ROB CARSON QUESTIONS ON THE MARLOWE CENSUS

1. The top-level folder is called "marlowecensus" and the main subfolder inside it is also called "marlowecensus." Is this a normal thing for a Django project? Is it maybe even a requirement? Or would it be fine to give them different names in the interests of clarity?

- Yes, Rob, it's a common convention in Django projects to have the top-level folder (project root) and the inner folder (project settings directory) share the same name, as created by django-admin startproject marlowecensus. The top-level marlowecensus is the project root, and the inner marlowecensus contains the project-wide settings like settings.py. However, it's not a strict requirement—you can rename either folder for clarity (e.g., marlowe_project for the root and marlowe_config for the settings directory) as long as you update the references in files like manage.py and wsgi.py to point to the new module name (e.g., marlowe_config.settings instead of marlowecensus.settings).

2. I'm also not clear what the difference is between a "project," an "app," and a "package." Is the top "marlowecensus" maybe a project folder, and the "marlowecensus" contained within it may be a package or an app?

- You're on the right track, Rob! Let me clarify the terms. In Django, a **project** is the entire application, encompassing all settings, apps, and configurations. Here, the top-level marlowecensus folder is the project root, and the inner marlowecensus folder is the project settings directory—a Python **package** (because it has an __init__.py file) that contains project-wide configurations like settings.py and urls.py. It's not an app, though. An **app** is a modular component of the project that handles specific functionality, like census in this project, which has its models.py, admin.py, and migrations/ folder. So, to summarize: the top marlowecensus is the project root, the inner marlowecensus is a package for project settings, and the census is an app within the project.

3. Can you remind me why there are two parallel "static/census/" folders, one in the top level and one in the lower level? Does it maybe have something to do with one being used in development and one in production? (Louis explained this situation to me once, but I've forgotten the answer.)

- The two static/census/ folders exist because of Django's static file handling. The lower-level marlowecensus/census/static/census/ folder is part of the census app and contains the app's static files (like modal.css and style.css).
  - In development mode, Django automatically finds and serves these files from each app's static/ directory. The top-level marlowecensus/static/census/ folder is created when you run python manage.py collect static, which gathers all static files into STATIC_ROOT (typically set to the top-level static/ folder) for production use.
  - In production, Django expects a web server like Nginx to serve static files from STATIC_ROOT, so the top-level folder is used there, while the lower-level folder is used during development.

4. My recollection is that these two folders need to duplicate each other. Do they sync automatically? If I were to make changes to either one of them, would I need to make changes to the other manually, or does one of them repopulate the other?

- The two static/census/ folders don't sync automatically in real-time, but they are related through Django's collectstatic command. The lower-level marlowecensus/marlowecensus/static/census/ folder contains the source static files, and the top-level marlowecensus/static/census/ folder is populated by running python manage.py collectstatic, which copies all static files into STATIC_ROOT for production. If you make changes to the lower-level folder, they'll be visible in development immediately, but you'll need to rerun collectstatic to update the top-level folder for production. You should <u>never edit the top-level static/census/ directly</u>, as it's managed by collectstatic and will be overwritten. The process is one-way: the lower-level folder repopulates the top-level folder when you run collectstatic.

5. Do we need to keep the "census" folder as a layer inside these two "static" folders? Since it's the only thing in there, could we eliminate a layer for clarity and just put the contents directly in the "static" folder?

- The census folder inside the static/ directories (e.g., marlowecensus/marlowecensus/static/census/) acts as a namespace to organize static files, preventing conflicts if you have multiple apps with similarly named files (e.g., census/style.css vs. blog/style.css). In a typical Django project, this folder would be inside a census app (e.g., marlowecensus/census/static/census/), but in your project, the census app's files are directly inside marlowecensus/marlowecensus/, so the path is marlowecensus/marlowecensus/static/census/. Django's static file finder expects this namespacing structure, but it's not strictly required. Since the census is currently the only namespace in static/, you could eliminate the census/ layer by moving its contents directly into static/ (e.g., marlowecensus/marlowecensus/static/modal.css), which would simplify the structure and make template paths shorter (e.g., {% static 'modal.css' %} instead of {% static 'census/modal.css' %}). However, this increases the risk of naming conflicts if you add more apps later (e.g., a blog app with its static/modal.css), so it's generally better to keep the census/ namespace for safety and convention unless you're certain you won't need it.

6. Similarly, do we need the "census" folder inside the "templates" folder? Is there a reason to keep the html files for the main pages of the site separate from the html files for the error messages? And if so, does this "census" folder need to share the same name as the "census" folders in the two "static" folders?

- The census folder inside templates/ (e.g., marlowecensus/marlowecensus/templates/census/) is a namespace to organize templates, preventing conflicts if you have multiple apps with similarly named templates (e.g., census/index.html vs. blog/index.html). In a typical Django project, this folder would be inside a census app (e.g., marlowecensus/census/templates/census/), but in your project, the census app's files are directly inside marlowecensus/marlowecensus/, so the path is marlowecensus/marlowecensus/templates/census/. It's a best practice to keep this namespace, but you could move the templates directly into templates/ (e.g., marlowecensus/marlowecensus/templates/about.html,….) if the census is the only namespace—just be aware of potential conflicts if you add more apps. Regarding the main pages and error pages (400.html, 403.html, etc.), there's no strict need to separate them, but it can be clearer to move error pages to a project-wide templates directory (e.g., marlowecensus/templates/400.html) since they're typically used across all apps, while main pages are app-specific. Finally, the census folder in templates/ doesn't need to share the same name as the census folders in static/—their names are the same because they both organize files for the census functionality, and it's a good convention to keep them aligned for clarity.

7. Is there a reason that style.css is not inside the "css" folder, which seems to me like it would be the more natural place to put it? And similarly, is there a reason that the "import_export" folder isn't inside the "js" folder with the other JavaScript add-ins? (I suppose it has a .css file of its own — but could that also go inside "css" or should it stay bundled up with the JavaScript files?)

- The placement of style.css at the top level of static/census/ (e.g., marlowecensus/marlowecensus/static/census/style.css) instead of inside static/census/css/ is likely a developer choice, possibly to make it the primary stylesheet and easier to access in templates (e.g., {% static 'census/style.css' %}). However, there's no strict reason for this, and it would be more conventional to move it into static/census/css/ alongside modal.css for better organization—just update the template paths to {% static 'census/css/style.css' %} if you do. As for the import_export/ folder, it's not inside js/ because it's managed by the django-import-export library, which uses import_export/ as its namespace for static files (like import.css). These files need to stay in import_export/ to maintain the library's expected structure, even though they include CSS files that might seem to belong in a css/ folder. This is standard practice for third-party Django apps, which bundle their CSS and JavaScript files under their namespace to avoid conflicts.

8. Finally, can I confirm that the higher level "static/admin/" folder has no original code in it and is part of the Django framework? This is the code that runs the Admin side of the website, right?

- Yes, Rob, you're correct! The top-level static/admin/ folder contains static files for Django's admin interface, provided by the django.contrib.admin app and it has no original code written by the project's developers. These files (CSS, JavaScript, and images) are copied into STATIC_ROOT when you run collectstatic, and they originate from Django's source code (e.g., site-packages/django/contrib/admin/static/admin/). They provide the styling and interactivity for the admin side of the website, such as the interface at http://your-site/admin/, which is powered by django.contrib.admin. The Python logic for the admin is in django.contrib.admin, but these static files handle the front-end presentation.