OCCTIVE Content Guide

Last Updated: November 8, 2025

The following document describes the structure of the data used in the Google Sheet. This sheet provides the source data for the OCCTIVE website and also gives insights into the code hosted on GitHub.

Table of Contents:

Where's My Data?

Data on this site is generally stored in one of two locations.

Dynamically via the Content Spreadsheet, or **Statically** via the codebase. Both of these can be edited easily, and most of it will update via the content spreadsheet. The different tabs are documented in the rest of this document.

When does stuff update?

The site uses a method of caching data locally in the browser. To clear loaded data, the user must close the current tab containing the OCCTIVE page, and re-open the site. One thing to note is that Google Sheets can take a minute or so to refresh its data, so even after re-opening the site, it may take a few more minutes for new data to appear on the site. If you want to see changes straight away, open an incognito browser and visit the OCCTIVE page.

Static content updates with each server deployment. If you want to update this, make a change to the github repository (with the content update), then redeploy the site.

Google Apps Script Automation

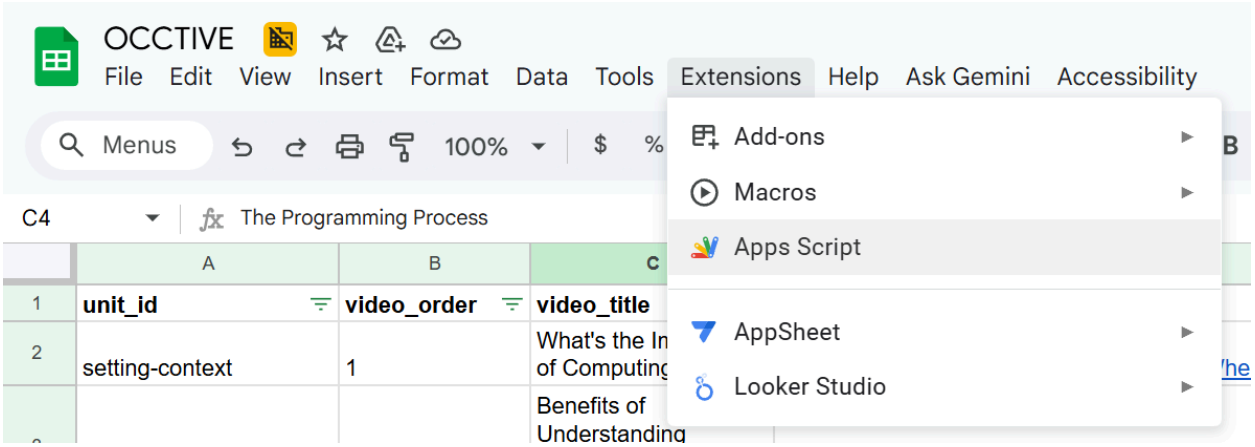
What is Apps Script?

Google Apps Script (GAS) is a built-in scripting environment for Google Sheets, Docs, and other Workspace tools. It allows you to automate actions, validate inputs, and react to changes. In this case, keeping your content spreadsheet clean, structured, and safe for use by the OCCTIVE web app.

Our spreadsheet uses two .gs files (Google Script files) that automatically run whenever someone edits specific cells in the "Units" or "Videos" tabs. They enforce validation rules, prevent errors that could break the site, and organize data automatically.

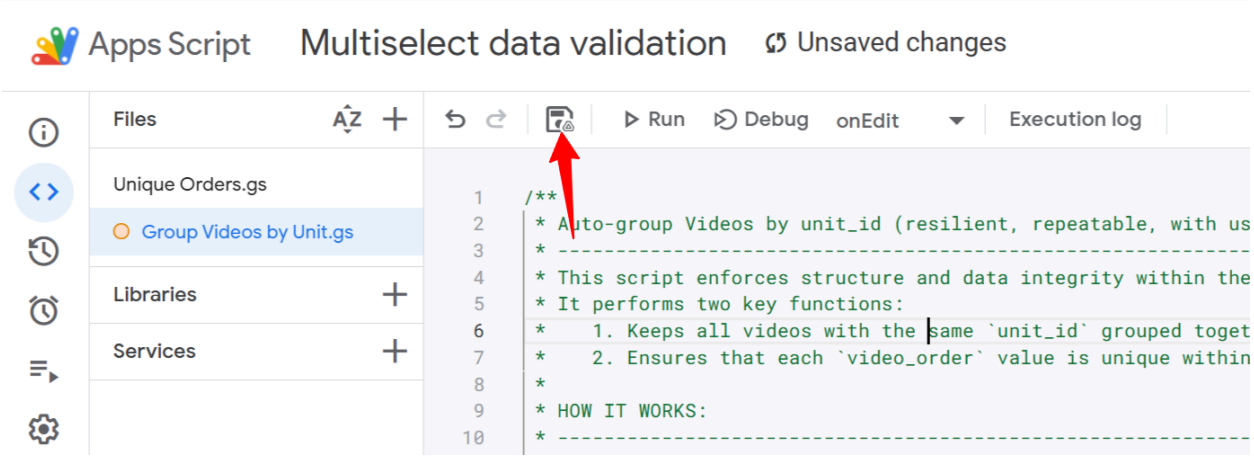
To open or edit these scripts:

In the spreadsheet, click Extensions → Apps Script.



The Apps Script editor will open in a new tab.

You'll see our scripts listed on the left under filenames such as Unique Orders.gs and Group Videos by Unit.gs. Each file is automatically linked to this spreadsheet only. Changes apply immediately once saved. You must click the save button after every change for the changes to apply.



Current .gs Files

UniqueOrders.gs – Unit Order Validation

This script maintains the integrity of the "Units" tab by enforcing two key rules:

1. Unique Order Values: No two rows can share the same order number (column F).
2. Positive Numbers Only: The order value must be a positive whole number (1, 2, 3, …).

If a user enters a duplicate or invalid value, the script automatically:

- Reverts the cell to its previous value.
- Displays a popup explaining what went wrong.

Danger:

This script explicitly references column F (orderColumn = 6). If you move, insert, or delete columns, you must update this column number in the script. Otherwise, the validation will stop working or target the wrong column.

AutoGroupVideos.gs – Video Organization and Order Validation

This script manages the “Videos” tab and performs two main functions:

1. Auto-grouping by Unit ID:
   When a user edits the unit_id column (column A), the script finds all rows with the same ID and automatically moves the new or edited row just below that group.
   It keeps all videos for a topic together, maintaining a clean, logical structure.
   A popup message appears to confirm when a move occurs.

2. Unique Video Order Validation:
   When a user edits the video_order column (column B), the script checks that the new order number is unique within that unit_id group. It checks whenever any other column of that row is edited. If a duplicate order is found, the change is reverted and a warning message is displayed.

Danger:

Like UniqueOrders.gs, this script uses hardcoded column references:

- unitIdCol = 1 → column A
- orderCol = 2 → column B
  If you rearrange the spreadsheet columns, rename the sheet, or add new columns to the left, you must update these values in the script. Failing to do so will cause the automation to stop functioning correctly or to rearrange the wrong data.

Deployment

Deployment happens by copying a built version of the project to the server.

0. Install npm (https://nodejs.org/en/)
    a. Select the LTS version, and then choose the relevant installer for your operating system.
1. Install yarn (https://classic.yarnpkg.com/en/docs/install#mac-stable)
    a. `npm install --global yarn`
    b. or `sudo npm install --global yarn` if there are permission issues
2. Clone the repository locally (https://github.com/melvyn9/OCCTIVE/tree/main)
    a. `git pull` if you are out of date
3. Install dependencies via yarn
    a. Run `yarn install` in the directory of the repo
4. Build the project
    a. `yarn build`
5. Deploy the project (requires the server password)
    a. `npx server -s build`

Running the Site Locally

You can run it locally via yarn.

1. Clone the repository locally (https://github.com/melvyn9/OCCTIVE/tree/main)
    a. `git pull` if you are out of date
2. Install dependencies via yarn
    a. Run `yarn install` in the directory of the repo
3. Run a local copy of the site
    a. `npx serve -s build`

Running Locally Again

If you have successfully run the site locally on your computer in the past and have come back to run it again. You will need to install the dependencies again.

1. Open the GitHub repository on your favorite IDE and make sure to `git pull`.
2. In the terminal of the IDE, type
    a. `npm install –global yarn`
    b. `yarn build`
    c. `npx serve -s build`

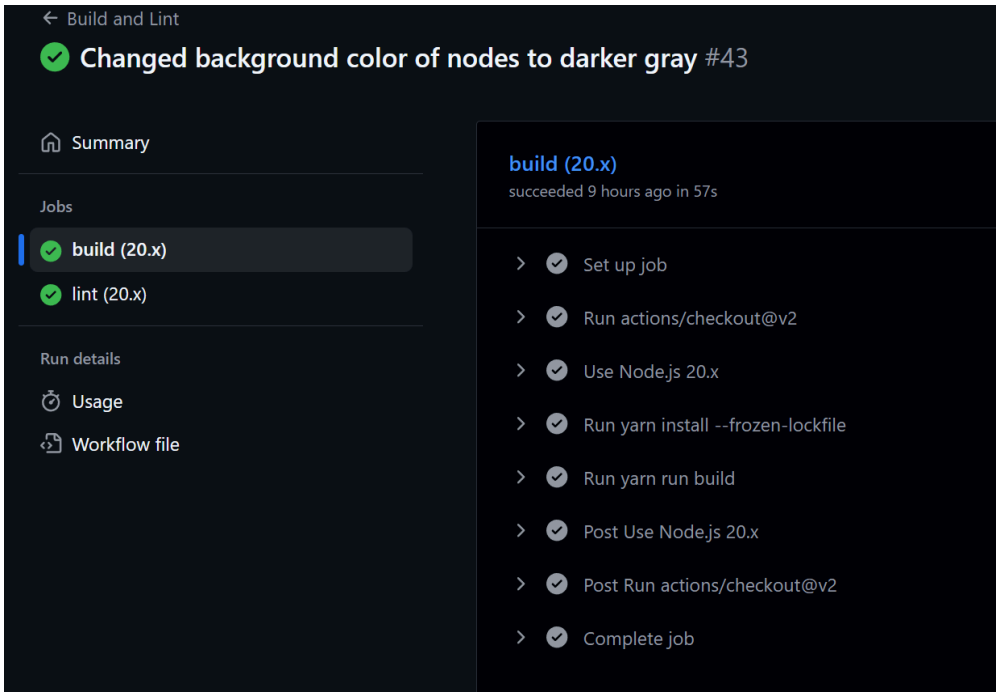
Automatic Testing

Whenever code is pushed into a branch or whenever there is a pull request, there is a build process that will be run to ensure the code is of high quality and follows a good coding standard. The build process is run using Github Actions, more about it can be read here. The build process can be found in the repository in the **.github/workflows** folder.

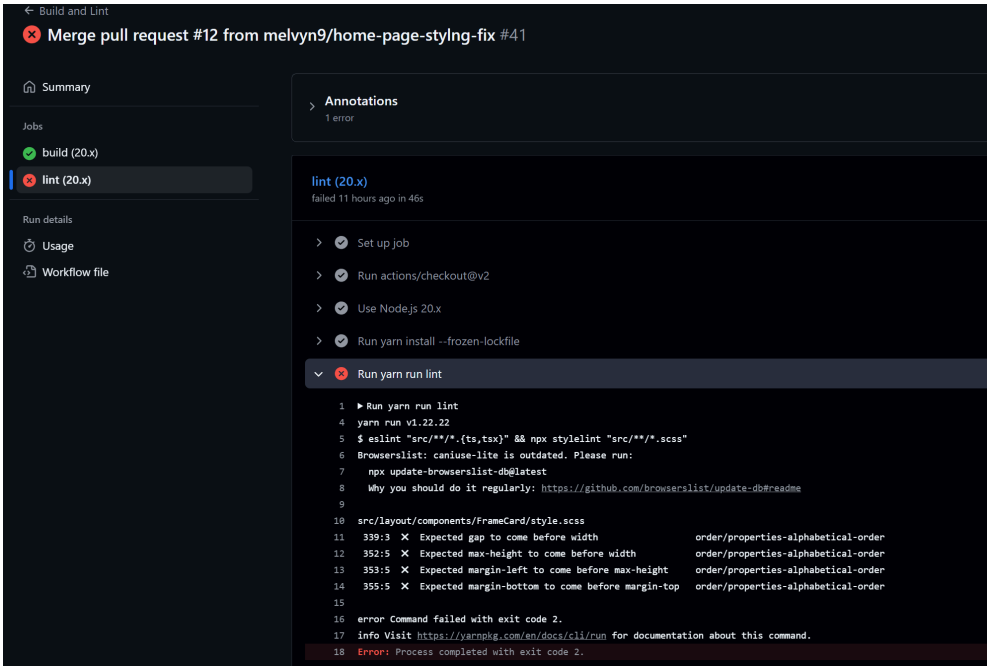
There is currently one build process called **Build and Lint**, located in **lint.yml**. This file contains a linter that is used to ensure good coding practices as well as reduce the need for redundant code such as catching unused variables, indentation, and many others. More coding standards can be added by editing the **lint.yml** file. To get a better understanding of what a linter is and how it is used, this is a good resource to look through.

You can check locally if you pass test cases with `yarn run lint` and you can automatically fix most errors with `yarn run lint:fix`.

An example of a successful build + lint run.



An example of an unsuccessful build + lint run.


Build and Lint
Merge pull request #12 from melvyn9/home-page-stylng-fix #41
Summary
Jobs
build (20.x)
lint (20.x)
Run details
Usage
Workflow file
Annotations
1 error
lint (20.x)
failed 11 hours ago in 46s
Set up job
Run actions/checkout@v2
Use Node.js 20.x
Run yarn install --frozen-lockfile
Run yarn run lint
1 ▶ Run yarn run lint
4 yarn run v1.22.22
5 $ eslint "src/**/*.{ts,tsx}" && npx stylelint "src/**/*.scss"
6 Browserslist: caniuse-lite is outdated. Please run:
7 npx update-browserslist-db@latest
8 Why you should do it regularly: https://github.com/browserslist/update-db#readme
9
10 src/layout/components/FrameCard/style.scss
11 339:3 ✖ Expected gap to come before width order/properties-alphabetical-order
12 352:5 ✖ Expected max-height to come before width order/properties-alphabetical-order
13 353:5 ✖ Expected margin-left to come before max-height order/properties-alphabetical-order
14 355:5 ✖ Expected margin-bottom to come before margin-top order/properties-alphabetical-order
15
16 error Command failed with exit code 2.
17 info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
18 Error: Process completed with exit code 2.

Updating Dependencies

It is important to update dependencies frequently as newer versions of node may be incompatible with the older versions of the dependencies. Therefore, you may encounter bugs or error messages when trying to run the site locally on a newer version of node. If this is the case, always be sure to check your version of node using `node -v` and then switching to an older version of node by using node packet manager. It is recommended to update the dependencies once a month.

We can view the outdated dependencies by typing `yarn outdated` or `npm outdated` in the terminal. You can see the versions of the dependencies that require updating along with their version numbers. This gives you an idea as to which dependencies require updating and if you choose to do so can manually update each dependency one at a time. However, this approach can be time consuming as we have many dependencies that require updating. Therefore, the easiest way to update the dependencies is by using `npm update`. This command will update all the dependencies to the “wanted” version.

If it is the case that you get an error, it may be because some dependencies do not get updated to a major version. If this is the case, try using `npm audit fix --force` but be sure to **do this on a branch and not on main** as this can cause the build to break. Please be very careful when using this command, read up more about the command here and here to understand what it does before using it.

Publishing to GitHub Pages

The package.json file includes scripts to build and deploy the website to GitHub Pages, a static hosting service provided by GitHub. This allows your project to be accessible via a live public link like https://melvyn9.github.io/OCCTIVE.

To make changes live on GitHub Pages:

1. In package.json, add the following line **below** "private": true:
   "homepage": "https://melvyn9.github.io/OCCTIVE",
2. Then run the following commands in your terminal:
   npm run build
   npm run deploy

Script Breakdown

- "predeploy": "npm run build"
  Automatically runs the build script before deploying.
- "deploy": "gh-pages -d build"
  Publishes the contents of the build/ folder to the gh-pages branch using the gh-pages tool.

Troubleshooting GitHub Pages

If your site shows a 404 error not found at https://melvyn9.github.io/OCCTIVE/#/:

1. Go to your GitHub repository settings.
2. Under the “Pages” section, find the “Build and deployment” settings.
3. Change the source branch from gh-pages to another branch like main, and click Save.
4. Then change it back to gh-pages, click Save again.
5. Wait 3–5 minutes, then refresh your GitHub Pages link.

This will reset the GitHub Pages configuration and trigger a fresh deployment.

Updating Local CSV Backups

The OCCTIVE site relies on Google Sheets as the primary data source.
To guard against downtime or access issues, we maintain **local backup CSV files** inside the repository (`/public/data`). These backups are used automatically if the Google Sheets data cannot be loaded.

Follow these steps to update the backups:

Automated Monthly Update (Preferred Method)

This process is now fully automated through a GitHub Action workflow.
The workflow automatically downloads the latest data from the Google Spreadsheet on the first day of every month and commits any updated CSV files to the repository.
It only creates a commit if there are actual content changes between the new and existing CSVs.

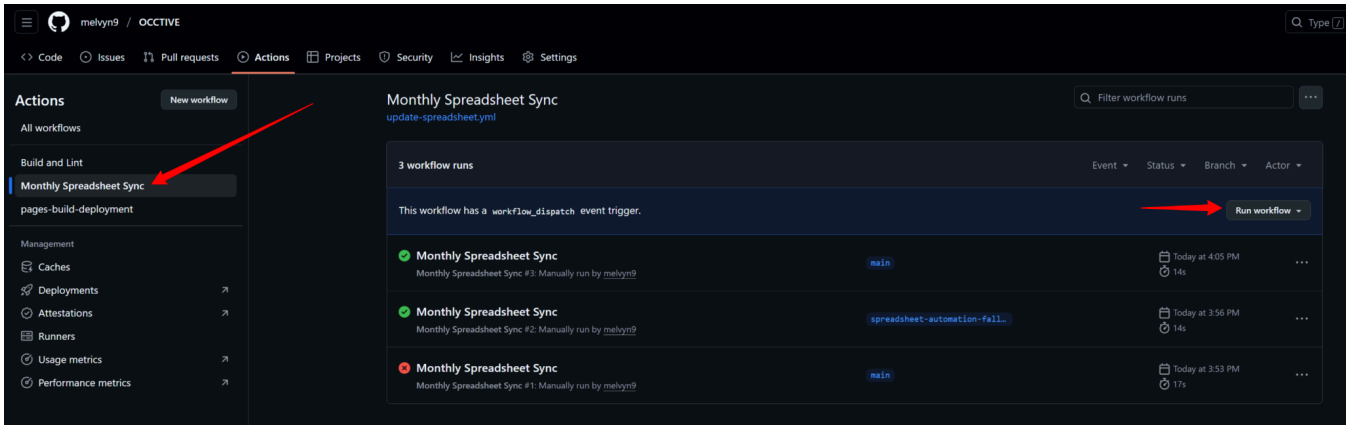
Workflow details:

- **File location:** `.github/workflows/update-spreadsheet.yml`
- **Workflow name:** *Monthly Spreadsheet Sync*
- **Runs automatically:** At 12:00 AM Pacific Time on the 1st of every month
- **Data source:** Public CSV export links for each sheet (Videos, Units, Dependency Graph)
- **Output files:**
  - `public/data/videos.csv`
  - `public/data/units.csv`
  - `public/data/dependency_graph.csv`

To manually trigger the workflow:

1. Go to the **GitHub repository** page.
2. Click on the **“Actions”** tab at the top.

3. Select **“Monthly Spreadsheet Sync”** from the left sidebar.
4. Click **“Run workflow”** (top-right corner).



5. Choose the `main` branch, then click **Run workflow**.
6. The workflow will:
    - Download the latest CSVs from Google Sheets.
    - Commit and push the updated files to `/public/data` (if there are changes).
    - Log the details under the “Actions” tab for confirmation.

How it works internally:

- The `.yml` file uses a `curl` command to fetch each sheet as a CSV.
- GitHub Actions runs with `contents: write` permission, allowing it to push updates automatically.
- It runs as `github-actions[bot]`, and you’ll see commits like:

  `Updated CSVs: public/data/videos.csv public/data/units.csv public/data/dependency_graph.csv`

Important Note: Spreadsheet Visibility Requirement

This automated workflow only functions if the Google Spreadsheet is publicly accessible via its CSV export link. If the spreadsheet is set to private (restricted access), the curl command will fail because GitHub Actions cannot authenticate without credentials.

To make this workflow compatible with private sheets, you would need to use a Google Service Account with a private API key (stored as a GitHub Secret) and update the workflow to authenticate requests through the Google Sheets API instead of direct public CSV export links.

Manual Update (Fallback Method)

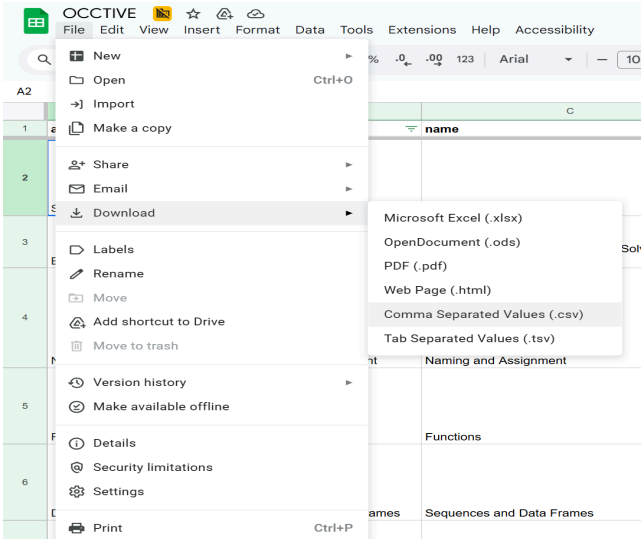
1. Open the Google Spreadsheet

- Navigate to the main OCCTIVE data spreadsheet (link provided by the team).
- Each tab corresponds to a dataset:
    - **Videos**
    - **Units**
    - **DependencyGraph**

2. Download CSV Files

For each tab:

1. Go to **File → Download → Comma-separated values (.csv)**



2. Save the file locally.

Rename each file exactly as follows:

- **Videos** tab → `videos.csv`
- **Units** tab → `units.csv`
- **DependencyGraph** tab → `dependency_graph.csv`

3. Replace Local Backup Files

1. In the repo, open the `/public/data` folder.
2. Drag and drop (or copy/paste) the new files in, overwriting the old ones.
    - `/public/data/videos.csv`
    - `/public/data/units.csv`
    - `/public/data/dependency_graph.csv`