

Database and Website for Underwater Landslide Research



Brianna Gaede
Lucas Kitaev
Vlad Muresan

CSMJobe
June 10, 2020

Outline

Outline	2
1 - Introduction	3
2 - Requirements	4
2.1 - Functional Requirements	4
2.2 - Non-Functional Requirements	4
3 - System Architecture	5
4 - Technical Design	8
5 - Quality Assurance	10
6 - Results	11
7 - Appendices	12
7.1 - Appendix I - Website - Viewer Page	12
7.1 - Appendix II - Website - Contributions Page	13
7.1 - Appendix III - Website - Administrator Access	14
7.1 - Appendix IV - Website - Downloading Data	15
7.1 - Appendix V - Links for Reference	16

1 - Introduction

Our client, Zane Jobe from the geology department at CSM requested an open source website to aid in the research of underwater landslides. The website should allow underwater landslide data to be uploaded, downloaded, and manipulated for research purposes. The scientific community and others in industry are interested in the movement of sediment, especially in subaqueous environments. Figure 1 below shows types of data collected. This information can help to predict natural events such as tsunamis as well as damage to pipelines that may be caused by a landslide. By having an open source platform to hold landslide data and modeling tools, smooth collaboration in the scientific community can be encouraged.

Specific requests from the client included a homepage on the website that shows general information about what data is being collected, why it is important, and other scientific information. Descriptions of the open source information that is available on the website as well as what tools for analytics are available on that page as well. Diagrams of an underwater landslide morphometrics and descriptions of the different types of the terminology the user would encounter along the way would complement the page. The main portion of the website is for data viewing and analysis. Users are able to download any data that they select as well as upload their data to the database. Selected data can be analyzed using plotting tools to show metrics and locations on a map. There is also a contribution page that contains all contributors to the collection of data as well as members on the team who helped develop the website.

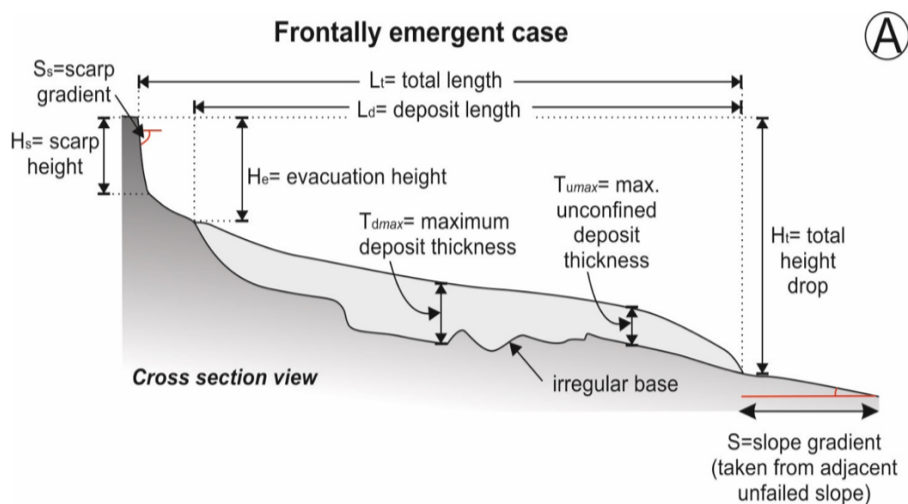


Figure 1. Diagram describing points of data accumulation for research and collaboration.

2 - Requirements

2.1 - Functional Requirements

- Data upload and download capabilities
 - Need to support CSV format
 - Have security measures to prevent bad uploads
 - Manual approval
 - Email service to check if data is good, ask for admin approval
- Database to store data
 - Python API to interact with database
 - Data is provided in XLSX format
 - Can convert to CSV and split into tables to import to database
 - SQL schema is provided
 - User input from forms (making a query)
 - Be able to search name, contributor, date, age
- Scatter plot capabilities
 - D3.js library can be used for visualization
 - Mutable X and Y axes, to allow for modular visualization
- Map to compare locations with data
 - Leaflet JS library
 - Display additional information when you hover over data points
- Ability to add contributors
 - User accounts
 - Username
 - Password
 - Email address
 - Verification system
 - Or log in with Google, Facebook, etc.
 - OAUTH API

2.2 - Non-Functional Requirements

- The website does *not* need to handle many concurrent users
- Performance should scale well as the database grows
- Hosting should preferably be done on the Mines domain
 - Should run on a Linux server
- The code and data are to be made open source

3 - System Architecture

The system we designed is built on the Django web framework. Django is a backend Python framework that offers dynamic URL paths for simplified website navigation, HTML templating for building dynamic web pages, and useful APIs that are described in the quality assurance section. Django can interface with many different database systems. For development, we used the built-in SQLite3 database, which uses simple file-based storage, however Django offers the flexibility to switch to a “bigger” database like PostgreSQL in a production environment. Django also provides a built-in lightweight HTTP server for use during development, but can easily integrate with a production HTTP server like Apache or NGINX.

On the frontend, the codebase uses TypeScript, which is a typed superset of JavaScript. TypeScript source code is transpiled to JavaScript browser code using Webpack with the Babel plugin for TypeScript. We are also able to use JavaScript libraries from within TypeScript, like JQuery for general utilities, D3.js for data plotting, and Leaflet for data mapping. Webpack bundles these dependencies together with our transpiled code into a single file for easy deployment. All of these dependencies are managed using npm, which enables automatic installation and updates of our selected packages.

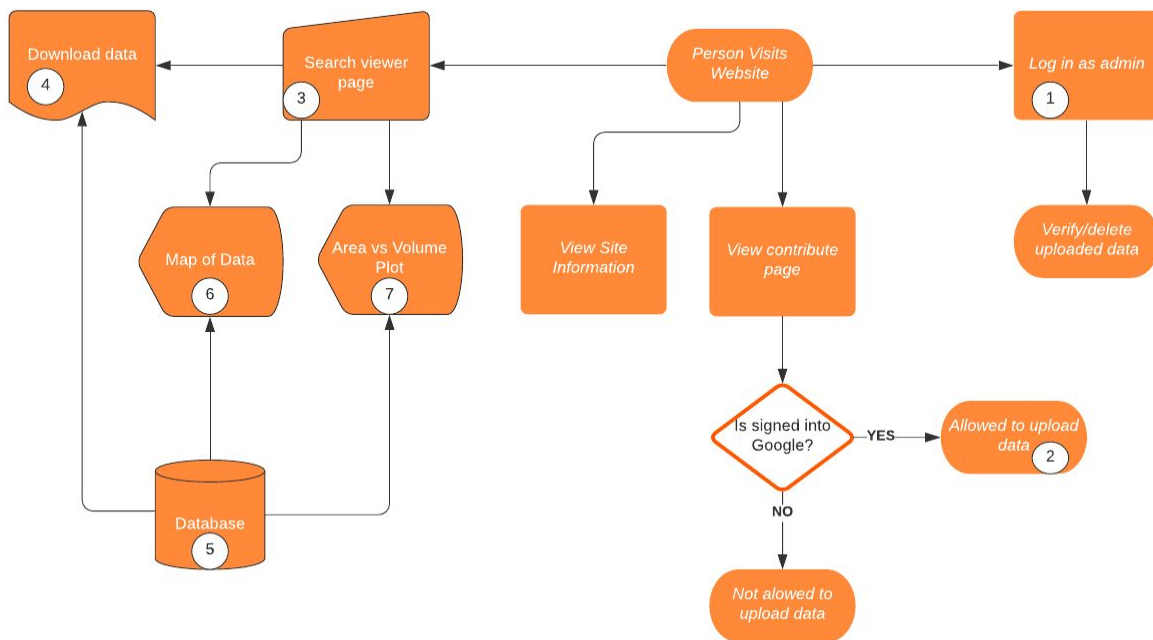


Figure 2. Website flowchart.

The above figure illustrates the high-level workflow for the website. Descriptions of each numbered label are included here:

1. Django provides a built-in administration website. Logging in as a Django admin allows you to verify and delete database entries.
2. If you are signed into Google via their OAUTH API, you may upload data to the database. All new uploads are marked as unverified by default. This is done to ensure that uploads come from real people, and potential banning of users that upload bad data, in addition to keeping track of contributor information.
3. We implemented a few different ways to search the database. There are fields for name, contributor, latitude and longitude, upload date, and age period (the geological time of the landslide).
4. You can download the selected data as .csv or a .json file. These are generated on-the-fly through the use of a browser data URI that encodes text content into a link that, when clicked, initiates a download.
5. The database is stored in four tables. All database entries must have a unique latitude and longitude combination. A more detailed description is available in the technical design section below.
6. We used the Leaflet JavaScript library to implement a map that displays the location of each data entry using latitude and longitude values.



Figure 3. Example of a Leaflet map.

7. We used the D3 JavaScript library to create a plot that compares the area and volume values of each row selected. The axes use a logarithmic scale, due to large differences in order of magnitude.

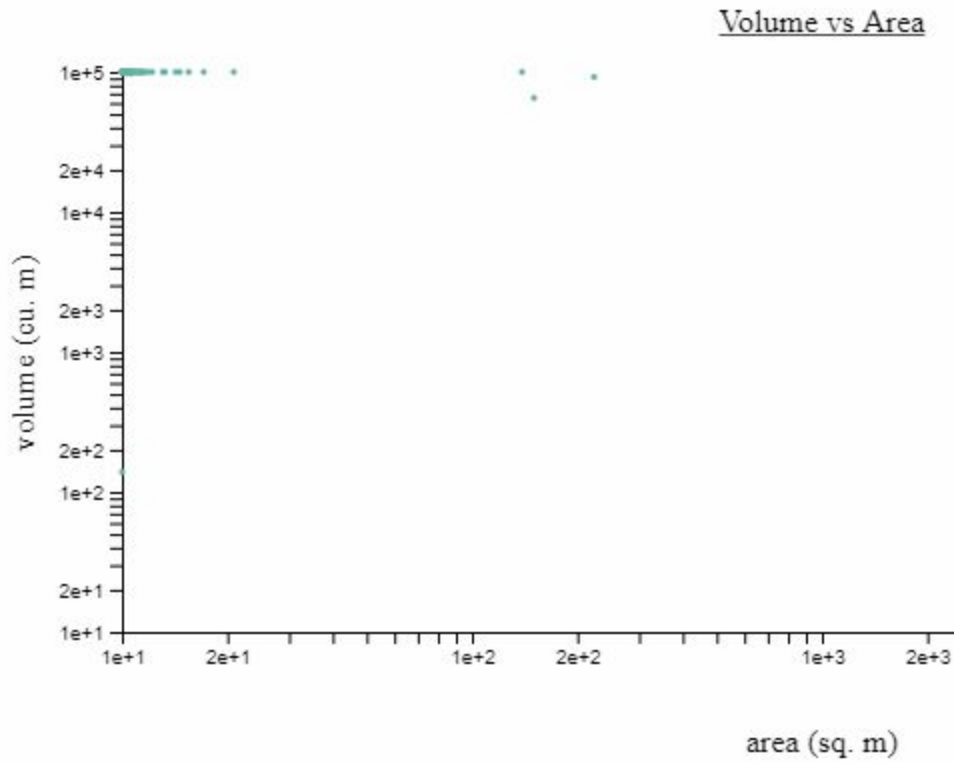


Figure 4. Example of a D3 plot. Results are skewed to the top due to issues with the dataset we were provided.

4 - Technical Design



Figure 5. Database tables.

The figure above shows what data is used to aid scientists in storing information in a standardized manner. `summary_info_id` contains an auto-incremented ID that serves as a primary key that is referenced by the other tables as a foreign key. These tables are created as models using Django and .csv files are used to populate the database. We provide contributors with a template .csv file that can be downloaded to ensure proper formatting. When a request is made to contribute to the database the user is prompted to log in. After a successful log in, the .csv file that is being uploaded is verified for proper format and the user is alerted to the success or failure of the upload. After the upload, an administrator needs to approve the data before it is marked as verified. This makes it easy for the user to convey their information to our database, and in turn other professionals.

The database is laid out into four tables. The `summary_info_id` table takes in information from the other three tables in order to provide search results for the website. It is easy to go back and forth between the tables using foreign keys and Django queries. For each search term that is provided, a query on the column associated with the term is

made, and the result is stored in a `QuerySet` variable. A reverse lookup with foreign keys is performed to append search results to the final `QuerySet`.

Data can be easily transferred from the backend to the frontend using JSON (JavaScript Object Notation). JSON is a standardized data format that is obviously supported natively by JavaScript, but also is supported well by Python and Django. When Django fetches data from the database, each database record is stored as a dictionary in a list. This list of dictionaries is then passed to the HTML template for building the search results page, and each dictionary is printed to the web page in JSON format using a built-in Django filter. The browser code is then able to read the JSON data from the page contents and store the results as an array in browser memory. From there, the array is exposed as a global variable to other functions that handle plotting, mapping, and file downloads.

The process for sending data from the frontend to the backend is very similar. The user selects the filled-in CSV template using the browser's file selection dialog. The CSV is converted to JSON by removing extraneous columns not specified in the template (that may have been added by accident), as well as removing empty values. The resulting JSON array is then converted to a string, which is sent to the server in a HTTP POST request, along with the user's name and email. When the server receives the request, the stringified JSON is converted to a list of dictionaries using the `json.loads` function in Python. This data is then inserted into the database by wrapping the operations in an atomic transaction. If there is an error inserting one row of data (for example, by violating the unique constraint on latitude and longitude pairs), the entire upload fails, and the user is sent an error response. Below is a piece of code showing how the JSON is handled in Python code.

```
@transaction.atomic
def upload(request):
    data = json.loads(request.POST["data"])
    for row in data:
        summary = summary_info_id(**row["sum"])
        summary.save()
        morpho = landslide_morphometrics(**row["morpho"], landslide=summary)
        morpho.save()
        metrics = landslide_metrics(**row["metrics"], landslide=summary)
        metrics.save()
        meta = meta_table(**row["meta"], landslide=summary,
                          contact_name=request.POST["name"], contact_email=request.POST["email"])
        meta.save()
```

Figure 6. Example of code using json to populate database

5 - Quality Assurance

In order to provide a higher level of confidence in the quality of our product, we made extensive use of built-in features that are available in the third-party tools our project uses. Django offers several security measures against attacks such as cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF). This means that malicious users who attempt to embed content from another website or attempt to execute unintended SQL code on our database will be prevented from doing so. In addition, Django ships with several APIs that provide abstractions of lower-level operations in pure Python code. The model API allows us to define our database tables as Python classes, and from there can execute database queries, updates, and deletes on instances of those classes. The form API similarly allows us to define website forms as Python classes and can then provide methods by which the form data can be validated. Both APIs make the process of database interaction and form interactions less error-prone, as it does not require us to write any SQL or HTML code, respectively. While Django offers framework-level quality-assurance, TypeScript offers language-level guarantees of correctness through its static type-checking system. Running the type-checker allows us to be sure that all our variables and function parameters are of the right type and warns us of any potential errors at compile time before they occur at run time.

In addition to third party tools, we also make use of unit and integration testing for our application. As there is not a lot of input/output involved in the application, the unit testing aspect focuses on the handling of data files. Firstly, we must ensure that data present in the database is formatted properly and with the correct data when file downloads are generated. Secondly, we must ensure that user-provided file uploads are rejected if they do not contain the required columns and are otherwise accepted. We did not have an automated system for integration testing, instead collaborators should regularly pull and run the code on their local machines to ensure that nothing is broken by recent changes. This helps maintain cross-platform compatibility for development. Even though the production application will run in a Linux environment, we want to make sure that developers on both Windows and Linux are able to contribute.

Another manual method of quality assurance that we have included are code reviews, which are conducted continuously throughout the workday. Keeping everyone on the same page as this makes the code easier to maintain and makes us more likely to detect bugs when they first occur, rather than letting them go undetected for a long time. We also get regular feedback from our client to ensure that the product is meeting his specifications and is of acceptable quality. All these factors together allow us to be sure

that the product we have made is the product that we set out to make at the beginning of this field session.

6 - Results

While we worked to get as much done as possible, the limited amount of time available to us made it so that some goals could not be accomplished. One feature that was not implemented is a full user account system, which entails sign up, log in, and profile pages, as well as database tables to store user data. We also did not add the ability for plotting data to be changed by the user, and the ability to show a trendline. The limited dataset we were given made it hard for us to test the quality of our plotting tools, and in the end, we decided to not implement this functionality.

Despite these uncompleted features, we were able to complete the majority of the tasks required to lay out the groundwork for the website. Most importantly, we made it possible to search the database on several different columns, including the name of the landslide, the geographical location of the landslide, the name of the contributor, and the upload date. In addition to that, we added support for authenticated users to upload data to the website, and for new uploads to be verified by an administrator. The final major feature that was added by us enables website contributors to be recognized by displaying their information on the website (all users must agree to this when they upload data). Screenshots of these features on the website can be found in the Appendix.

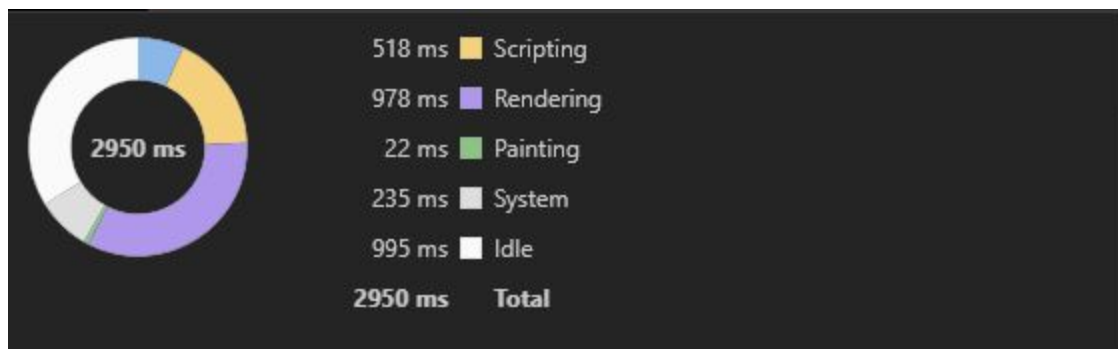


Figure 7. Chrome DevTools performance test.

The above figure shows a performance test for loading database entries containing “Canada” in the name, which includes over 1000 rows. The idle time corresponds to time that was spent by the server processing the request, all other timings are client side. 3 seconds per 1000 rows is a reasonable level of performance, and hopefully scales linearly, we did not have enough data to test this. The application was tested on multiple desktop browsers, including Chrome, Firefox, and Edge. The application was not tested

for compatibility on mobile devices, because it is primarily focused towards research, and most research is done using a desktop or laptop computer.

There are several different avenues for future work on this project. Naturally, the features we did not implement would be a great place to start, but there is the possibility to add completely new features, like email notifications that can be sent to administrators when there is new data that needs to be approved. It could also be possible to improve the compatibility of the website with mobile devices, as today mobile device usage is continually getting more common. Another aspect to investigate for future development would be additional analysis tools such as histograms and statistics tools.

Finally, we do have a couple lessons that have been learned from working on this project. First, it has become apparent that working with the IT department to set up a server on the school network can be a bit of a pain and is a much more painless thing to handle yourself. Additionally, we found the agile software development methods have diminishing returns when working in a smaller group, since we were able to stay in contact most of the time while we were working. One positive lesson that we learned is that abstractions, particularly the ones provided by Django, are useful in simplifying the process of web application development. Being able to do most of the backend code in pure Python is more straightforward and less prone to errors, which is important when creating an application for a production environment. Overall, working on this project has been a valuable experience and we are glad to have helped in furthering research in the scientific community.

7 - Appendices

7.1 - Appendix I - Website - Viewer Page

Search Data

Name: Contributor:

Starting Latitude: Ending Latitude:

Starting Longitude: Ending Longitude:

Starting Upload Date: Ending Upload Date:

Age Period:

Data Viewer

<input type="checkbox"/>	Name	Summary
<input type="checkbox"/>	Canada East Coast - West Levee of Western Laurentian Valley	View Summary
<input checked="" type="checkbox"/>	Canada West Coast - Woodfibre Creek Delta Failure	View Summary
<input type="checkbox"/>	Canada West Coast - Fraser River Delta Failure	View Summary
<input checked="" type="checkbox"/>	Canada West Coast - Kitimat Arm Failure 1975	View Summary

Here you can see how we set up search fields in order to query the database. It also shows a few search results. If you type in search terms into fields, the database is queried for those terms and append the results. So, the results in this case would be all data from Canada as well as all data from the Holocene period.


7.2 - Appendix II - Website - Contributions Page

Thank you for showing interest in contributing to the S4Slide Database. If you wish to upload data, you must be signed in with Google.

In order to facilitate data storage, file uploads must be in a consistent format. A CSV template with column headers [can be downloaded here](#). Please fill them in to the best of your ability. Further details on the schema [can be found here](#).

Only the latitude and longitude columns are required, all other columns are optional, however, please try to include a name for every row, as this makes database entries easier to search for. Any columns that are completely empty are ignored, and any additional columns that are included will be ignored.

All newly uploaded entries will be marked as unverified by default. An administrator will mark an entry as verified if it is deemed acceptable. Failure to include pertinent data, like an entry name, may result in the entry being removed. Repeated upload of bad data may result in your Google account being banned from this website.

 Signed in

Navigating away from this page, reloading the page, or closing the page will sign you out.

No file chosen

By clicking submit, you agree to have your name and email associated with the file upload.

Below is a listing of the top contributors to this website. Only verified entries count as contributions.

Website Contributors

Number of Contributions	Contact Name	Contact Email
1164	G. Lintern, J.Rutherford	kim.conway@canada.ca, gwyn.lintern@canada.ca
41	G. Lintern, J.Rutherford	john.shaw@canada.ca
21	Lucas Kitaev	lkitaev@mymail.mines.edu
16	G. Lintern, J.Rutherford	mriedel@geomar.de
3	G. Lintern, J.Rutherford	gordon.cameron@canada.ca
3	G. Lintern, J.Rutherford	gwyn.lintern@canada.ca
1	A. Normandeau	alexandre.normandeau@canada.ca
1	G. Lintern, J.Rutherford	brian.todd@canada.ca
1	G. Lintern, J.Rutherford	calvin.campbell@canada.ca
1	G. Lintern, J.Rutherford	gordon.cameron@canada.ca

This shows the contribution page for a signed in user. The left column shows a listing of the current contributors in the database, and the left column allows the user to select a file to upload.

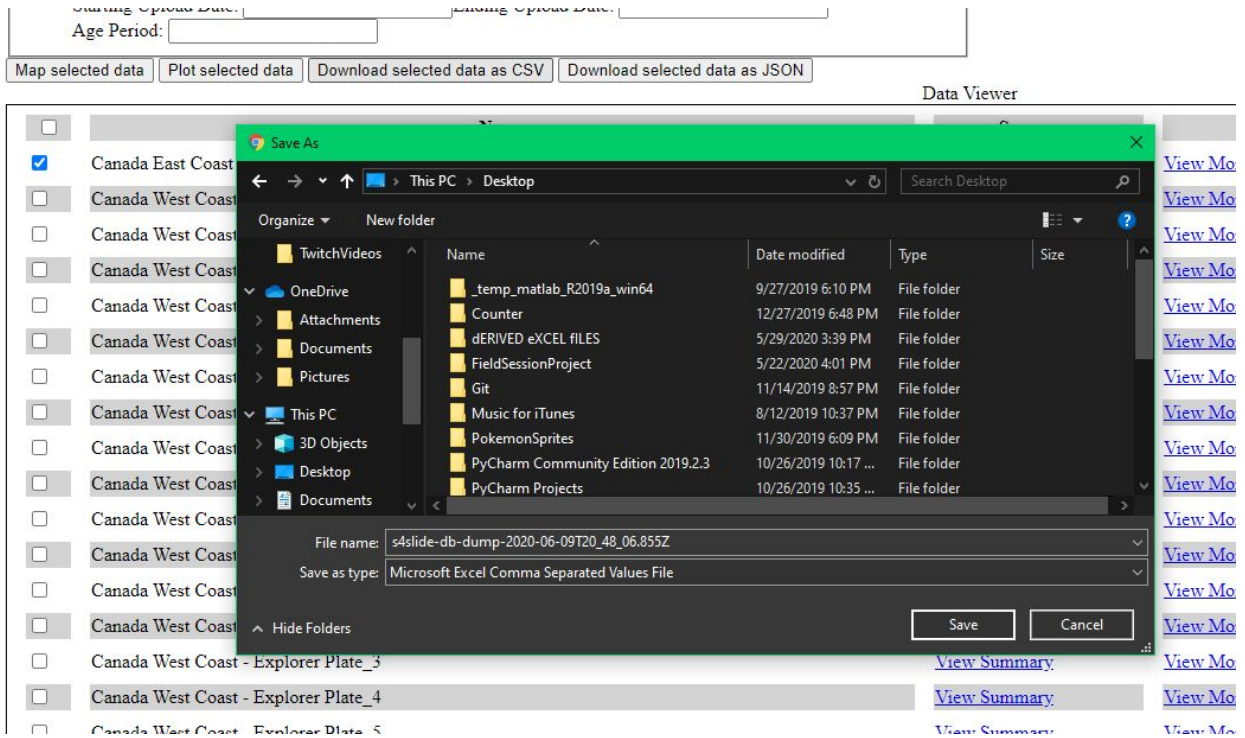
7.3 - Appendix III - Website - Administrator Access

The screenshot displays the Django administration interface for the 'meta_table'. At the top, it shows 'Django administration' and 'WELCOME LUCAS: VIEW SITE / CHANGE PASSWORD / LOG OUT'. Below the navigation bar, there's a breadcrumb trail 'Home > App > Meta_tables' and a date range '2020 June 4 June 5'. The main content area is titled 'Select meta_table to change' and features an 'Action:' dropdown set to 'Verify selected rows' and a 'Go' button. A table lists various data sources, each with a checkbox for selection. The table columns are LANDSLIDE, DATA TYPE, DATA SOURCE, CONTACT NAME, and CONTACT EMAIL. A 'FILTER' sidebar on the right allows filtering by 'verified' status (All, Yes, No). A '+ ADD META_TABLE' button is also visible.

<input type="checkbox"/>	LANDSLIDE	DATA TYPE	DATA SOURCE	CONTACT NAME	CONTACT EMAIL
<input type="checkbox"/>	Arctic Canada West - Beaufort Sea Slump		Amundsen 2004-004: Beaufort Sea / Amundsen Gulf / Northwest Passage, June 23- August 27, 2004	G. Lintern; J.Rutherford	0
<input type="checkbox"/>	St Lawrence - Betsiamites - Lower St. Lawrence			G. Lintern; J.Rutherford	0
<input type="checkbox"/>	Canada East Coast - Newfoundland_1449		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1450		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_738		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_739		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_740		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1156		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1157		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1159		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1160		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1162		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1787		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1888		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1917		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_1918		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_2355		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_2356		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_6768		Ned King excel data ???	G. Lintern; J.Rutherford	
<input type="checkbox"/>	Canada East Coast - Newfoundland_6771		Ned King excel data ???	G. Lintern; J.Rutherford	

The built-in Django administration interface is shown here. Selecting a specific time period will include only uploads that were added during that time. It is also possible to filter by verified and unverified uploads. Using the checkboxes on the left enables the administrator to perform mass verification/unverification and deletion of rows.

7.4 - Appendix IV - Website - Downloading Data



Once you select some data, you can click the “Download selected data as CSV” or “Download selected data as JSON” and a window pops up prompting you for a location on where you would like to save it.

7.5 - Appendix V - Links for Reference

GitHub - <https://github.com/zanejobe/S4SlideDB>

Django <https://docs.djangoproject.com/en/3.0/>

Leaflet <https://leafletjs.com/reference-1.6.0.html>

D3.js <https://d3js.org/>

TypeScript <https://www.typescriptlang.org/index.html>

General <https://www.w3schools.com/>