two AJAX calls are made. (see next page)

These calls effectively send the same information, the user ID, the world ID and the word password to the specified pages. These scripts check if the file in question exists, if the user ID matches the owner tag and if the password sent matches the world password. If all three are true, they delete their respective files.

$.ajax({

type: 'POST',

url: "AssetBundleBuilder/Assets/Theater/deleteWorld.php",

data: {

name: editWorldId,

user:userID,

pass:editWorldPass

},

async:false

});

$.ajax({

type: 'POST',

url: "worlds/deleteWorld.php",

data: {

name: editWorldId,

user:userID,

pass:editWorldPass

},

async:false

});

### World Details Editing

Editing the world metadata is similar to deleting a world. Upon selecting the button labelled Details the function showEdit(world ID, world name, world description, tag string, world password) is called. The tag string is generated by using the PHP join function with a comma as a joining string.

This function, in addition to revealing the editing panel and storing the world ID and world password into the same variable used by the deleteWorld() function, also populates the form with the information belonging to the world, placed into the function by the PHP script creating the world listings. The data population is done using the .val() function from JQuery.

After the changes are made, pressing the button labelled Update World calls the updateWorld() function.

First, this gets the world data structure from it’s JSON and updates it with the new data from the form, after using the same process as the createWorld() function to ensure the tags are all unique. Then, it makes two AJAX calls.

The first AJAX call sends the full JSON to the updateWorld.php script in the worlds folder. After validating the inputs, it checks that the provided password is the same as the world password and that the userID matches the owner of the file. If so, it writes the JSON string to the world file.

$.ajax({

type: 'POST',

url: "worlds/updateWorld.php",

data: {

json:JSON.stringify(currentWorld),

name:editWorldId,

pass:editWorldPass,

user:userID

},

async:false

});

The second AJAX call is to the updatepass.php script in the Theater folder (full location: AssetBundleBuilder/Assets/Theater/updatepass.php). After validating the input, it compares the user ID provided and the owner ID as well as the password to the world password. If they match, the password for the world is updated to match the new password.

$.ajax({

type: 'POST',

url: "AssetBundleBuilder/Assets/Theater/updatepass.php",

data: {

newpass : $("#EW\_Private").val(),

name: editWorldId,

pass:editWorldPass,

user:userID

},

async:false

});

### World Sharing

Sharing a world operates in a similar manner to the other functions. A panel is revealed as the editWorldId variable is set. The user then inputs a list of email addresses, separated by commas. The resultant string is split at the commas, that array is trimmed to 20 strings in length and then any strings that do not meet the regex match for an email address are removed.

This array is converted into a JSON string and sent via AJAX request to the shareWorld.php page in the scripts folder. Once this is complete, the hideENW() function is run.

$.ajax({

type: "POST",  
url: "scripts/shareWorld.php",

data: {

wold:editWorldId,

adresses:JSON.stringify(uniqueTags)

}

});

The php script validates the inputs and checks again to ensure that a maximum of twenty email addresses have been provided. It then uses the send mail add on for XAMPP to send a simple email to the specified recipients.

### World Editing

The button labelled Edit is actually the only visible element of a HTML form. Two hidden input fields named room and pass are also present and are populated by PHP scripts to have the world ID and world password. Upon selecting this button, the form is submitted to the editor.php page. Further details in the next section.

## World Editor

The world editor page primarily makes use of a Three.JS viewer to render the world as per the instructions of the .json file used to save the state of the world. It also allows the user to edit the items on this ThreeJS viewer using click and drag controls or by manually entering values into the Active Object tab.

### Loading World

Upon entering the page, the PHP will read the data sent by the form on user.php and confirm that the user in question has access to the world, checking their userID against the world ownership. If not, the user is sent back to user.php.

If they are allowed to edit the page, the page populates the Assets and Uploads tabs, in both these tabs, a short PHP script scans the relevant directory, and after filtering out unnecessary files produces a series of buttons that call the function loadFBX(model location, model type). The model location specifies the location of the model and if the model is an uploaded model or one of the default models.

Once the canvas has loaded, a script will initialize the WebGL stage, as well as load the portal frame and the grid into the scene.

Then, once the page has loaded, an AJAX call is made to getWorld.php script in the worlds folder that will, after validating the inputs, return the contents of the world file. The world file is parsed into the world variable and the items, in particular, is copied into the items variable.

$.ajax({

Type: "POST",

url: "worlds/getWorld.php",

data: {

wold:readWorldName(),

pass:readWorldPass()

},

async: false,

success: function(value){

world = JSON.parse(value);

items = world["objects"];

}

});

The function then populates the world by passing each item into the function downloadFBX(model location, model name, x-position, y-position, z-position, x-rotation, y-rotation, z-rotation, x-scale, y-scale, z-scale, action, url, type).

This function loads the FBX file into ThreeJS environment and places a link to the object into a data structure which is added to the array models. The data structure is as follows :

newObj = {

name: name,

object: object,

script: script,

url: url,

type: type

}

This model is primarily to simplify the process of keeping track of the active models. This is further augmented by the addition of a button to the element with the ID panel1, the control panel corresponding to the Objects tab. This button calls the setModel(model number) function which sets the currently controlled object to the object within the object at the index provided within the function. More details in the relevant section.

### 3D Editor

The 3D editor is in two parts. The first part is the click and drag functionality provided through WebGL and ThreeJS. When a mouse-up function is detected on the canvas, the currently controlled object has its position, scale and rotation data read and used to update the relevant fields in the Active Object panel. This is done using the updateActive() function.

The Active Objects panel works slightly differently. The changes are not automatically carried over, the user must click the Update Object button to call the updateObject() function which works similarly but in reverse.

The Active Objects panel, however also allows the user to view and change the script tag and the data connected to it.

It also allows the user to delete the active object. Clicking the Delete Object button calls the deleteActive() function which will use the JavaScript splice function to remove the object from the items and models arrays as well as the WebGL scene. It then clears the contents of the Objects panel and reloads it with the new list. It then renders the scene again to update what is visible by the user.

### Control Panels

The control panels are controlled fairly simply by the selectPanel(panel number) function. Each tab each panel is given an ID that follows a specific format (tab followed by the panel number for the tabs and panel followed by the panel number for the panels). The function simply hides all the elements with the panelButton or controlPanel class and then shows the elements with the IDs matching tab[panel number] and panel[panel number].

#### Active Object

As detailed in the 3D editor subsection above, this tab contains a number of form elements that use javascript to update the ThreeJS environment and the world file.

#### Objects

Objects tab is populated by the javascript component. Clicking on the button calls the function setModel(model number), which updates the active model variable, attaches the controls to the model in the ThreeJS environment and updates the Active Object tab.

#### Assets

The PHP script generates buttons that allow loading the default assets.

#### Upload

In addition to the buttons generated by the PHP script to show the uploaded assets, this panel also contains a simple HTML form to upload new assets. In addition to the visible file upload and submit sections, there is a hidden field that passes along the world ID to the upload.php page which handles the uploads. This is expanded upon below in the Upload subsection of User Assets.

#### Controls

A simple panel using plain HTML to list the controls of the ThreeJS editor.

### Save Button

The save button simply calls the function generateSave() on a click. This parses the models array and rearranges the relevant data into the format used by the JSON world files this the updates the objects array in the world object before an AJAX call is made to the updateWorld.php page in the worlds folder. This process is near-identical to the process that is described in the World Details Editing subsection of the User Page section above.

## User Assets

### Uploading

As described in the Upload page description, the world file is sent to upload.php. This page first validates the input, ensuring that it is of the correct filetype and size before copying it to the fbx folder in the world folder in the Theater folder the full location is as follows:

AssetBundleBuilder/Assets/Theater/[world id]/fbx/[file]

Upon loading the page, it also sends an AJAX request to createAssetBundle.php in the AssetBundleBuilder folder, sending the world ID at the same time. This script, in turn, runs a bash command to run the unity AssetBundle builder script to update the unity asset bundle stored in the respective folder.

Troubleshooting note: Open the folder AssetBundleBuilder as a unity project at least once and close out of it before running the web app. This ensures that unity will recognise the project when it is run from the command line.

### Downloading

Downloading the script from the mobile app is done via an AJAX request to dlab.php. This request includes the world ID and the world password. After the inputs have been validated, the password is checked against the password stored in the world JSON file in the worlds folder.

If it checks out, then a short PHP script is used to add the headers necessary to allow the AJAX request to download the file via an octet-stream file transfer.

If not, an error message JSON string is returned.