AWS Makerspace

2024 Capstone Project

CLEMSON MAKERSPACE

Table of Contents

Introduction

Team Bios

Duncan Hogg - Team Lead

Senior Computer Science student focusing on Cyber Security. Implemented the backend api and spearheaded tiger training integration.

Noah Broome

Senior Computer Science student. Developed forms and dashboards for the updated Makerspace site.

Caleb Murphy

Senior Computer Science student focusing on Cloud Engineering. Handled AWS environment services and pipeline management.

Partners & Stakeholders

CLEMSON® UNIVERSITY MAKERSPACE

aws

“Clemson's hub for hands-on projects and community” offering free usage of various equipment like 3D printers, laser engravers, and more for creative minds.

Amazon Web Services (AWS) is a cloud computing platform offering a wide range of services enabling businesses to scale, innovate, and operate more efficiently.

Project Introduction

This project is an ongoing collaboration between Clemson University Makerspace and AWS, focused on enhancing the experience for visitors and staff by leveraging AWS solutions. Throughout the semester, we worked closely with the Makerspace team and AWS experts to identify the best solutions for their unique needs, focusing on three key areas for improvement: the sign-in process, data collection, and data storage.

Currently, the sign-in process is slow and repetitive, requiring visitors to fill out redundant information with each visit. Data collection is fragmented across multiple forms, making it difficult to manage and track. Additionally, the Makerspace relies on Excel for data storage, which has led to instances of data loss and misplaced files in the past. Our solution aims to address these challenges by creating a centralized, efficient, and secure data management platform.

Project Description

Our solution transforms the Clemson University Makerspace website into a centralized hub for data collection and management. With this new platform, staff can eliminate the need for multiple, disconnected data sources or manual processes to track visitor information and access safety training records.

Sign-in process improvements: The new system features an integrated RFID scanner that enables visitors to sign in by simply scanning their Clemson ID, making the sign-in process faster and more convenient.

Data collection: All equipment logs and registration forms are accessible in one convenient location on the Makerspace website, simplifying the experience for visitors. Additionally, direct links to Clemson University's Tiger Training platform will allow anyone with a Clemson ID to easily access required Makerspace training courses.

Data storage: The Makerspace also benefits from centralized data storage through AWS, providing reliable, long-term storage to prevent data loss and ensure that records are preserved.

By consolidating these resources and processes, this solution creates a safer and more user-friendly environment in the Makerspace.

Project Scope

Amazon Web Services

A portion of the project scope focused on optimizing and updating the existing architecture in AWS for the Makerspace Student Management System. This consisted of a comprehensive audit and cleanup of the previously provisioned resources and a realignment of the architecture for current operational needs.

Tiger Training

A portion of the project scope focused on the integration of a new service called Tiger Training, which will be used to manage and house all student certifications and training records for equipment such as 3D printers and laser cutters. This enhancement aims to streamline the certification process and make it more accessible for both students and administrators.

Project Scope

CU Makerspace

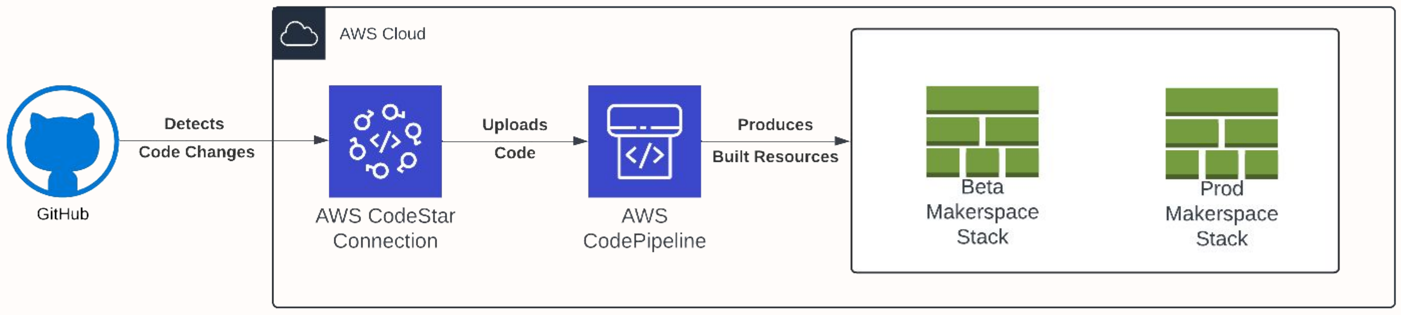
A portion of the project scope focused on improving the user experience in the Makerspace. This included the development and implementation of an RFID phone scanning system and the reworking of the data collection process. The phone scanner allows students to log their visits with a simple tap of their phone, creating a seamless check-in process. The original collection processes required repetitive information to be inputted each time a user used the Makerspace; the new collection process removed these inefficiencies. The data is recorded and stored in AWS, supporting real-time access and reporting through the existing student website interface, displayed on a tablet at the entrance of the Makerspace.

Milestone Timeline & Roadmap

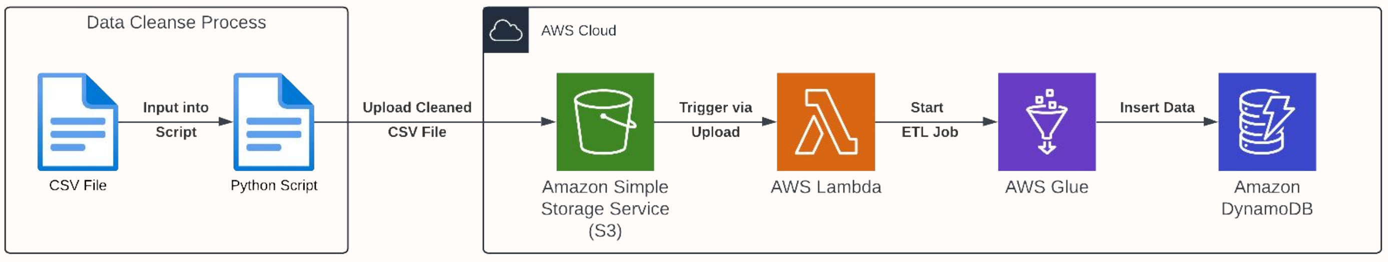
Milestone #1 8/25-9/19	• Familiarize with architecture • Obtain project requirements • Research current services and workflows
Milestone #2 9/20-10/10	• Sieve, clean up, and simplify architecture and data processing • Terminate unused resources • Research improvements to workflows and services
Milestone #3 10/10-11/14	• Implement improved workflows • Integrate Tiger Training • Centralize data storage • Migrate old data into new system • Data collection moved to site console
Milestone #4 11/15-11/28	• Integration with build pipeline • Documentation, testing, and finalized edits • Final report preparations • Last meetings with teams
Milestone #5 11/29-12/13	• Finalized report • Present project at End of Semester Event • Celebrate!

Architecture Overview

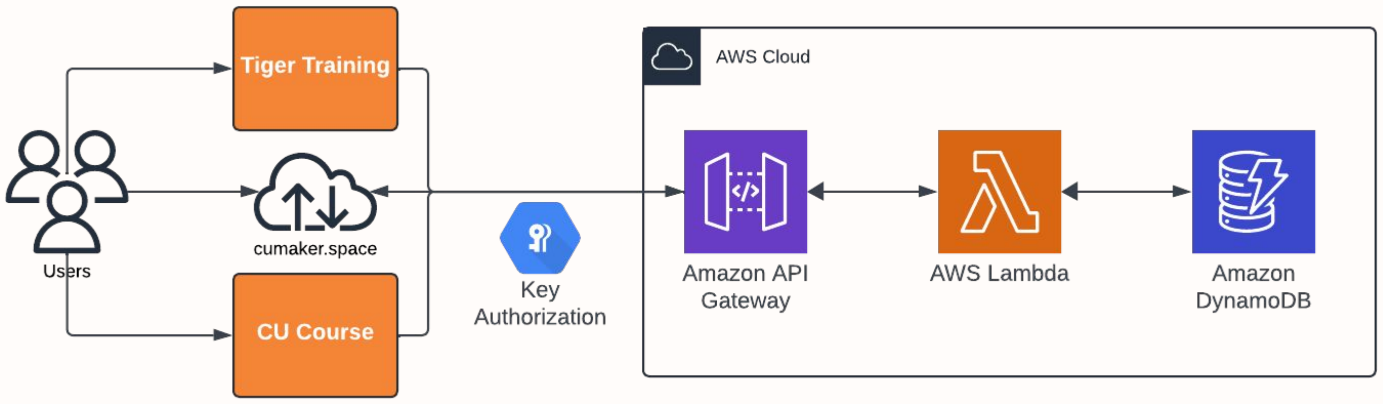
Pipeline



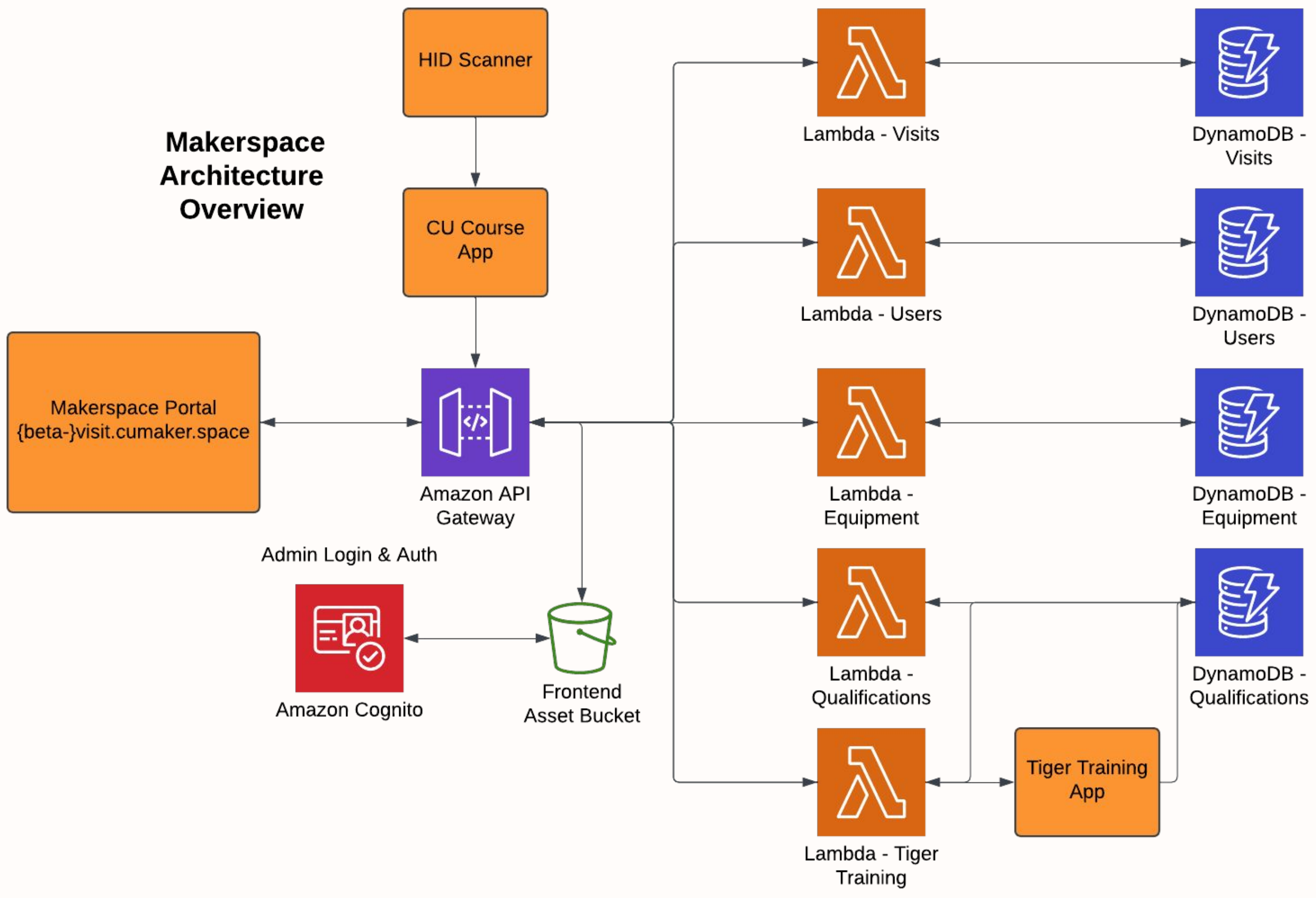
Data Migration



Workflow Overview



Architecture Overview



Developer Guides

Account Setup & Requirements

AWS

- Administrator permissions for Beta environment
- Read permissions for Production environment

GitHub

- Full read/write permissions; Obtain from an AWS Mentor

Tiger Training (Optional)

- To be able to create courses/programs: Author role account
- To use the api to manage courses/programs: Manager role api key/secret
- Optional due to existing api key/secret and the promise that Makerspace personnel will manage the created courses and program

CU Course (Optional)

- To be able to manage sessions - be in the Makerspace group
- Optional due to self-management of sessions implemented by Dana and the promise that Makerspace personnel will manage preparing the desktop and scanner

General Requirements / Recommendations:

- Test existing solutions and integration as early as possible
- Communicate often, and communicate to mutually understand
- When being requested for implementing new features, obtain a documentation of explicitly defined requirements, and ensure both parties understand what will need to happen to meet these requirements.

Tiger Training Data Retrieval

Purpose:

To retrieve user completion states for courses related to the Makerspace's trainings, waivers, and other miscellaneous items, and store this information in the AWS environment.

Components:

- AWS Lambda
- Clemson Tiger Training Author Account
  - Only necessary to use a GUI to create programs/courses
- Clemson Tiger Training Api Access (Manager level)
  - Requires obtaining values relating to a key and a secret
- Clemson Tiger Training Program (and courses)

Definitions:

- Course:
  - A set of related content and or questions to answer.
- Program:
  - A collection of courses.
- Learner:
  - A user who can register for courses and programs.
- Author:
  - A user that can create courses and programs.

References:

- Bridgeapp Api Documentation

Setting Up:

- See Creating Lambda Functions
    - No special permissions are required
    - Code located in cdk/visits/lambda_code/tiger_training
- Create a Secrets Manager secret “BridgeLMSApiSecrets”
    - See AWS Secrets Manager - Creating a Secret

      Key: value to store (replace <> with appropriate value)

      “key”: <Bridge api key>

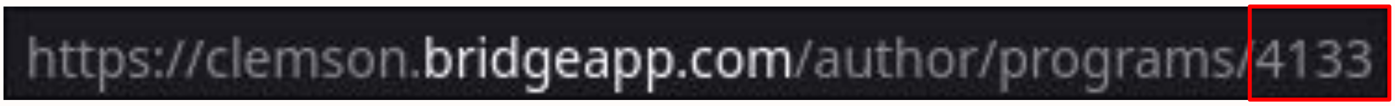
      “secret”: <Bridge api secret>

Lambda Workflow:

1) Generate the Bridge LMS api authorization token
    i) See “API Overview: Using an API Key and Secret or Token"
    ii) Key and secret values for the existing api access are stored in Secrets Manager under “BridgeLMSApiSecrets”
2) Get all of the course ids from a program
    i) See “Authors - Programs”
3) For each course, get all enrolled learners
    i) See “Authors - Course Templates”
    ii) Depending on implementation, for each call of this lambda, save and reuse the most recent “updated_at” timestamp to reduce the amount of data to parse.
4) Parse courses for all learners that have a “complete” state.
5) For each learner, save all completed courses in the AWS environment (DynamoDB)
    i) For ease of use, the implementation as of Fall 2024 sends the data to the backend api and lets the backend api manage storing the data.

Setting Up Tiger Training:

1) View the User Guide - Tiger Training to get started
   a) In addition to emailing Mary-Kate regarding "Author" role, an api key and secret value pair with "Manager" permissions will be required. The "Manager" role has access to the program endpoint while the "Author" role for the api does not.
2) Obtain a program id
   a) Navigate to the program that will be used
      i) On the left sidebar menu, select "Author > Programs"
      ii) Search for the name of the program
      iii) Click on the program
   b) The id of the program will be the number at the end of the url
      - For example, the following program's id is 4133


https://clemson.bridgeapp.com/author/programs/4133

Notes:

The "Manager" role has more permissions for what the intended purposes are. Unfortunately, this cannot be rectified at this time due to the implementation of permission management in Bridge LMS. As such, there is a mutual trust between developers using this key and the Tiger Training Administration that this key will not be abused. Failure to comply will likely end in the termination of the key, and prevention of access to the system in the future.

To see implementation details and comments, please navigate to the appropriate lambda using the AWS gui console, or view the handler code here.

Front End – Makerspace Console

The site is built using modern web development tools such as:

React - a popular JavaScript library to build user interfaces
https://react.dev/learn

TypeScript - A superset of JavaScript that has static type checking
https://www.typescriptlang.org/docs/

Bootstrap - a frontend framework with pre-built styling components
https://getbootstrap.com/docs/5.3/getting-started/introduction/

Vite - as the build tool https://vite.dev/guide/

Node.js - for the JavaScript runtime and package manager
https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

Development

This project requires Node.js version 14 or greater to be installed locally on your machine.

1) Navigate to the /site/visitor-console/ and run ‘npm install’ to install all required dependencies
2) To start the local development server run ‘npm run start’ and the site should be accessible at http://localhost:3000/. If you are unable to view the site at that address check the output of the command to see where it can found.

Building and Dev Deployment

1) Navigate to the /site/visitor-console/ and run ‘npm install’ to install all required dependencies
2) To make a production build run npm run build
   i) To make the frontend hit the desired api, set the environment variables mentioned below in a .env file
   ii) This will create a folder called dist with all of the static files ready to serve.
   iii) You can preview what the production deployment will look like by running npm run preview after building the app.

Environment Variables

VITE_API_ENDPOINT (default: https://api.cumaker.space) - Will determine what api will be hit when making requests from the frontend. Note, there should not be a trailing slash as the routes will be appended to this endpoint with a prefixed slash. If you are deploying to your Dev stack, you should use the endpoint provided from the SharedApiGateway that gets stood up.

VITE_BACKEND_KEY - This will come from a .env that is created during the pipeline process or a .env.local file during local development. This allows API that require a key to add the X-Api-Key header.

Amplify Config - The user pool ID and user pool client ID are required for the admin logins. These can be found in the AWS console and updated in the getAmplifyConfig.tsx file.

Form Constants

These are located in /lib/constants.ts and can be changed to alter the options on form selections. Everything is an array of strings or a JSON object. For example the list of Makerspace Interns can be updated by adding or removing names from the “interns” array.

Project Structure

All frontend files are located in the unified-makerspace repository on GitHub under site/visitor-console.

1) **/public** - Contains static that will be served as is such as images, fonts, favicons or any other static files.
2) **/src** - Contains all source code for the site.
    a) **/assets** - Contains reusable assets such as images for the background and Makerspace logo.
    b) **/components** - Contains reusable React components.
        - **/EquipmentFormStages** - contains the individual stages for the equipment form.
    c) **/config** - Stores configuration files and environment specific settings for the application.
        - **getAmplifyConfig.ts** - This file makes the connection between Amplify and Cognito on the AWS backend.
    d) **/library** - Contains utility functions and other shared logic.
    e) **/pages** - Contains all main pages for the site.
    f) **/styles** - Contains customized stylesheets that will be utilized by Bootstrap.
    g) **main.tsx** - This is the starting point for the project that renders the background for all pages.
3) **/node_modules** - Contains all required dependencies for the project installed by npm. Generally this can be left alone.
4) Miscellaneous configuration files

Page routing

Page routing is handled inside of App.tsx using React Router.
https://reactrouter.com/home

To create a route you will use a Route component to specify the page and path as follows.

<Route path=”/[route]” element=”{<[Page element] />}”

Forms

All forms utilize React Hook Form components and Yup for schema-based validation.

https://react-hook-form.com/get-started

https://github.com/jquense/yup

To modify answer choices for the forms such as locations, equipment types, or printer names update the corresponding const inside of /library/constants.ts.

1) **EquipmentForm.tsx** - This is a multistage form which collects equipment usage information. Response data is stored in the Equipment Usage DynamoDB table using the /equipment API endpoint POST request. The following stages are located in /components/EquipmentFormStages.
   a) InitialInfo.tsx
   b) ProjectDetails.tsx
   c) Survey.tsx
2) **RegistrationForm.tsx** - This form collects additional user information such as class status and major. Response data is stored in the User Information DynamoDB table using the /users API endpoint POST request.

Dashboard pages

Dashboard pages fetch and display information from DynamoDB so that it is accessible to interns on the site.

1) **EquipmentUsage.tsx** - This dashboard allows interns to view equipment usage logs retrieved from the Equipment Usage DynamoDB table using the /equipment API endpoint GET request. This dashboard also provides the ability to update the status of 3D prints using the EditModal component and /equipment/{username} API endpoint PATCH request.
    a) EditModal.tsx
2) **Visits.tsx** - This dashboard allows interns to view Makerspace visits from all locations retrieved from the Visits DynamoDB table using the /visits API endpoint GET request.
3) **Qualifications.tsx** - This dashboard allows interns to view completed trainings and waivers for a specific user retrieved from the Qualifications DynamoDB table using the /qualifications/{username} API endpoint GET request. It also has a refresh button which will fetch information from Tiger Training to update the Qualifications table in DynamoDB.

Back End – Data Management API

Purpose:

To provide a simple interface to manage storing, updating, and retrieving data collected by the Makerspace.

Components:

- AWS Lambda
- AWS API Gateway
    - API Key and Usage Plan
- AWS DynamoDB

References:

- Backend API Documentation

Setting Up Lambda Functions:

- Create an IAM role matching the following criteria:
    - Entity: AWS Lambda
    - AmazonAPIGatewayInvokeFullAccess Policy
    - AmazonDynamoDBFullAccess Policy
    - AWSLambdaBasicExecutionRole Policy
- Create a lambda function to handle requests for each of the major endpoints:
    - /users, /visits, /equipment, /qualifications
    - Use the created IAM role above for the permissions of all these lambda functions
    - Code centrally located at /cdk/api_gateway/lambda_code

Setting Up DynamoDB:

1) Create a DynamoDB table to store data for each of the major endpoints, matching the following criteria:
   a) Primary key: user_id (string value)
   b) On-Demand read/write capacity
   c) For all tables except the one for user information:
      - Sort key is a timestamp (string value) related field
      - Global Secondary Index (GSI) is a field named "_ignore" (string value) and the sort key is the same timestamp related field
      - Attribute projections for the GSI should be "Only keys" to minimize duplicated data

Notes:

The GSI is useful for the tables with a timestamp field – given the GSI primary key always contains the same value – as it allows to query the table entries by timestamp only. Without the GSI used in this manner, an exact user_id would be required when searching by timestamps. This exactness is impossible to use when desiring a search for all users.

Setting Up API Gateway:

1) Create a Rest API Gateway matching the backend api documentation (see References) and matching the following criteria:
   a) All methods use a Lambda Proxy for the integration request
      i) The lambda proxy is the corresponding lambda created earlier for each major endpoint
   b) Methods that require an api key, require an api key

Setting Up API Gateway Continued:

2) Deploy the api to a stage
3) Create an api key
    a) See Creating API Gateway - API Key Usage
    b) Name it "SharedApiGatewayAdminKey"
    c) Also Create a Secret with the same value used to create the api key. Name the main secret "SharedApiGatewayKey", and the key value should be "api_key": <key_value>
4) Create a usage plan
    a) Also see Creating API Gateway - API Key Usage
    b) Ensure the api has been deployed to a stage
    c) Name it "Shared Api Gateway Usage Plan"
    d) Associate the deployed stage with the plan
    e) Associate the created key above with the plan

Rest API Request Workflow:

1) Rest API request is made to the backend api
2) API Gateway receives the request and routes it to the appropriate lambda proxy for the requested endpoint
3) Endpoint proxy handles the request as per the documentation
    - Retrieve (GET) from the corresponding data table
    - Put/update (POST/PATCH) data to the corresponding data table
    - Delete (DELETE) data from the corresponding data table
4) Endpoint proxy returns a response body as per the documentation

To see implementation details and comments, please navigate to the appropriate lambda using the AWS gui console, or navigate to the appropriate handler code here.

Notes:

Regarding POST/PATCH requests, the current implementation is designed to allow for all key: value pairs in a request body to be inserted into the table. This is intentional design to meet the requirement of being capable of storing more data fields in the future. As such, to add more fields to the tables, add more fields to the request body. The only stipulation is when data validation / field requirements need to be checked. That is when explicitly defined fields and behavior needs to be added to the code. To make that process easier, each lambda handler provides a validateXRequestBody function (X is the name of the data type, e.g. "Visit" or "Equipment) where all logic of that sort can be found.

An additional note about the usage plan, any time the stack needs to be completely destroyed and rebuilt, the usage plan will need to manually be associated with the deployed stage(s) again. Along with this, for whatever reason, there exists a "prod" stage (we assume somewhere in the Pipeline stack memory) that always is deployed with the actual "Beta" and "Prod" stages. Sometimes, however, this "prod" stage takes precedence and is the actual stage that the CNAME record points to. As such, until this "prod" stage can be confirmed to not be redeployed, associate both "Beta"/"Prod" and the "prod" stage with the desired usage plan / api key to prevent authorization issues. We are not entirely sure why the "prod" stage is still deployed as it is nowhere to be found in the cdk code, but, as found in the guides regarding the CDK, we do NOT recommend deleting the Pipeline stack to find out.

Back End – Configuring AWS CDK Code

In this section, we will go over how to configure and setup the GitHub repository to have it locally on your machine as well as how to push the changes to the mainline branch in the Unified Makerspace repository.

First, please ensure you have completed the developer account setup in the **Developer Account Setup & Requirements** section prior to continuing.

Next, navigate to the Unified Makerspace GitHub repository and create a forked repository for your own changes. To create a forked repository, follow the instructions in the **Create a Forked Repository** section and then come back to this page once completed.

To add your changes to the forked repository, follow the following instructions for the configuration of **Git GUI** (not recommended), **GitHub Desktop**, or through **Git Bash**.

To compile code, refer to the section How to Compile AWS CDK Code

To merge your forked branch with the Makerspace branch, refer to the section Merge Forked Repository with Makerspace's Repository.

Create a Forked Repository:

1) Navigate to the Unified Makerspace repository in your browser.
2) Find the four buttons on the top right corner labeled **Edit Pins**, **Watch**, **Fork**, and **Star**.
3) Click the gray **Fork** button and then on the screen that pops up, click the green **Fork** button on the bottom right-hand side.
    a) The information on this screen does not need to be changed.
4) Once the repository has been forked, it will now be in your repositories in GitHub.
5) To locate this forked repository, follow the next section below. Otherwise, return to Back End - Configuring AWS CDK Code.

Locate New Forked Repository:

1) To locate the forked repository, click on your profile icon in the upper right corner.
2) Then, click **Your repositories** and you will see the repository you forked there.
3) Next, access that repository by clicking on it, as you will need this for the next instructions.
4) Return to Back End - Configuring AWS CDK Code.

Git GUI:

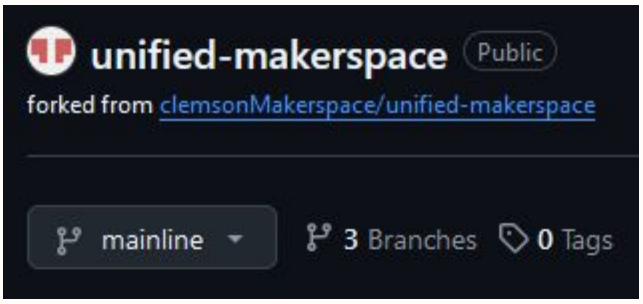
1) Download and open Git GUI
2) Click **Clone Existing Repository** and then navigate to the forked repository you recently created.
3) Once there, click the green **Code** button on the top right.
4) Copy the **HTTPS URL** and paste it into the **Source Location** text box in Git GUI
5) For the **Target Directory**, follow the instructions below:
    a) If you click browse, navigate to the **parent directory** you would like the folder to be created at, which stores the code from the repository. In other words, if you would like to store the code in a folder on your Desktop, navigate to the Desktop and click **Select Folder**. Then, click into the **Target Directory** textbox and add the following string, without quotes, to the end of the string currently there: *"/nameOfFolder"*
    b) If you know the file path you'd like to store the code, paste the file path into the **Destination Source** textbox. For example: *C:\Users\nameOfUser\Desktop\nameOfFolder*
6) Press **Clone**
7) Open the folder in your text editor and make your changes. Then, save your changes and navigate to Git GUI.
8) Prior to moving forward, ensure you are on the correct branch. To do so, go to the next page and continue the instructions.

Git GUI (continued):

1) To ensure you are on the correct branch, with Git GUI open, click **Branch**. Then, click **Checkout**.
2) Next, click the bubble selector for **Local Branch** and select the option **master**. Click **Checkout** on the bottom right of the popup.
3) Press **Rescan** (next to Commit Message textbox) and you should see the files you have made changes to.
4) From here, verify the changes you have made by scrolling through the code file and change files by clicking one of the files on the left-side. Once changes are verified, move on to the next step.
5) Next, click the files under **Unstaged Changes** (CTRL + Click if multiple files) and then click **Stage Changed**.
    a) This will add the files to the upcoming commit and push to the forked repository.
6) Add a message to the **Commit Message** textbox.
7) Click **Commit** and then click **Push**.
    a) From this popup when Push is clicked, you should see **master** under Source Branches. Leave other options default.
8) Click **Push** on the bottom right-hand corner of the popup. Once successful, close the popup.
9) Navigate back to your forked repository in your repositories on the GitHub website.
10) Continue onto the next page.

Git GUI (continued):

1) Once back in your forked repository, click the button that looks like the button on the bottom left of the image below:



2) There will be a branch in your forked repository labeled **master**. Navigate to this branch by clicking on it.
3) Next, you will notice that there will be a **Contribute** button in this branch. Click that button and click the green **Open pull request** button.
4) Configure the pull request's title and description there, if needed, and then verify you are merging the correct branch into the right branch. This will be above **Add a title**.
5) Once verified, click the green **Create pull request** button on the bottom right. Once the checks pass and there are no merge conflicts, click the green **Merge pull request** button. Then, click **Confirm merge**.
6) You have now added your changes to your forked repository!

To merge your forked repository with the Makerspace repository, refer to the Merge Forked Repository with Makerspace's Repository section.

GitHub Desktop:

1) Download and open GitHub Desktop and ensure you are logged in.
2) Click **File** in the top left corner and then click **Clone repository…**
3) Under the **GitHub.com** tab, you will see the forked repository you recently forked. Click this repository.
4) Then, where it states **Local path**, do one of the following:
    a) If you click **Choose…**, navigate to the **parent directory** you would like the folder to be created at, which stores the code from the repository. In other words, if you would like to store the code in a folder on your Desktop, navigate to the Desktop and click **Select Folder**. This will create the folder named the same as the repository containing the repository files on the Desktop.
    b) If you know the file path you'd like to store the code, paste the file path into the **Local path** textbox. For example: *C:\Users\nameOfUser\Desktop\nameOfFolder*
5) Next, with GitHub Desktop open, click the button (third from the left) labeled **Fetch origin.** This will obtain any changes from the forked repository, if any.
6) Next, open the created folder from earlier in any text editor and make your changes. Then, save your changes and open the GitHub Desktop application. You should see the change you made populate.
7) Continue on the next page.

GitHub Desktop (continued):

1) Now that you can see your changes, verify the changes are correct.
2) Next, add a **Summary (required)** on the bottom left corner of the GitHub Desktop application. This is the title of your commit to your forked repository.
3) Then, it is best practice to add a description of the changes you have made in the **Description** text box below the Summary text box.
4) Once those fields are filled, click the blue **Commit to *branchName*** button below the Description text box.
5) Next, the button that used to be labeled **Fetch origin** has turned into **Push origin**. Once you click **Push origin**, it will push all of the changes to the files to your forked repository.
6) You have now added your changes to your forked repository!

To merge your forked repository with the Makerspace repository, refer to the Merge Forked Repository with Makerspace's Repository section.

Git Bash:

1) Download and open Git Bash
2) In your browser, navigate to your forked repository in your GitHub repositories and click the repository. Then, click the green **Code** button and copy the HTTPS URL link.
3) In Git Bash, change your working directory to the **parent directory** you want the cloned repository folder to populate.
4) Once there, type the command, without quotes, "**git clone *Copied_HTTPS_URL_Link***". This will create the folder containing the repository.
5) Next, open the folder in your text editor and make your changes. Then, save your changes in the file(s).
6) Navigate back to Git Bash and ensure you are in the correct directory. You will know you are in the correct directory if you see the following in Git Bash (Windows):
    a) *UsersName*@DESKTOP-*RandomString* MINGW64 ~/Desktop/*nameOfRepository* (*nameOfBranch*)
7) Once in the correct directory, type the following command, without quotes, "**git status**". This command will show you the modified files with changes. Verify these files are correct.
    a) If the files are not correct or not all there, you may be in the wrong directory.
8) Next, type the following command, without quotes, "**git add** *". This command will add **all** of the changed files to the upcoming commit and push.
9) Continue onto the next page.

Git Bash (continued):

1) Now that the files have been added, we can move on to the next step.
2) Next, type the following command, without double quotes, "**git commit -m '*Title of Commit*' -m '*Description of Commit*'**". Change the italics in the command to a descriptive title and description of what was changed. This command will commit the changed files and prepare them to be pushed to your forked repository.
3) Next, type the following command, without quotes, "**git push**". This command will push the files that you added and committed to your forked repository. To view these changes, navigate to your forked repository.
4) You have now added your changes to your forked repository!

To merge your forked repository with the Makerspace repository, refer to the Merge Forked Repository with Makerspace's Repository section.

Merge Forked Repository with Makerspace's Repository:

1) Navigate to your forked repository that you created earlier.
2) Click on the button labeled **Contribute** and then click the green button labeled **Open pull request**.
3) Then, edit the title and description to what you would like; apply best practices and provide descriptive titles and descriptions as to what you have changed.
4) Prior to moving forward, on the right-hand side of the screen, there will be a section labeled **Reviewers**. Ensure you select multiple reviewers to validate your code before **EACH** merge with the Makerspace's repository.
5) Next, click the green button labeled **Create pull request.** Once the checks pass, there are no merge conflicts, and the reviewers have reviewed your code, you will be able to click the green **Merge pull request** button. Then, click **Confirm merge**.
6) You have now successfully merged your forked repository with the Makerspace's repository!

Back End – AWS CDK File Structure

All AWS CDK files are located in the unified-makerspace repository on GitHub under **cdk/**. To access these stacks, access the Beta Environment and go to the AWS CloudFormation service.

1) **cdk/api_gateway** - Contains all files for the creation of the SharedApiGateway & BackendApi Stacks
    a) **api_gateway/lambda_code** - Contains all Lambda function's code files pertaining to API GW
2) **cdk/cognito** - Contains the setup files to setup AWS Cognito
3) **cdk/data_migration** - Contains setup files for Data Migration Stack (only needed to be used once for initial migration)
    a) **data_migration/lambda_code** - Contains Lambda function code pertaining to Data Migration Stack
4) **cdk/database** - Contains all files for the creation of the Database Stack & setup of DynamoDB tables
5) **cdk/dns** - Contains all files for the creation of the MakerspaceDns & MakerspaceDnsRecords Stacks & Domains class
6) **cdk/visit** - Contains all files for setting up the Visit Stack
7) **cdk/accounts_config.py** - Contains account information
8) **cdk/app.py** - Where application starts, makes App and Pipeline
9) **cdk/cdk.json** -
10) **cdk/makerspace.py** - Contains code to build each Makerspace Stage and the creation of the Makerspace Stack

1) **cdk/Pipeline.py** - Contains code to build the Pipeline in AWS Code Pipeline
2) **cdk/requirements.txt** - Contains the packages that will be downloaded & installed to each environment when the Pipeline is ran
3) **cdk/setup.py** - Documentation for this file
4) **cdk/source.bat** - Contains code to setup Python virtualenv on Windows

Back End – How AWS CDK Code Works

In this section, we will go over the workflow of where the code starts and how it goes through each step in the process to provision and build all of the necessary resources required for the Beta and Production environments. This workflow starts after a merge or commit to the Makerspace's repository commences.

Merge or commit is made to Makerspace's repository:

1) **Start in app.py**
    a) Creates AWS App object and Pipeline class instance object
        i) When pipeline object is created, we shift over to Pipeline.py
2) **Pipeline.py**:
    a) Creates an object to store the CodeStar Connection
        i) This CodeStar Connection is an active listener on the mainline branch in the Makerspace Repository. When changes are made to it, it will pick up the changes and run the Pipeline with the new changes

References:

- AWS CDK V2 Documentation
- App Object Documentation
- Pipelines Module Documentation

3) **Pipeline.py (cont.)**:
   a) Next, we will create our CDK Shell Step object
      i) This object stores all of the commands that will run when the pipeline is ran in both Beta & Prod and is ran in the UpdatePipeline pipeline stage in AWS CodePipeline.
   b) Next, we create our pipeline object
      i) This utilizes the CDK Shell Step Object and allows cross account keys to be used
   c) Next, we create the Beta stage object in the pipeline
      i) This object is created from the MakerspaceStage class in cdk/makerspace.py file, which we will switch to now
4) **makerspace.py**:
   a) In this file, we start in the MakerspaceStage class
      i) This class then creates a MakerspaceStack object
   b) In the MakerspaceStack class, we create all resources associated with the environment.
      i) We first create all of the domain strings, from the Domains class in cdk/dns, and return with them created in the domains object.
      ii) Next, we create all of the public hosted zones with their associated domains, which creates the MakerspaceDns Stack in cdk/dns
      iii) Next, we create and configure the database tables and Database Stack in cdk/database

4) **makerspace.py (cont.)**:
    iv) Next, we create the Visit Stack, which creates the artifact source bucket and CloudFront Distribution in cdk/visit
        (1) When creating the CloudFront Distribution, if you ever run into the error of the AWS ACM Certificates remaining in the **Pending Validation** status, which shouldn't take longer than 5 minutes, then check the Gandi.net domain name registrar and ensure that the main domain's (cumaker.space) sub-domains (beta-visit, beta-api) NS records are pointing to the correct Namespace servers. To see which Namespace servers they should be pointing to, view the Route53 Hosted Zones for the associated domain name and select the radial selector. From there, click **View Details** and then you will see the correct NS record Namespace services it should be pointing to under the **Value/Route traffic to** column. Then, change the NS records in Gandi.net to point to the Namespace servers provided in the Route53 hosted zone.
            (a) A member of the Makerspace will have access to the Gandi.net account. Coordinate with them to make these changes.

4) **makerspace.py (cont.)**:
   v) Next, we will configure the Cognito setup
      (1) This will be utilized by the interns under their admin account in the user group, which will automatically be created in cdk/cognito
      (2) The setup configures the User Pool and the App Client required for the Cognito setup
   vi) Next, we will create and configure the BackendApi Stack
      (1) In this Stack, located in cdk/api_gateway, we create the required Lambda functions to support the API, its API calls, and what it should do with each call. Additionally, we give the Lambda functions permissions to be invoked by the API.
   vii) Next, we create and configure the SharedApiGateway Stack
      (1) In this Stack, located in cdk/api_gateway, we create the REST API and configure all of the endpoints. Additionally, we deploy the API stage(s) and create a usage plan for the API.
   viii) If not in a dev stage (which you shouldn't ever be in a dev stage - we removed this), then it will create the MakerspaceDnsRecords Stack.
      (1) In this Stack, located cdk/dns, we create the A records for any sub-domain we are supporting. Currently, we only have visit.cumaker.space and api.cumaker.space setup.

4) **makerspace.py (cont.)**:
        i) Lastly, we grant permissions for the Lambda functions utilized by the BackendApi to access the various DynamoDB tables
    b) Now that all of the Stacks and resources have been created that are required for the stage (Beta/Prod), we go back to the Pipeline.py file to complete the process.
5) **Pipeline.py (cont.)**:
    a) Now we have walked through what happens when the Beta stage is created, we will continue.
    b) Next, we want to test the the Beta stage's CloudFront Distribution (CFD) URL endpoint by curling the beta-visit.cumaker.space endpoint
        i) This ensures that the endpoint ties back to the CFD and we can access the sub-domain beta-visit.
    c) Next, we want to test the Beta API Endpoints by running a script and if all of the tests pass, then the creation of the pipeline will continue.
    d) Next, we will create all of the resources needed for the Prod stage by following the process discussed here in makerspace.py.
        i) In AWS CodePipeline, in the management console, you have to access the pipeline and Accept the stage **PromoteBetaToProd** for the Prod stage to actually run. If not, the pipeline will remain in the **InProgress** status until then. Reject to stop pipeline from staying in InProgress status. Beta will still deploy.
    e) Then, we will repeat steps b) and c), but for the Prod stage
    f) Next, we go back to the app.py file

6) **app.py (cont.)**:
   a) Once the pipeline object is complete and the entire pipeline is done, then we will synth the application at the bottom of this file
   b) You have now successfully ran the pipeline and built all of the necessary resources needed to operate!

For how each stack works, please refer to the Unified Makerspace repository and view the comments in the desired file to learn more.

Backend – Common Errors with Pipeline

If at any point you must tear down the stacks in CloudFormation to rebuild the entire environment, it will take all resources associated with each stack with it. **DO NOT DELETE THE PIPELINE STACK**. The only resources that will be retained are S3 buckets created by the Pipeline with a deletion policy (the artifact bucket for holding website files) and DynamoDB tables with termination protection turned on and with a retain deletion policy.

Therefore, when the environment is rebuilt through the Pipeline after all resources have been deleted, you **WILL** need to change where the namespace servers on Gandi.net (where the main domain, cumaker.space, is hosted) are pointing. These namespace servers are found in Route53, click any given public hosted zone, and look at the records down below. You have to change the NS records on Gandi.net to be the exact numbers provided in the A record under the different public hosted zones.

For example, if you're stuck on creating the certificates for beta-visit.cumaker.space, then the problem is that your NS records in Gandi.net for all beta-visit subdomain NS records are not matching. Once you change the record information in Gandi.net, the certificates should become validated after a couple of minutes. Make the appropriate changes to each public hosted zone you plan on using (typically beta-visit and beta-api).

Another common issue that you may encounter is if you delete the stacks and have to recreate them, then the Cognito setup will be completely different.

Why is this important? When the interns over at the Makerspace attempt to login to the site through the Admin portal, they are required to use a username and password that is in the User Pool that is created through the CognitoConstruct (in makerspace.py). Unfortunately, there is not a way to make these credentials automatically currently implemented in the Pipeline. As long as the stack that contains the CognitoConstruct resources is not deleted, then the credentials will remain. If the stack must be removed, then recreate the stack through the Pipeline, and then access the management console, then Cognito, and access the new User Pool. In this User Pool, click "Users" on the left side under "User Management", then click "Create user". In there, you can leave everything as default, add the username "admin", and create a password that must be shared with the Makerspace. The Makerspace will manage this account information on their side. You are now good to go!

Another common issue that is faced is when something changes in the DyanmoDB table structure (i.e., partition key needs to be changed), you will be required to recreate ALL of the Stacks, which will then require the Certificates in AWS Certificate Manager to be recreated, which will require the Gandi.net subdomain records to be changed as well. It’s like a chain reaction, as some of the stacks use linked resources in other stacks, and so on. That is why if you attempt to delete the Database stacks, then it will state that the delete has ran into an error because of resources that are tied to the BackendAPI stack cannot be deleted, and so on.

Another common issue that is faced is if you attempt to recreate resources that should be used by the pipeline manually, it will not work, as in the backend of AWS when the pipeline is ran does a lot of resource tying and whatnot. Therefore, if you accidentally delete a DynamoDB table and must recreate it with the same exact name it had before, it will not work. You would then have to rerun the Pipeline and **hope** that it can rebuild the resource. If not, you will have to delete the stacks, and so on (as described in the paragraph above).

BackEnd – How to Compile AWS CDK Code

In this section, we will go over how to compile the AWS CDK code after we have forked the repository and made the desired changes to the code.

1) First, ensure you are in the cdk/ folder
2) Next, run one of the following commands:
    a) python app.py
    b) python3 app.py
        i) This will compile the entire environment and determine if you have any bugs in your code prior to merging with the Unified Makerspace Repository
3) If no errors are found, then you can proceed with the merge.
4) If errors are found, debug the errors and then repeat Step 2.

Why do we want to do this?

When the forked branch is merged to the Unified Makerspace Repository, it will run the pipeline with the code added to this repository. Then, in the Build stage in AWS CodePipeline, it will compile the code again and if there are any errors, it would waste time waiting for the logs to produce the errors. Instead, this saves you time in development and decreases the risk of breaking the current resources in the Beta environment.

BackEnd – Data Migration Process

In this section, we will go over how the Data Migration process works and how we used it. This process should only be ran once through the pipeline, as if there are a lot of records in the data you are attempting to migrate, it will be quite costly. Be mindful of what you're migrating and ensure to only migrate data on the last planned time of pushing the environment to Production, as Database tables may change names and the data you've migrated could be no longer used if pushed to Prod again. Here is how we used it:

1. First, we have an S3 bucket in the Beta environment called "csvs-for-glue-job" that contains the file(s) we want to migrate into production. Add your csv file to this bucket and decide to keep what is in there (if needed to be migrated again for any reason; i.e., database names have changed) or remove it.
2. Next, check makerspace.py to ensure that the DataMigration stack will be created properly. The code for this stack is located in cdk/data_migration/__init__.py.
3. The code for the DataMigration stack should take any files that are in the S3 bucket and upload them to their corresponding database tables in DyanmoDB. To find out what data goes where, view the file glue_script.py in cdk/data_migration. This script is the AWS Glue ETL Job that is ran, which takes certain columns of data and places them where they belong.

1. If you need the data to go to other columns or change the entire script due to having a different dataset needing to be migrated, you can do so in the glue_script.py file.

How the Data Migration Works:

1. General workflow: Uploaded files in the S3 bucket (csvs-for-glue-job) will be taken from a lambda function that gets created via the Pipeline and used to start the AWS Glue ETL Job.
2. In depth, we start in makerspace.py, where the DataMigration stack gets created when we push to Prod.
3. Once in Prod and the DataMigration stack is created, we now enter the file cdk/data_migration/__init__.py. Here, we create some variables to hold some important information that will be utilized later (beta_account_id/etc.).
4. Next, we create a Temp bucket to store Glue temporary data
5. Next, we grant Glue some permissions to be able to access DynamoDB, along with the standard service role.
6. Next, we add additional permissions that grant Glue to access the file stored in the Beta S3 Bucket
7. Then, we obtain the Glue script (glue_script.py) and use it as an inline script, utilized later.
8. Then, we create and define the AWS Glue ETL Job with information obtained earlier.
9. Next, we create the Lambda function with the code from cdk/data_migration/lambda_code/data_migration.py and give it permissions to be able to start the Glue Job.
10. Next, we grant the Glue Role permissions to the Temp bucket created earlier.

11. Next, we create the cross_account_policy to allow Prod to access Beta S3 Bucket
12. Then, we apply the bucket policy using a custom resource.
13. You now have the DataMigration stack created!

To view the files used for the Data cleansing and what file we created to be used in the official data migration, view the Google Drive that we will share with you during the transition (in the DataMigration Contents folder).

If the folder is not shared to you in time, please reach out to one of the AWS + CU Makerspace members from Fall 2024 (listed on Contacts page at the end of this document).

Creating AWS Services

Creating Lambda Functions:

1) Navigate to the Lambda page within your AWS AdminAccess Account
2) Click “Create Function” in the top right corner
   a) Provide a meaningful name for the function
   b) Use the dropdown to select your runtime environment of choice
   c) Unless explicitly needed, use the default x86_64 architecture
   d) Unless other permissions are needed, use the default LambdaBasicExecutionRole role
   e) Unless other additional configurations are needed, use the default additional configurations
   f) Click “Create Function” in the bottom right corner to finish creating the lambda function
3) Paste code into the “Code Source” area
   a) Ensure the entry point into the pasted code follows the selected environment’s handler function name
   b) For example, in Python, the default handler should look like:
      def lambda_handler(event, context):

Creating IAM Role:

1) Navigate to the IAM page within your AWS AdminAccess Account
2) Navigate to the roles page by clicking “Access Management > Roles” in the left sidebar navigation menu
3) Click “Create role” in the top right corner
    a) Choose the trusted entity (AWS service)
    b) Choose the service that will be given permissions
    c) Search for the policy or policies to allow or deny the effects a service with this role can do
    d) Click on the checkbox to the left of the policy to prepare it to be added to the role
    e) After selecting all the policies to add to the role, keep the default permissions boundary settings and click “Next” in the bottom right
    f) Provide a meaningful name for the role
    g) Optionally provide a description
    h) Review the trusted entities and permissions that will be added
    i) Click “Create role” in the bottom right once everything is as expected

Creating DynamoDB:

1) Navigate to the DynamoDB page within your AWS AdminAccess Account
2) Navigate to the tables page by clicking “Tables” in the left sidebar navigation menu

Creating DynamoDB Continued:

3) Click “Create table” in the top right corner
   a) Provide a meaningful name for the table
   b) Provide the name of the partition (primary) key field for the table, and select the type of data it will be
   c) Optionally provide the name of the sort key field and select its data type
   d) Click “Customize Settings” in “Table Settings”
   e) Keep the default “DynamoDB Standard” table class
   f) Choose the appropriate read/write capacity
   g) Keep the default “Warm throughput” settings
   h) Optionally create a global index (global secondary index - GSI)
      i) Provide name of the secondary partition (primary) key field and select its data type
      ii) Optionally provide the name of the sort key field and select its data type
      iii) Choose the appropriate attribute projections (what fields will be duplicated from the table)
      iv) Keep the default “Warm throughput” settings
   i) Keep everything else with the default values
   j) Click “Create table” at the bottom right corner once everything matches expected criteria

Creating API Gateway - Rest API:

1) Navigate to the API Gateway page within your AWS AdminAccess Account
2) Click “Create API” in the top right corner
    a) Scroll down to “Rest API”
        i) Select “Import” if an OpenAPI yaml or json documentation file is available (continue from Part 5 if this option is chosen)
        ii) Select “Build” to make the API manually (continue reading for further instructions)
    b) Provide a meaningful name for the api
    c) Provide an optional description
    d) Keep the default value for “API endpoint type”
    e) Click “Create API” in the bottom right corner
3) To create a new resource (endpoint):
    a) Click the “Create Resource” button located above the list of resources
    b) Select an existing resource path to branch off of. E.g., choosing the path “/” will make endpoints like “/endpoint”, and choosing the path “/test” will make endpoints like “/test/endpoint”.
    c) Provide a meaningful name
    d) Click “Create Resource” in the bottom right corner
4) To create a method for an endpoint:
    a) Select the endpoint from the list of resources
    b) In the “Methods” box, click on “Create method”
    c) Choose the desired method from the drop down
    d) Choose the integration type. This will most likely be Lambda Integration, and the remaining steps will assume this was the selected integration method.

Creating API Gateway - Rest API Continued:

- g) Provide the ARN (Amazon Resource Number) of the Lambda function to handle requests for this endpoint and method. These ARNs should be for the lambda functions created earlier.
- h) Keep the default integration timeout value unless explicitly specified
- i) In "Method Request Settings", click on the "API Key required" checkbox
  - See "Creating API Gateway - API Key Usage" if using an api key
- j) Add any query parameters in the "URL query string parameters" as necessary
- k) Click "Create Method" when everything meets the expected criteria

5) To update a method:
   - g) Select the method under the endpoint to update from the resources list
   - h) To update the method's request settings:
     - i) Click "Method Request" located underneath the "Client" image.
     - ii) Click "Edit" on the right side of the screen
     - iii) Change values as needed
   - i) To update the method's integration request:
     - i) Click "Integration Request" located to the right of "Method Request" and underneath "Method Response"
     - ii) Click "Edit" on the right side of the screen
     - iii) Change values as needed

Creating API Gateway - Rest API Continued:

6) To deploy the API:
   a) After adding all the necessary endpoints and methods, click “Deploy API” in the top right corner
   b) Choose the stage to deploy to from the dropdown menu
      - If no stage exists, click “*New stage*” and provide a meaningful name and optional description.
   c) Click “Deploy” from the bottom right of the pulled up menu
7) The API should be deployed now, and the url to use as the base url of an api request is available under “Invoke URL” in the middle of the screen.

   E.g. to GET the endpoint /test given an invoke url of “https://execute-api.us-east-1/deployment”:

   curl -X GET https://execute-api.us-east-1/deployment/test

Creating API Gateway - API Key Usage:

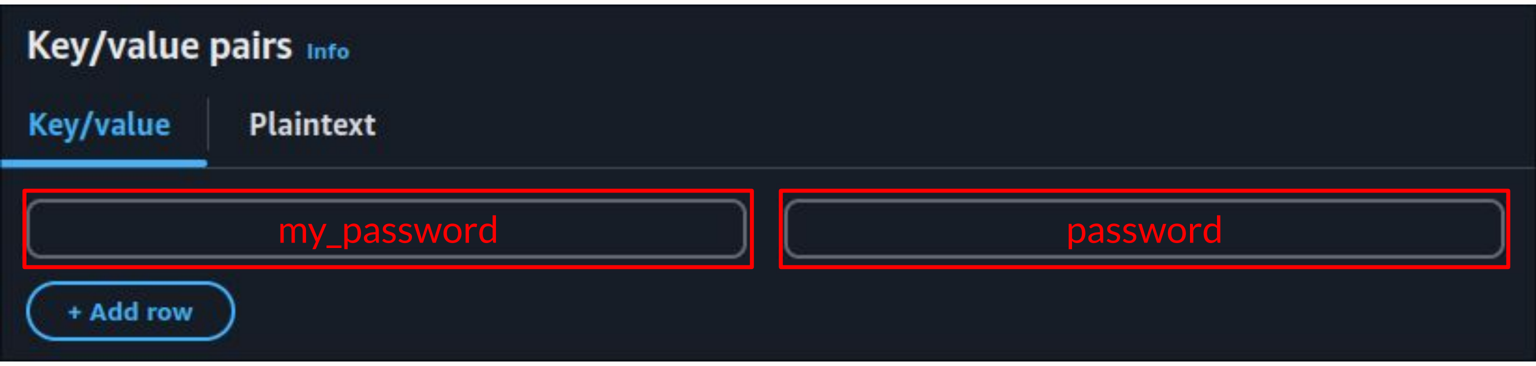
1) Navigate to the API Gateway page within your AWS AdminAccess Account
2) Navigate to the “API Keys” page using the left sidebar navigation menu
3) Click “Create API Key” in the top right corner
   a) Provide a meaningful name
   b) Optionally provide a description
   c) Choose to have AWS automatically generate a 20-character value for the key, or choose to manually provide a value for the key (minimum 20-characters)

Creating API Gateway - API Key Usage Continued:

4) Navigate to the “Usage Plans” page using the left sidebar navigation menu
5) Click “Create Usage Plan” in the top right corner
   i) Provide a meaningful name
   ii) Optionally provide a description
   iii) Turn off “Throttling”
   iv) Turn off “Quota”
   v) Click “Create Usage Plan” in the bottom right corner
6) Click on the newly created usage plan
7) In “Associated Stages”, click on “Add API Stage”
   i) Select the api that will require api key authentication from the dropdown
   ii) Select the deployed stage from the dropdown
   iii) Click “Add to usage plan”
8) Under “Associated API Keys”, click “Add API Key”
   i) Select “Add existing key” for the type
   ii) Select the created api key from the dropdown
   iii) Click “Add API Key”
9) The selected api will now require the value of the selected api key for all methods that require an api key
   i) To get this value, navigate to either the api key in “API Keys” or to the “Associated API Keys” in the usage plan that uses the api key
   ii) In the “API Key” column, click on the squares to copy the hidden value
   iii) Optionally, to see in plan text, click on the name of the api key, then click “Show” next to the hidden field of the api key

AWS Secrets Manager - Creating a Secret:

1) Navigate to the Secrets Manager page within your AWS AdminAccess Account
2) Click "Store a New Secret" in the top right corner
    a) Select the correct secret type. To store a general purpose secret, select "Other type of secret"
    b) For each secret you want to store, in "Key/value pairs" give a name to reference the key by in the left box, and the value in the right box. For instance, to store the secret "password" with the key "my_password", the string "my_password" would go in the left box (and would be the reference string for the secret), and "password" would go in the right box



my_password

password

    c) To add more secrets, click "+ Add row", repeat part b).
    d) Keep the default Encryption key unless explicitly told otherwise
    e) Click "Next" in the bottom right to continue
    f) Provide a meaningful name to "Secret Name"
    g) Optionally provide a description
    h) Keep all other values default
    i) Click "Next" in the bottom right
    j) Keep all values default
    k) Click "Next" in the bottom right
    l) Review the information to ensure it is correct
    m) Click "Store" to store the secret

AWS Secrets Manager - Accessing a Secret:

1) Navigate to the Secrets Manager page within your AWS AdminAccess Account
2) Select the name of the collection of secret(s) to get the values of (this is the meaningful name from “Creating a Secret”)
3) In the “Secret Value” section, click “Retrieve secret value” on the right side
4) All of the key: value pairs of secret names and values will appear. Copy the desired value.

Uploading to S3 Bucket:

1) Navigate to the S3 page within your AWS AdminAccess Account
2) From the “General purpose buckets”, click on the name of the bucket to upload a file to
3) Buckets are essentially a directory on a local computer. Navigate to the directory where the file uploaded should be put
4) Click “Upload” in the top right corner
    a) Add file(s)/folder(s) to upload using the corresponding buttons in the top right corner
    b) Keep all the other values as default unless explicitly stated otherwise
    c) Click “Upload” in the bottom right corner to upload the content to the bucket

User Guides

Makerspace Site

https://visit.cumaker.space/

Purpose:

To provide a centralized location for Makerspace visitors and interns to access forms and view data.

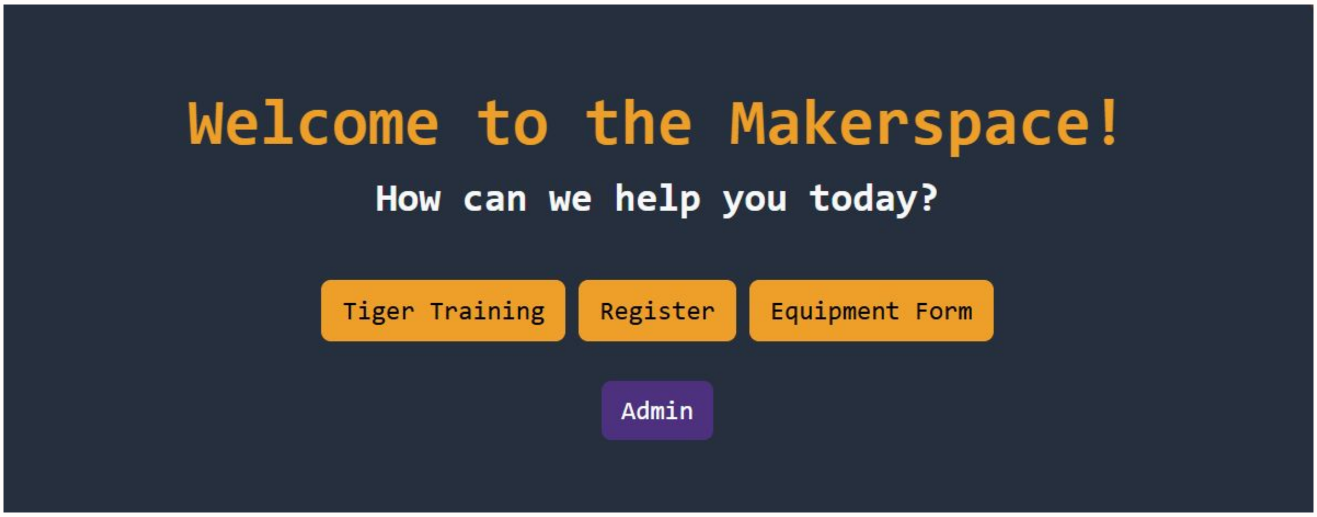
Roles:

Admin - Makerspace interns with access to administrator credentials

Visitor - Anyone with a Clemson username who visits a Makerspace location

Visitor access:

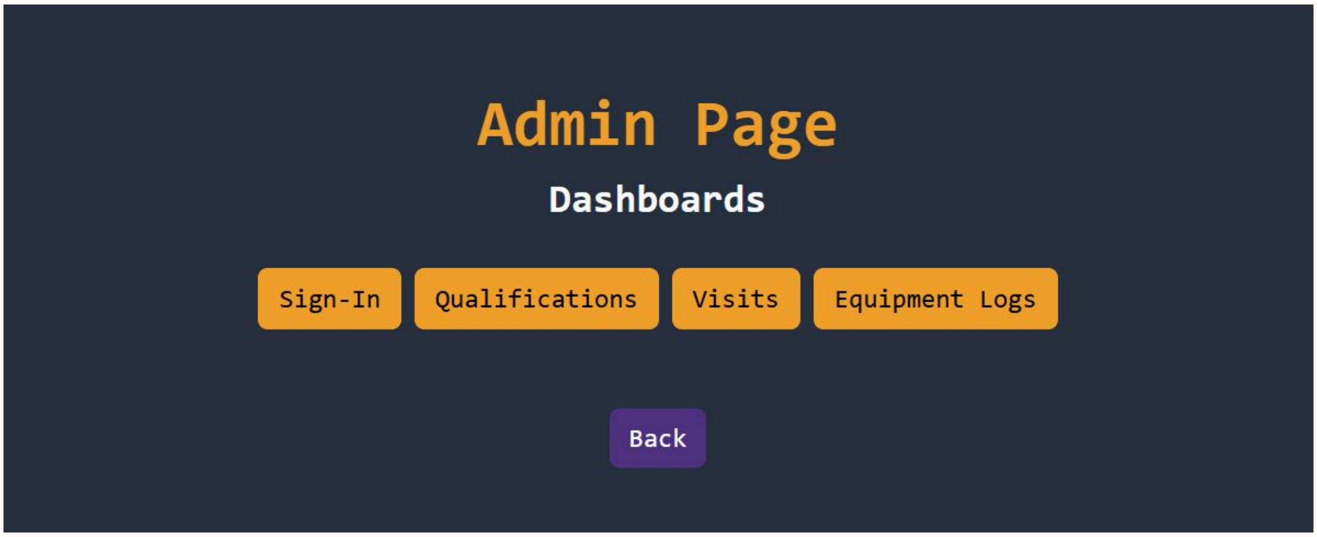
Visitors have access to everything on the home page. This includes the link to access TigerTraining courses, the Registration form and the Equipment Usage form.


Welcome to the Makerspace!

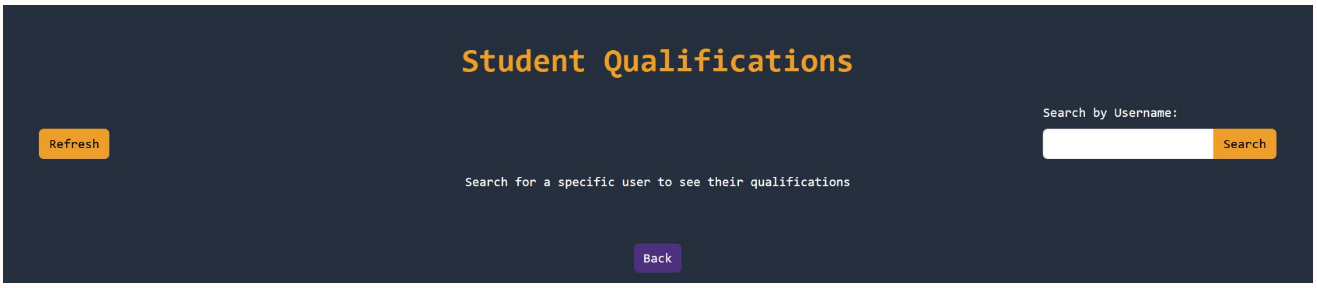
How can we help you today?

Admin access:

Admins have access to everything that visitors have access to as well as the admin page. This page contains a link to CU Course to set up the sign-in scanner and the dashboards for qualifications, visits and equipment usage.

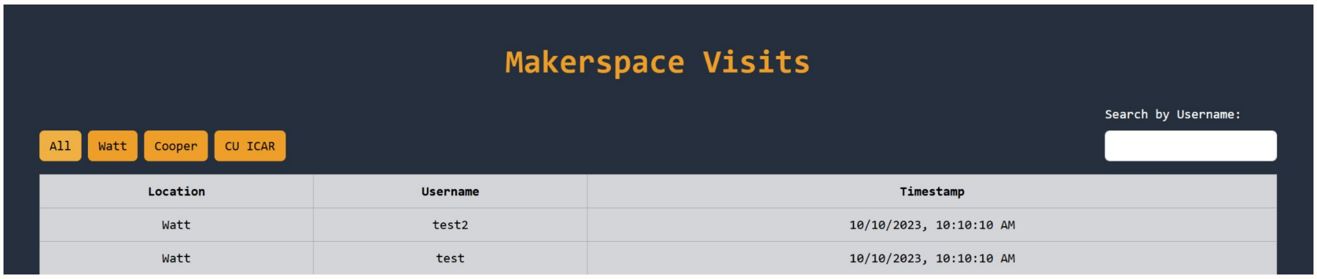


Qualifications Dashboard - Admins can view the completed training courses and waives for a user using their Clemson username. They can also use the refresh button to request updated data from Tiger Training. This button should only be used when necessary to prevent excessive spending.



Search for a specific user to see their qualifications

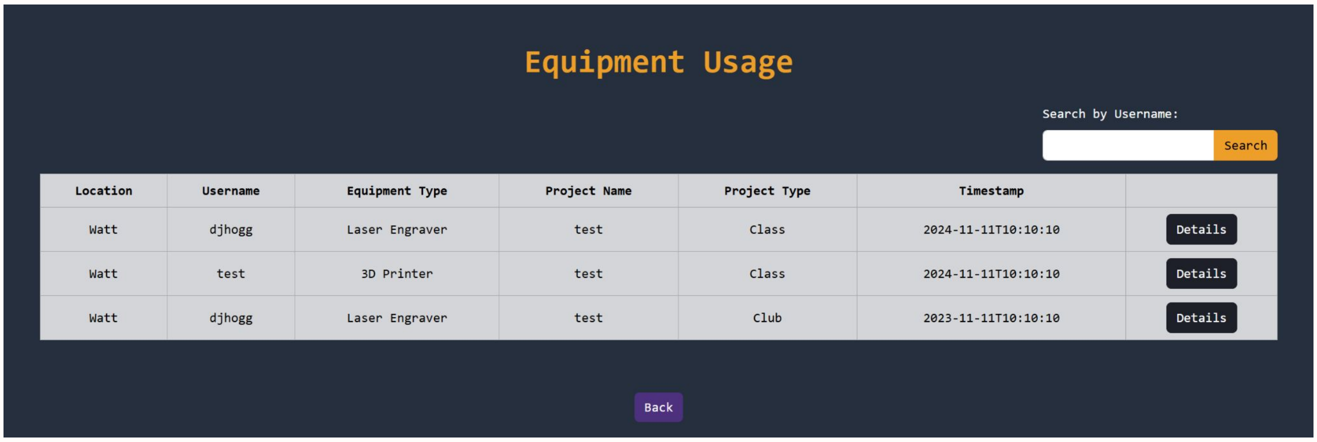
Visits Dashboard - Admins can view recent visits to all Makerspace locations. They can filter by location using the buttons in the top left and search for a specific username in the top right.



Makerspace Visits

Location	Username	Timestamp
Watt	test2	10/10/2023, 10:10:10 AM
Watt	test	10/10/2023, 10:10:10 AM

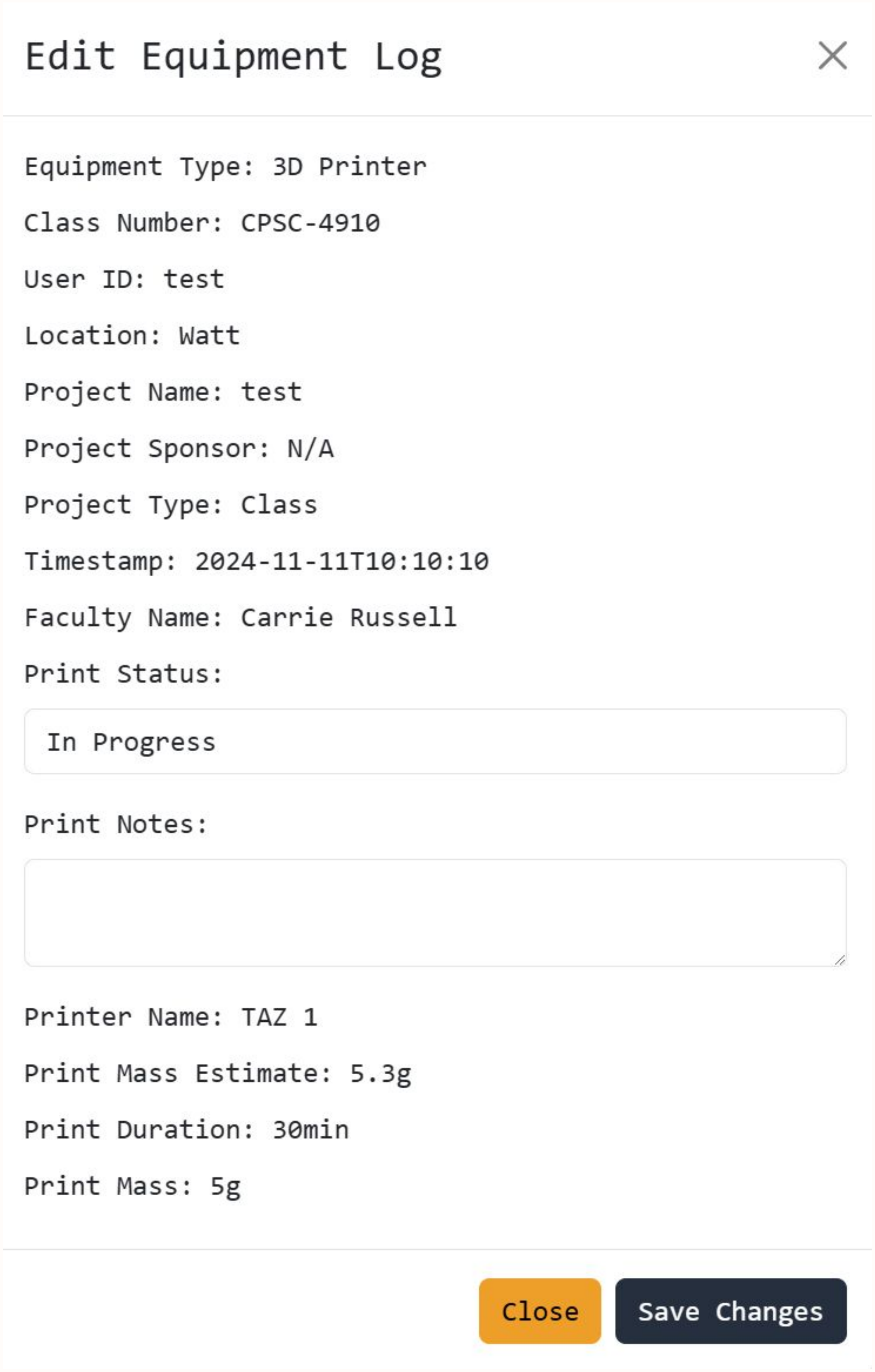
Equipment Usage Dashboard - Admins can view recent equipment usage logs and view additional details using the Details button. This will bring up a modal page where they can see all the information from that equipment log and update the status of 3D prints.



Equipment Usage

Location	Username	Equipment Type	Project Name	Project Type	Timestamp	
Watt	djhogg	Laser Engraver	test	Class	2024-11-11T10:10:10	Details
Watt	test	3D Printer	test	Class	2024-11-11T10:10:10	Details
Watt	djhogg	Laser Engraver	test	Club	2023-11-11T10:10:10	Details

Edit Equipment Log Modal



Edit Equipment Log

Equipment Type: 3D Printer

Class Number: CPSC-4910

User ID: test

Location: Watt

Project Name: test

Project Sponsor: N/A

Project Type: Class

Timestamp: 2024-11-11T10:10:10

Faculty Name: Carrie Russell

Print Status:

In Progress

Print Notes:

Printer Name: TAZ 1

Print Mass Estimate: 5.3g

Print Duration: 30min

Print Mass: 5g

Tiger Training

Purpose:

To provide a consistent platform to host the Makerspace's training courses, waivers, and other miscellaneous content that users can complete at their own discretion.

Gaining Access:

1) Email Mary-Kate for "Author" account access
2) You may be required to take a crash course with the Tiger Training staff to be provided with materials on how the system works before gaining access to the author role. If you are not required to take the course, you may always request it.

Creating Courses:

1) Visit https://clemson.bridgeapp.com/
2) On the left sidebar menu, select "Author > Courses"
3) Click "+ New Course" in the top right
    i) Name the course in the top left corner
    ii) Add content, questions, and change settings as needed
    iii) Click "Publish Course" in the top right after changing any content of the course
    iv) Optionally, change any of the course settings after publishing the course content. This does not require the course to be republished. It is generally recommended to turn the "Add due date" option off as the intended purpose of the Makerspace's training is to allow users to complete the course(s) at their own convenience.

Creating Courses Continued:

4) Course type specification:
   - As per the current requirements and general feedback, content relating to the Makerspace can take on one of the following types when stored in the backend: training, waiver, and miscellaneous.
   - To specify a course as training, include the word "training" (case insensitive) in the name of the course
   - To specify a course as a waiver, include the word "waiver" (case insensitive) in the name of the course
   - As of the current implementation, any course that does not include either of the previous keywords in its name will be considered a "miscellaneous" course

Examples :

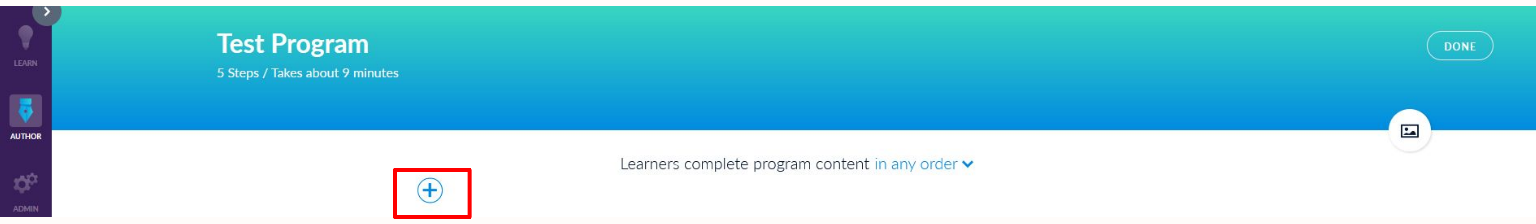
Name of Course	**Type of Course**
"Test - Training"	Training
"Test - training"	Training
"Test - Waiver"	Waiver
"Test - Nothing"	Miscellaneous

Notes:

- Though not truly enforced, it is recommended to branch Makerspace courses and program(s) with "Makerspace" – or something to that effect – in the names. This makes it easy to locate them via searching, and generally ensures edits are made only to content owned by the Makerspace.

Creating a Program:

1) Visit https://clemson.bridgeapp.com/
2) On the left sidebar menu, select "Author > Programs"
3) Click "+ New Program" in the top right
    i) Name the program in the top left corner
    ii) To add courses, hover your mouse over the region to the left of "Learners complete program…"
    iii) Click on the plug-sign icon that pops up to search for courses to add
    iv) Add courses by name


LEARN
AUTHOR
ADMIN
Test Program
5 Steps / Takes about 9 minutes
DONE
Learners complete program content in any order

    v) Due to the intended effect of "users can finish content at their convenience" it is recommended to change the program to "Learners complete program content *in any order*" from "*in order*". This will enable users to finish the courses they want to, when they want to.
    vi) Click "Done" in the top right when finished with creating the program

CU Course – Visit Tracking

Purpose:

To provide a method of tracking user visits to the Makerspace. Enables users to scan their smart device with a hardware scanner to sign them into the Makerspace.

Links:

- CU Course Development Site
- CU Course Production Site

Gaining Access:

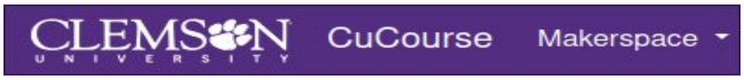
1) Email Dana for account access to the Makerspace sessions

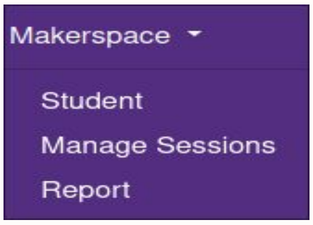
Usage Requirements:

- A computer with a USB-A port (dedicated for sign in)
- Internet access to Clemson's network
- A CCIT hardware scanner

Using CU Course:

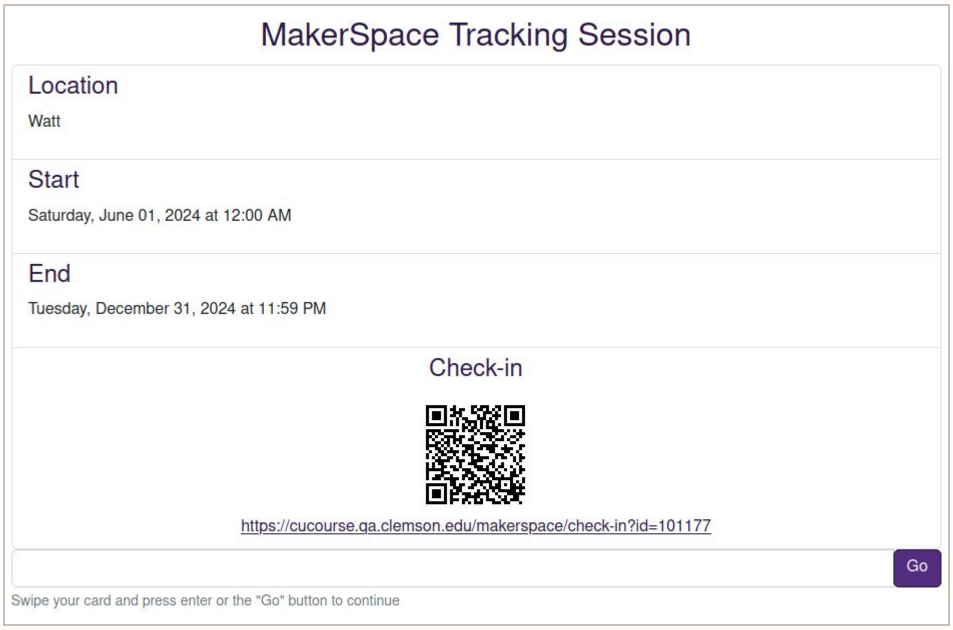
1) Using the dedicated computer, sign in to the production CU Course site
2) Access the "Manage Sessions" section for the Makerspace by clicking on "Makerspace" in the bar across the top of the page





Using CU Course Continued:

3) There should be three sessions (one for each provided Makerspace location: Watt, Cooper, and CUICAR) that should be automatically generated
   - In the case the sessions are not there, please contact Dana and view "Creating a Session Manually" to create a temporary session to continue tracking visits in the meantime.
4) Click on the "Swipe" link on the right side of the screen that is for the session with the location of the Makerspace you are in
5) A page similar to the following should have shown up with the text box next to the "Go" button selected


MakerSpace Tracking Session

Location
Watt

Start
Saturday, June 01, 2024 at 12:00 AM

End
Tuesday, December 31, 2024 at 11:59 PM

Check-in

https://cucourse.qa.clemson.edu/makerspace/check-in?id=101177

Swipe your card and press enter or the "Go" button to continue

6) Plug in the hardware scanner to the computer
7) You must now leave the computer on, with the text box next to "Go" selected with the mouse, and remain on this screen for as long as visits should be collected. Users will scan their smart device on the scanner which should automatically sign them in.

Notes:

The Makerspace CU Course was custom made to the following requirements: allow for users to sign in as many times as they want; automatically send the data to the current AWS backend to log the visit; automatically create a Spring session (January 1 - May 30) and Fall session (June 1 - December 31) for all three Makerspace locations. If these requirements need to be changed, please ensure the new requirements are thoroughly thought through and contact Dana to begin the process of changing the implementation details.

Creating a Session Manually:

1) Access the "Manage Sessions" section for the Makerspace by clicking on "Makerspace" in the bar across the top of the page
2) Click the "Add Session" button near the top left corner
3) Fill out the fields with the appropriate information
    - i) Provide a meaningful title to "Title"
    - ii) Make "Capacity" 0 (zero) to allow for unlimited sign ins
    - iii) Select "Anyone" for the "Type"
    - iv) Choose an appropriate start and end date
    - v) Provide one of the following values (case sensitive) to the location: Watt, Cooper, CUICAR
    - vi) Optionally provide a description in "Session Description"
    - vii) Leave every other field with the default values (or blank)
4) Scroll down and click "Save" near the bottom left to create the session
5) To start collecting data from this manual session, see Part 4 - Using CU Course and select "Swipe" for the created session

Future Work Ideas

1) Migrate old DynamoDB table's data ({stage}-Database-{stage}-Database…) into in the new DynamoDB tables ({stage}-Database-...)
2) Determine stable versions for each of the modules in requirements.txt for future deployments. Ensures the pipeline can still work no matter what changes are made in future updated to AWS CDK V2.
    a) Although, if something becomes unsupported, i.e. a method, then the version must be updated when encountered to the supported version and update the troublesome method.
3) Disconnect DNS Records from being populated via the Pipeline. Manually create the Hosted Zones and associated records (A, NS, CNAME, SOA) and AWS ACM Certificates for each domain (beta-visit, beta-api, etc) and reference these by ARN through the code. (or a different working method)
4) Improve the testing suite of the pipeline. Currently, basic tests exist that only test functionality of the backend handlers. More tests can be created to extend proving the handlers function as expected. Another testing suite could be implemented to test that the api endpoints are reachable using the urllib3 or requests. With both of these suites in use, it would confirm new logic added to the backend functions as expected, and that the backend can actually be reached via requests.

5) Additional data fields to possibly collect. According to Dana, though not one-hundred percent confirmed, CU Course should be able to send more user information – fields perhaps similar to year, name, college, etc – when users sign in. This data could be sent to the users endpoint while the visit is still logged to the visits endpoint. Other data to be collected could be equipment usage for equipment other than 3d printers, or perhaps some more explicitly defined course types for Tiger Training and the qualifications endpoint.
6) Automatically create the admin user, with a password, in the User Pool that is automatically created through the Pipeline. Ensure this is not visible through GitHub and can be passed on to the Makerspace (automatically or manually) to allow access to the Admin portal on the site.
7) Look into Amplify authentication for the frontend to prevent weird prevented login issues. We are unsure exactly what causes this issue, but every now and then, if a user refreshes a page behind an Amplify authentication – or closes the tab and reopens to those pages, it sometimes logs the user out. The problem lies in, when logging in, it thinks the user has already logged in (cookies saved in browser), and the user cannot log in again to view the page. The only way to fix this, as far as we know, is to clear the browser's cookies for the visit page.
8) Look into data visualization / data analytics solutions that can, ideally, easily integrate into the visit site. The goal is to move the Makerspace away from reliance duplicating the data from AWS to their data analytics google sheet, and instead keep them all in one place. The solution must be easily accessible to all people of all backgrounds, including those that may or may not be technically savvy. One example that fails to meet this is Grafana.

Contacts

Capstone Team

- Caleb Murphy - cmurp25@clemson.edu
- Noah Broome - nmbroom@clemson.edu
- Duncan Hogg - djhogg@clemson.edu

AWS

- Tommy Johnston - awstommy@amazon.com
- Owen Phillips - oap@amazon.com

Makerspace Faculty Advisor

- Dr. Todd Schweisinger - todds@clemson.edu

Tiger Training Admin

- Mary-Kate Califf - marykat@clemson.edu

CU Course – CCIT Applications Developer

- Dana Freudenberger - freuden@clemson.edu