

The Port For Real This Time

Technical Journal Entry Begins

Hail Mary at Porting Apps Tonight

Ugh, it's weeks later and I still haven't ported the 2 key apps. But that's okay. I do it tonight. I go late and long if I have to. And tomorrow, I have my first client call that I'm managing myself in a long while. So I can do both things without screwing up, right? Right!

The Art of AI-Enabled Problem Solving Is Evolving Quickly

Everything becomes easier as you get more adept at AI-wrangling, right? It's just like being an SEO or Webmaster before it. There are no hard and fast rules. Some people might think so or try to tell you so, usually to make money off of you some way or another, but all best are off. The deck is being reshuffled, but thankfully some skills will shine through and serve you in any tech-enabled age.

I've Found My Creative Spark Again Through Tiny Details of New Era

I've fallen back in love with those tiny details. I thought I might have been burnt out, but I was challenged by my boss to to the best work of my life. And with AI reshuffling everything the way it is, it's really when I'm at my best. And I'm coming up from a rather deep dive on recalibrating the new reality and aligning my skills just so, and tweaking my tool-set.

The iron is hot and it's time to strike quickly, shaping and tempering the stuff of a new generation into new tools and secret weapons for SEO. Just when I thought I was out, they pull me back in! And happily, things are interesting enough again to infuse another decade or so of love into it. So, let's hit it.

Making Work a Show: Connecting Old and New Skills

Tomorrow, it's back to managing meetings and such, but I have to make it a show. And the key to making it the magic show (with the substance behind it) that makes it so enjoyable for me and everybody else, is to connect the dots of my new work with my old job. Bring my new skills and new tools and new shows to bear every day. Let the word and the wonder spread of this new thing, this way Mike works that is just... just... well, words *should* fail.

The Port Awaits a Fast and Eventful Few Hours Ahead

And that's why the port is going to be fun. I'm going to blast through it the best and as fast as I

can. We're talking a few hours, as opposed to the few weeks it took me to get this far. 1, 2, 3... 1? ...is how it goes. That's the language I use. I'm always on step #1, because you can't do anything more than what's immediately before you. The now-moment is everything... and now... and now... methinks App #1 in the app placeholder section is... set token!

I'm not going to make the same mistakes I made getting to this point, which is leaving choices too open-ended for the AI. In the state they're in today, they are only too anxious to lead you down deep, winding rabbit holes from which there is no return—at least not in the same night and not without frustration and exhaustion. To keep frustration and exhaustion at bay by not going rabbit-hole-diving, one must attach some rappelling rope at the top of the abyss and be ready to pull yourself up at a moment's notice. And so we do that.

Switching From ZeroMQ to WebSockets for Real-Time Communication

I have two code-bases. One was my first web app implementation under FastHTML, and only barely at that. For interprocess communication, I used the still amazing ZeroMQ, aka Zero-wait Message Queue, or just zmq for short. It was big in the Python world and made it around to lots of other platforms for its timeless inherent cross-domain value, like embedding RegEx or SQLite. However, an alternative industry standard called WebSockets has really taken off, and despite their different use-case sweet spots, one does not use WebSockets anymore. Everything that chats uses them, and it's that distinct real-time difference you feel in ChatGPT, Perplexity, Anthropic and all the other chatty user interfaces that are copying each other these days. While you could make things tighter with zmq if it's all on the same physical hardware, why would you when you could be learning what the world uses to write chat apps? So a big part of the porting and refactoring is unwiring things from zmq and rewiring them to websockets.

Using Small Databases for Smaller Applications Is Totally Adequate

The other major difference is the way database stuff works. Even calling it database does it a disservice because it makes you think **big** stuff like a SQL server of some sort like PostgreSQL, Redis or MySQL. Or a NoSQL database like MongoDB or Cassandra. These things are built to support big enterprise infrastructures. But sometimes you just want a database to replace browser cookies. If you have total control of the server, why not just keep track of who the user is client-side with a session-id and do all your database stuff on the server in an extremely lightweight fashion with SQLite or even reading and writing text files.

Using JSON Files for Text Persistence with a Library Approach

If you read and write textfiles, the best candidate is using a library that treats it as a JSON data structure so that your library takes care of all the API-work making you feel like it's not just a textfile. And so, that's what I did to achieve a sort of cookie-like persistence on the server. I read from and wrote to a JSON file on the drive. It's a great strategy if you're not using FastHTML, which I am, which wraps SQLite in this nifty little MiniDataAPI spec wrapper making it even

easier than SQL. It's an interoperable API layer that almost any database or database-like thing can be wired to, so you can swap out the actual physical database mechanism without changing your code. And so another part of this port is rewiring these bits from the JSON file to FastHTML's MiniDataAPI.

And then finally, there's the details of the app itself, which involves...

- Getting the user to find and input their Botify API token
- Helping the user find a Botify Project URL (reminder to self: find a slicker way later)
- Present a list of Crawl Dates, which we call Analysis Slugs (analyze any interesting slugs lately)
- Using the user selections to perform several API-calls to Botify
- Allow a long-running task to occur, informing the user when it's done even if they surf away
- Allowing multiple of these long-running tasks to be performed at the same time.
- Automatically downloading the file produced as a result of these API calls and long-running jobs.
- Hosting a "secondary" web server that serves up these downloads "funneling" the request through the querystring API of another service.

Pshew! This would all sound intimidating if it weren't actually already done. And so to make it fun and interesting, it has to be as Pipulate/Botifyon-like as possible. And that means lots of LLM help and fun.

- We need a home message. Set the stage for the magic.
- We need a 1-page-application that does it all (App 1 on the menu)
- We need a token acquisition experience.

Creating a New App: Breaking Free from Conventional Wisdom

Okay, let's start there. The rest falls into place as the momentum gets rolling. But I see that I don't want to feel constrained by keeping a main app template intact and dancing around it frightened to break it and all that nonsense. I want to go into it like a bull in a china shop and just slam this app out. And that means... that means... that means I may be intentionally git branching for the first time in my life. I hate branches and forks. I'm a linear non-linear thinker Mr. Unicorn, so leave me alone.

Okay... this looks like a job for... git cloning for starters. This thing works out of whatever directory, so let's give it a strong new identity tied to the app. All the automatic white-label rebranding tricks I designed into it will kick in. That's how it starts out Pipulate and becomes Botifython. But now due to the Link-graph nature of the work, it's going to be Botifymograph! It's a bit long of a word, but hey, it'll help put this whole thing through the paces. A stress-test, if you will.

Already, I hit a snafu. Because proprietary stuff outside the Pipulate main codebase is going to be used here, I broke it off into a whole other GitHub account and even disconnected the fork so that I could make it private. And as such, I have multiple SSH keys to manage in my ~/.ssh/ folder, and I have to actually distinguish between and use particular ones on particular actions, which is about one of the most arcane and nonsensical APIs in existence. Doing this is what AI's are made for.

SSH Keys to Success

MikeLev.in: I have a ~/.ssh folder with various key pairs that I use for different GitHub accounts. Normally, it defaults to using my main id_rsa key pair, but I need to do an ssh protocol git clone telling it to use a particular key. The repo is <https://github.com/Botifython/botifython> and the keys are named botifython_develop and botifython_develop.pub. All the permissions are already set up GitHub-side and botifython_develop.pub is registered as a full permission developer. How can I first, do the git clone in ssh mode to download it locally a folder named botifymograph? And second, how can I change it's main origin so that all subsequent git pulls and pushes use the right key and origin without requiring any special git command modifiers?

Perplexity: To accomplish your tasks, you'll need to follow these steps:

1. Git Clone with SSH Using a Specific Key

To clone the repository using SSH with a specific key, you can use the following command:

```
GIT_SSH_COMMAND='ssh -i ~/.ssh/botifython_develop' git clone  
git@github.com:Botifython/botifython.git botifymograph
```

This command does the following: - Sets the GIT_SSH_COMMAND environment variable to use your specific key - Clones the repository using the SSH URL - Renames the local folder to "botifymograph"

2. Changing the Origin to Use the Right Key

After cloning, you can set up a custom SSH configuration to always use the correct key for this repository. Here's how:

1. Edit your SSH config file:

```
nano ~/.ssh/config
```

2. Add the following configuration:

```
Host github-botifython  
  HostName github.com  
  User git  
  IdentityFile ~/.ssh/botifython_develop
```

3. Save and close the file.

4. Change the repository's origin URL:

```
cd botifymograph  
git remote set-url origin git@github-botifython:Botifython/botifython.git
```

Now, all subsequent git operations (pull, push, etc.) will automatically use the correct key without requiring any special modifiers.

Additional Notes

- Ensure that your botifython_develop private key has the correct permissions (typically 600).
- If you're using multiple GitHub accounts, this method allows you to manage them separately without conflicts.
- Always verify that you're using the correct key by testing with `ssh -T git@github-botifython`.

By following these steps, you'll have successfully cloned the repository and set it up to use the specific SSH key for all future operations.

Hmmm. The fancy schmancy `GIT_SSH_COMMAND=` version isn't working, and so...

MikeLev.in: I have this entry in my `~/.ssh/config` file:

```
# Secondary GitHub Account
```

```
Host bf
```

```
  HostName github.com
```

```
  User git
```

```
  IdentityFile ~/.ssh/botifython_develop
```

```
  IdentitiesOnly yes
```

...give me instructions based on that.

Perplexity: Based on the SSH configuration you provided, you can use the following instructions to clone the repository and set up the origin for future operations:

Cloning the Repository

To clone the repository using the specified SSH key, use this command:

```
git clone bf:Botifython/botifython.git botifymograph
```

This command will: - Use the SSH configuration for the host "bf" - Clone the repository from GitHub - Create a local folder named "botifymograph"

Setting Up the Origin

After cloning, you don't need to change the origin URL because your SSH config is already set up correctly. The origin will automatically use the right key for all git operations. However, you can verify the remote URL with:

```
cd botifymograph
```

```
git remote -v
```

If for any reason you need to set or update the origin URL, use:

```
git remote set-url origin bf:Botifython/botifython.git
```

Verifying the Setup

To ensure everything is working correctly:

1. Try pushing a test commit:

```
touch test.txt
git add test.txt
git commit -m "Test commit"
git push origin main
```

2. Or pull from the repository:

```
git pull origin main
```

These operations should now use the botifython_develop SSH key automatically without any additional parameters.

Additional Notes

- Ensure the botifython_develop private key has the correct permissions (typically 600).
- If you're using multiple GitHub accounts, this method allows you to manage them separately without conflicts.
- You can test the SSH connection with: `ssh -T bf`

With this setup, all git operations for this repository will automatically use the correct SSH key, simplifying your workflow and eliminating the need for special command modifiers.

Citations:

[1] <https://locall.host/git-clone-with-a-specific-ssh-key/>

[2]

<https://support.circleci.com/hc/en-us/articles/21182136871835-How-to-use-a-Specific-SSH-Key-for-Git-Clone>

[3] <https://www.baeldung.com/linux/ssh-private-key-git-command>

[4] <https://ma.ttias.be/specify-a-specific-ssh-private-key-for-git-pull-git-clone/>

[5]

<https://stackoverflow.com/questions/4565700/how-to-specify-the-private-ssh-key-to-use-when-executing-shell-command-on-git/60530467>

[6] <https://www.warp.dev/terminus/git-clone-ssh>

[7] <https://graphite.dev/guides/git-clone-ssh-vs-https>

[8] <https://ralphjsmit.com/git-custom-ssh-key>

Oh yeah? Let's see...

```
[mike@nixos:~/repos]$ git clone bf:Botifython/botifython.git botifymograph
```

```
Cloning into 'botifymograph'...
```

```
remote: Enumerating objects: 712, done.
```

```
remote: Counting objects: 100% (329/329), done.
```

```
remote: Compressing objects: 100% (157/157), done.
```

```
remote: Total 712 (delta 233), reused 265 (delta 172), pack-reused 383 (from 1)
```

```
Receiving objects: 100% (712/712), 289.92 KiB | 7.83 MiB/s, done.
```

```
Resolving deltas: 100% (453/453), done.
```

```
[mike@nixos:~/repos]$ cd botifymograph/
```

```
[mike@nixos:~/repos/botifymograph]$ git remote -v
origin  bf:Botifython/botifython.git (fetch)
origin  bf:Botifython/botifython.git (push)
```

```
[mike@nixos:~/repos/botifymograph]$ git remote set-url origin bf:Botifython/botifython.git
```

```
[mike@nixos:~/repos/botifymograph]$ touch test.txt
git add test.txt
git commit -m "Test commit"
git push origin main
[main da88f96] Test commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 48 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 272 bytes | 272.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To bf:Botifython/botifython.git
   fb9ff2b..da88f96  main -> main
```

```
[mike@nixos:~/repos/botifymograph]$ git pull origin main
From bf:Botifython/botifython
 * branch      main      -> FETCH_HEAD
Already up to date.
```

```
[mike@nixos:~/repos/botifymograph]$
```

Well, I guess so! One hurdle overcome. And it's barely 8:30 PM. I've got the whole night to go!

Using Nix Flake Variable to Control White Labeling

Think! It's these little hurdles that should accelerate and not hinder. With a new folder in-hand, we get to nix develop it for the first time. Success! It didn't white-label as cleanly as I thought as I have a projectName variable in the flake.nix file. So since this will be a branch, I want it to be gloriously visually different so I never make the mistake of being on the wrong branch with a client or anything, so I change the variable name and delete the whole .venv folder. For good measure, I also do a nix flake update before I nix develop again.

Git Move File to Maintain Git History

Okay interesting! When you do that, the command to launch the app changes. I always have an alias that doesn't forcibly pop open a browser window, which I'm keeping bf no matter how you relabel it. But when you do change projectName in flake.nix, you also have to rename the file it's

going to try to run, in this case from botifython.py to botifyfymograph.py. And because I don't want to lose the git histories, I do it with a git command:

```
git mv botifython.py botifyfymograph.py
```

Pseudo-Templates Offer Freedom for Rapid Development

And this has the extra benefit that if I ever do merge branches, the filenames won't collide and I won't have a big mess of a file to sort out. Multiple "server.py" files can happily live in the same folder, so I'm making a very good pseduo-template system. Not real templates, because it's all working off of a single file which you edit away however you like. So in a sense you start from a template, but then it can radically diverge. But not too much, or it will all break. This has pluses and minuses as all things do. If I make innovations I want to roll into the main branch, I'm going to have to carefully transplant them. But the freedom in the immediate term is well worth it. Okay, forge ahead! We need to start wedging in stuff. The homepage has to notice that you don't have the Botify API token yet and pester you to provide it. I literally could have it pester you with the local LLM, but this is precisely the sort of rabbit hole I need to avoid to pull this thing off tonight. I've been psychologically avoiding this next round of customization, because it really turns Botifython (or Botifyfymograph in this case) into something very distinct from Pipulate. Oh, I really should have branched before I renamed the file. But anyway...

MikeLev.in: How do I make a new git branch?

Perplexity: To create a new Git branch, you have several options:

Basic Branch Creation

The simplest way to create a new branch is:

```
git branch <branch-name>
```

This creates the branch but doesn't switch to it[1]. To switch to the new branch after creation, use:

```
git checkout <branch-name>
```

Create and Checkout in One Step

A more common and efficient method is to create and switch to the new branch in one command:

```
git checkout -b <branch-name>
```

Or, using the newer git switch command:

```
git switch -c <branch-name>
```

These commands create a new branch and immediately switch to it[2].

Creating Branches from Specific Points

You can also create branches from specific commits, tags, or other branches:

- From a tag: `git branch <branch-name> <tag-name>`
- From a specific commit: `git branch <branch-name> <commit-id>`
- From a different branch: `git checkout -b <new-branch-name> <existing-branch-name>`

Best Practices

- Choose descriptive branch names that reflect the purpose of the branch[2].
- Adhere to your team's branch naming conventions for consistency.
- Regularly list your branches using `git branch -a` to keep track of all local and remote branches[1].

Remember, creating a branch is a lightweight operation in Git, so don't hesitate to create branches for experimenting or isolating feature development[4].

Citations:

[1]

<https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Git-Branch-Create-Example-Command-Checkout-Commit-Tag>

[2] <https://phoenixnap.com/kb/git-create-new-branch>

[3]

<https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-and-deleting-branches-within-your-repository>

[4] <https://www.atlassian.com/git/tutorials/using-branches>

[5] <https://www.youtube.com/watch?v=Lf3DYRvCPFo>

[6]

<https://stackoverflow.com/questions/66882952/how-to-create-new-local-branch-and-switch-between-branches-in-git>

[7] <https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

[8] <https://www.git-tower.com/learn/git/faq/create-branch>

```
[mike@nixos:~/repos/botifymograph]$ git branch botifymograph
```

```
[mike@nixos:~/repos/botifymograph]$ git checkout botifymograph
```

```
A  botifymograph.py
```

```
D  botifython.py
```

```
M  flake.lock
```

```
M  flake.nix
```

```
Switched to branch 'botifymograph'
```

```
[mike@nixos:~/repos/botifymograph]$
```

Confirmed!

Replacing Plugin App Menu Placeholders

Okay, so no forget about using the "home" page for the token. This is going to be one unified single-page application. You go to Link Graph on the menu, and the whole experience is going to be top-down much like a Jupyter Notebook, except when something is finished or dispensed with permanently, such as setting the API token, that "block" is going to disappear. So this is sort of a peek-a-boo module I'm creating that will appear if you don't have a token set and disappear if you do.

And now we enter Cursor AI.

Me: I need to change the name of App 1 (app_1) into Link Graph (link_graph).

Cursor: Certainly! I'll update the code to change "App 1" (app_1) to "Link Graph" (link_graph).

Here are the necessary changes:

// ... existing code ...

```
# Configurable constants for table names and endpoints
```

```
MENU_ITEMS = [todo_app.name, 'link_graph', 'app_2', 'app_3']
```

```
# Add this near your other configuration variables (around line 100-120)
```

```
APP_MESSAGES = {
```

```
    ": "Welcome to the home page!",
```

```
    todo_app.name: f"You've entered the {todo_app.name} area.",
```

```
    profile_app.name: f"You're now in the {profile_app.name} section.",
```

```
    'link_graph': "Welcome to the Link Graph!",
```

```
    'app_2': "Welcome to App 2!",
```

```
    'app_3': "Welcome to App 3!",
```

```
}
```

// ... existing code ...

This change replaces 'app_1' with 'link_graph' in the MENU_ITEMS list and updates the APP_MESSAGES dictionary to include a welcome message for the Link Graph.

Remember that this change only affects the menu item and the welcome message. If you have any specific functionality or views associated with the former "App 1", you'll need to update those as well. For example, if you have any conditions in your code checking for "app_1", you should update those to check for "link_graph" instead.

Also, make sure to update any templates or other parts of your application that might be referencing "App 1" or "app_1" to use "Link Graph" or "link_graph" respectively.

- ---

Apply to MENU_ITEMS
- Apply to APP_MESSAGES

Done and done. Save file. Visit app, and Voilà! Okay, let's go for a twofer...

Me: Change App 2 (app_2) to Gap Analysis (gap_analysis)

Remove App 3 (app_3)

Cursor: Certainly! I'll update the code to change "App 2" (app_2) to "Gap Analysis" (gap_analysis) and remove "App 3" (app_3). Here are the necessary changes:

// ... existing code ...

```
# Configurable constants for table names and endpoints
```

```
MENU_ITEMS = [todo_app.name, 'link_graph', 'gap_analysis']
```

```
# Add this near your other configuration variables (around line 100-120)
APP_MESSAGES = {
    ": "Welcome to the home page!",
    todo_app.name: f"You've entered the {todo_app.name} area.",
    profile_app.name: f"You're now in the {profile_app.name} section.",
    'link_graph': "Welcome to the Link Graph!",
    'gap_analysis': "Welcome to the Gap Analysis!",
}

// ... existing code ...
```

These changes do the following:

1. In the MENU_ITEMS list, we've replaced 'app_2' with 'gap_analysis' and removed 'app_3'.
2. In the APP_MESSAGES dictionary, we've updated the message for 'gap_analysis' and removed the entry for 'app_3'.

Remember that these changes only affect the menu items and the welcome messages. If you have any specific functionality or views associated with the former "App 2" or "App 3", you'll need to update or remove those as well. For example:

- Any conditions in your code checking for "app_2" should be updated to check for "gap_analysis" instead.
- Any functionality or views specific to "app_3" should be removed or commented out if you want to preserve the code for future use.

Also, make sure to update any templates or other parts of your application that might be referencing "App 2", "app_2", "App 3", or "app_3" to use "Gap Analysis" or "gap_analysis" respectively, or remove references to App 3 entirely.

- ---

Two Applies
- Two Accepts

...and this thing is now customized. Wow, it's amazing what a difference filling the real app names in does for making it feel real.

Renaming Customer to Client

Next, we relabel Customer to Client. Internally, the concept is actually Profile for invisibly wiring UI-stuff up, but I relabeled what displays to the user to Customer during template development to ensure that it was reliably relabelable. And I'm not going to trust this one to AI as it's too easy to misinterpret and break the app.

There's a magical mystery line which is one of Jeremy Howard's oddest decisions in designing the FastHTML system. Whereas the Flask API which all of this stuff is modeled after mainly has the app object as an instantiated instance of a Flask server, FastHTML has an app and rt instance. It uses a novel tuple object returner, so it follows the odd model:

```
app, rt = fast_app()
```

In other words, you just have to know by learning that calling fast_app, which itself is a convenience wrapper for other things, automatically **returns at least 2 objects**. Now when you're doing database work the parameters and arguments crammed within that open/close

parenthesis is going to be a whole bunch of stuff including the table name. In Jeremy's classic todo list example, 2 objects are returned:

```
app, rt (todos, Todo) = fast_app(<a bunch of stuff here>)
```

...or more thoroughly...

```
app, rt, (tasks, Task) = fast_app(
    "data/data.db",
    task={
        "id": int,
        "name": str,
        "done": bool,
        "priority": int,
        "profile_id": int,
        "pk": "id"
    },
)
```

My version very closely parallel's Jeremy's classic example showing up in his <https://fastht.ml> tutorials and YouTube videos. The thing to notice is the very precise naming convention here. Now a lot of things in FastHTML are loopy goosey and however you like. But if you want the clean database integration magic, you've got to go with the FastHTML flow.

And so that's a long way of saying that I'm changing...

```
app, rt, (store, Store), (tasks, Task), (customers, Customer) = fast_app(...)
```

...to...

```
app, rt, (store, Store), (tasks, Task), (clients, Client) = fast_app(...)
```

...and have to make the table match it for the UI to match. It will still work without renaming the table, but the UI takes the name from that. I would like the returned tuples to do all the dynamic renaming and re-endpointing work. I have a few more decouplings to do at some point, but 80/20-rule for now. Having to look a few places to rename (rebrand) such an important UI characteristic is a small price to pay.

```
app, rt, (store, Store), (tasks, Task), (clients, Client) = fast_app(
    "data/data.db",
    ws_hdr=True,
    live=True,
    hdrs=(
        AllTheJavaScript('sortable'),
        Script(type='module')
    ),
    store={
        "key": str,
        "value": str,
        "pk": "key"
    },
    task={
        "id": int,
        "name": str, # Changed from "title" to "name"
    }
)
```

```

        "done": bool,
        "priority": int,
        "profile_id": int,
        "pk": "id"
    },
    client={
        "id": int,
        "name": str,
        "address": str,
        "code": str,
        "active": bool,
        "priority": int,
        "pk": "id"
    },
)

```

Speaking of which you also need to do it in the code here:

```
# PLUGIN REGISTRATION
```

```
# Instantiate and register routes for the TodoApp plugin
```

```
# This is the plugin app to copy/paste/modify to create a new plugin
```

```
todo_app = TodoApp(table=tasks)
```

```
todo_app.register_routes(rt)
```

```
# This is the plugin app to leave alone as it is tightly integrated with navigation
```

```
profile_app = ProfileApp(table=clients) # <-- but you can still change the table name
```

```
profile_app.register_routes(rt)
```

```
# Aliases for table names. Variable names on the left never change.
```

```
# Variable names on the right must match your table name in the fast_app() call.
```

```
todos = tasks
```

```
profiles = clients
```

This reveals conceptually under the covers, I still use "todos" and "profiles" for hard-wiring in the code that's invisible to the user through the web interface. Sorting out the singular versus plural forms in all this... well, I'll leave that for a later time to explain (or maybe justify).

Customizing Main Plugin App Message

Okay, next we need to modify the placeholder message.

MikeLev.in: When the selected app is link_graph, change the placeholder message to tell the user this app is for visualizing the link graph of sites in Botify.

Cursor: Here's how we can update the code to change the placeholder message for the Link Graph app:

```
// ... existing code ...
```

```
def create_grid_left(is_todo_view, menux, todo_items):
```

```
    """
```

Create the left column of the main grid layout.

This function handles the content for the left side of the two-column layout. Plugins can integrate here to display their content in this area.

To add a new plugin at this level:

1. Add a condition to check if the current view is for your plugin
2. Create a Card or other container for your plugin's content
3. Use your plugin's `render_item` method to display individual items

```
return Div(
    Card(
        H2(f"{endpoint_fmt(menux)} {LIST_SUFFIX}"),
        UI(*[todo_app.render_item(todo) for todo in todo_items],
            id='todo-list',
            cls='sortable',
            style="padding-left: 0;"),
        header=create_todo_form() if is_todo_view else None,
    ) if is_todo_view else Card(
        H2(f"{endpoint_fmt(menux)}"),
        P("This is a placeholder for the selected application.") if menux != 'link_graph' else
        P("This app is for visualizing the link graph of sites in Botify."),
    ),
    id="content-container",
)
```

// ... existing code ...

This change modifies the `create_grid_left` function. When the selected app is not the todo view, it now checks if the `menux` is 'link_graph'. If it is, it displays the new message about visualizing the link graph of sites in Botify. For all other apps, it keeps the generic placeholder message. This update provides a more specific description for the Link Graph app while maintaining the existing behavior for other apps. Remember that this is still a placeholder - you'll need to implement the actual link graph visualization functionality when you're ready to do so.

Getting Botify API Token

Me: Okay, I like that. Thank you. Next, I'm going for a vertically stacked `Card()` design in the main area, aka `create_grid_left`. It should not replace but rather go underneath the Card containing the placeholder (and now app description), and its appearance should be controlled based on whether or not a file is present called `botify_token.txt` in the application folder.

Cursor: Certainly! I'll modify the `create_grid_left` function to add a vertically stacked Card design that appears below the existing Card, and its visibility will be controlled by the presence of a `botify_token.txt` file. Here's how we can implement this:

Note from Human: So, it really took a couple of back and forths to get the code I wanted, using `pathlib Path` instead of `os`, and getting the condition correct, but it was painless.

```
def create_grid_left(is_todo_view, menux, todo_items):
```

```
    """
```

Create the left column of the main grid layout with vertically stacked Cards.

This function handles the content for the left side of the two-column layout.

Plugins can integrate here to display their content in this area.

To add a new plugin at this level:

1. Add a condition to check if the current view is for your plugin
2. Create a Card or other container for your plugin's content
3. Use your plugin's render_item method to display individual items

```
    """
```

```
    main_card = Card(
        H2(f"{endpoint_fmt(menux)} {LIST_SUFFIX}"),
        UI(*[todo_app.render_item(todo) for todo in todo_items],
            id='todo-list',
            cls='sortable',
            style="padding-left: 0;"),
        header=create_todo_form() if is_todo_view else None,
    ) if is_todo_view else Card(
        H2(f"{endpoint_fmt(menux)}"),
        P("This is a placeholder for the selected application.") if menux != 'link_graph' else
        P("This app is for visualizing the link graph of sites in Botify."),
    )
```

```
    # Check if botify_token.txt exists
```

```
    token_file = Path('botify_token.txt')
```

```
    token_file_exists = token_file.is_file()
```

```
    # Create the additional card for the link_graph view
```

```
    additional_card = None
```

```
    if menux == 'link_graph':
```

```
        if token_file_exists:
```

```
            additional_card = Card(
```

```
                H3("Botify Integration"),
```

```
                P("Your Botify integration is active. You can now use the Link Graph features."),
```

```
                style="margin-top: 20px;" # Add some space between cards
```

```
            )
```

```
        else:
```

```
            additional_card = Card(
```

```
                H3("Botify Integration"),
```

```
                P("Botify integration is not set up. Please add your Botify token to use the Link Graph features."),
```

```
                style="margin-top: 20px;" # Add some space between cards
```

```
            )
```

```
    return Div(
```

```
        main_card,
```

```
        additional_card,  
        id="content-container",  
        style="display: flex; flex-direction: column;"  
    )
```

This updated version of `create_grid_left` does the following:

1. It keeps the existing main Card as before.
2. It checks for the existence of `botify_token.txt` in the application folder.
3. If the token file exists, it creates an additional Card with information about the Botify integration.
4. It returns a Div containing both cards (if the additional card exists), stacked vertically using flexbox.

The additional card will only appear if the `botify_token.txt` file is present in the application folder. This allows you to control the visibility of the Botify integration information based on whether the token file exists.

Remember to import the Path module at the top of your file if it's not already imported:
`from pathlib import Path`

This implementation provides a flexible structure that you can easily extend in the future. For example, you could add more cards or modify the content of the additional card based on other conditions or user interactions.

Me: Excellent! Now we need to show a PicoCSS style input field where the user can paste their token. It should have a submit button. You don't have to wire it up yet, but be prepared that this will preform an htmx call to an endpoint that will confirm API connectivity, write the textfile and conditionally on success replace the content of the Card with the username which will be the return value of hte API test. This is just to give you a heads up to the purpose and impending wiring of this form. For now, just add the input field suitable for this purpose.

Me: This is not a request for you to do anything. This is an article to give you context. After you absorb this, I'm going to feed you the old codebase again to get an overivw of what we're doing and to get your summary of how to prepare for next steps to feed back to you. For this article, all I want is an acknowledgment.

Cursor: I understand. Thank you for providing this detailed context about your project and the steps you've taken so far. It's clear that you're working on porting and refactoring a web application, moving from ZeroMQ to WebSockets, and adapting the database approach to use FastHTML's MiniDataAPI.

You've successfully set up a new Git repository for Botifymograph, using a specific SSH key for authentication. You've also renamed the main Python file and updated the Nix flake configuration to reflect these changes.

Your next steps involve implementing the Botify API token functionality and adapting the application to work with the new structure. You're right to be cautious about potential rabbit holes and to focus on the essential functionality first.

When you're ready to proceed, please provide more specific details about what you'd like to work on next, such as the token acquisition process or the home page message. I'll be here to assist you with code suggestions, implementation strategies, or any other questions you might have as you continue developing Botifymograph.

Me: Okay, this is the old codebase. Don't get confused over specific implementation. We're totally ignoring the streaming message stuff for now. We'll fill that back in later with chatq calls.

Also not we're not using the persistent data object nor zeromq anymore. But do take note of the next steps coming with getting Botify project URLs and looking up their Analysis slugs and what we're going to do once we get all the parameter data. Just take it all in and give yourself an outline I can feed back to you to prepare for next steps and to have a clearer idea of what to keep and what to throw away from this implementation as we proceed...

Thank you for providing the old codebase. I've analyzed it and prepared an outline of the key components and next steps. Here's a summary to help guide our next steps:

1. Key Components to Keep: - Main Interface (SPA template) - Authentication and User Management (Botify API token handling) - Project and Analysis Management - Data Pipeline Processes (Link graph generation) - File Management (CSV generation and handling) - Ollama AI Integration - Error handling and logging - Real-time status updates using Server-Sent Events (SSE)
2. Components to Replace or Modify: - ZeroMQ (ZMQ) for Inter-Process Communication: This can be replaced with WebSockets or a simpler messaging system if needed. - Persistent Data Management: Consider using FastHTML's MiniDataAPI instead.
3. Next Steps: a. Implement Botify API token acquisition and management:
 - o Create a form for users to input their Botify API token
 - o Implement token validation and storage
 - o Add functionality to clear/remove the tokenb. Develop Project URL and Analysis Slug selection: - Create an interface for users to input or select their Botify project URL - Implement fetching and displaying available analysis slugs for the selected project - Allow users to select an analysis slug for link graph generationc. Implement Link Graph Generation: - Develop the process to generate link graphs based on the selected project and analysis slug - Implement depth selection or automatic depth determination - Create a progress indicator for the link graph generation processd. Implement CSV File Management: - Develop functionality to list generated CSV files - Implement download functionality for generated files - Add option to delete generated CSV filese. Integrate Ollama AI (optional): - Implement Ollama AI integration for enhanced user interaction or data analysisf. Develop Real-time Status Updates: - Implement Server-Sent Events (SSE) for real-time status updates during long-running processesg. Error Handling and Logging: - Implement comprehensive error handling throughout the application - Set up logging for important events and errors h. UI/UX Improvements: - Enhance the user interface for a better user experience - Implement responsive design for various device sizes i. Testing and Optimization: - Develop unit tests for critical components - Perform performance optimization, especially for large datasets j. Documentation: - Create user documentation for the application - Develop technical documentation for future maintenance and updates

This outline should help guide the development process as we port the application to the new structure. We'll focus on implementing these components one by one, starting with the core functionality of Botify API integration and link graph generation.

Me: Now look at the existing codebase and take note of how we do things now and what will be replacing the old ways. Take particular note of chatq and the ability to stream data on long running jobs with websockets. Also get a sense of priorities, such as compliance with FastHTML conventions. But also take note of what from the old implementation was really strong, such as the analysis slug menu and the way user interface elements are locked and released and alerts are sent in the course of long-running events.

Cursor: After analyzing both the existing codebase and the old implementation, here's a summary of key points and priorities for the new implementation:

1. FastHTML Compliance: - The new codebase uses FastHTML conventions, which should be maintained and expanded upon. - The `fast_app()` function is used to set up the application, including database and WebSocket support.
2. WebSockets for Real-time Communication: - WebSockets have replaced ZeroMQ for real-time updates. - The `@app.ws('/ws', conn=on_conn, disconn=on_disconn)` decorator is used for WebSocket handling. - Functions like `on_conn`, `on_disconn`, and `insert_item_to_list` manage WebSocket connections and updates.
3. Streaming Long-running Jobs: - The `stream_chat` function demonstrates how to stream responses, which can be adapted for long-running jobs. - Asynchronous tasks are created using `asyncio.create_task()` for non-blocking operations.
4. User Interface Elements: - The old implementation had a strong approach to locking and releasing UI elements during long-running events. This should be replicated in the new system. - The analysis slug menu from the old system was effective and should be reimplemented in a similar fashion.
5. Alert System: - The old system's method of sending alerts during long-running events should be incorporated into the new WebSocket-based system.
6. Error Handling and Logging: - Comprehensive error handling and logging are present in both old and new implementations and should be maintained.
7. AI Integration: - The new system uses `chatq` for AI interactions, which should be leveraged for enhanced functionality.
8. Data Management: - The new system uses SQLite through FastHTML's data API, replacing the old JSON-based persistent data storage.
9. CRUD Operations: - The new system includes CRUD operations that are integrated with the WebSocket system for real-time updates.
10. Asynchronous Programming:
 - o The extensive use of `async/await` in the new system should be maintained and expanded for all I/O-bound operations.

Priorities for Implementation:

1. Maintain FastHTML compliance throughout the new implementation.
2. Implement the Botify API token acquisition and management using the new WebSocket system.
3. Recreate the project URL and analysis slug selection interface, incorporating the strengths of the old system.
4. Implement link graph generation with real-time progress updates using WebSockets.
5. Develop a robust system for locking and unlocking UI elements during long-running processes.
6. Implement an alert system for user notifications using WebSockets.
7. Integrate `chatq` for AI-assisted interactions and data analysis.
8. Ensure comprehensive error handling and logging throughout the application.
9. Optimize the UI/UX, taking cues from the effective elements of the old system.
10. Implement efficient CRUD operations for managing tasks and client data.

By focusing on these priorities and incorporating the strengths of both the old and new systems, we can create a robust, efficient, and user-friendly application that leverages the power of FastHTML and modern web technologies. Okay, you nailed it. So, give yourself precise instructions for the next step. Feel free to ask me to feed you back code examples from the old codebase which I have ready to copy/paste to you. Asking for a Botify Project URL shouldn't

require any reminders, but as you want to do the Analysis Slug lookup, I know there's some good stuff to show you again.

Me: Okay, you nailed it. So, give yourself precise instructions for the next step. Feel free to ask me to feed you back code examples from the old codebase which I have ready to copy/paste to you. Asking for a Botify Project URL shouldn't require any reminders, but as you want to do the Analysis Slug lookup, I know there's some good stuff to show you again.

Note: And this is where it went terribly wrong again. Like humans, AI's are remarkably hit-or-miss. Sometimes they're really good at figuring out precisely what you want, and other times it's... just too difficult, maybe?

Refactoring and Porting App #1 Goes Into Day 2

I didn't want to, but I had to pull the plug around 3:00 AM and get some sleep. The good news is that most of the hard bits are done.

- I've "wedged in" the plugin to the page layout
- I have a progressive reveal of PicoCSS-style "stacked cards" for a Jupyter Notebook-like progressive reveal
- I have connectivity with the Botify API, both for login and one of the harder menu tricks
- I got the old app working again for side-by-side comparison

Tech-Craft & Not Fight Diminishing Returns of Exhaustion

But that's where I ran out of steam. Diminishing returns on productivity from exhaustion, and I really have to dive into preparing for the client meeting today, at around 2:00 PM. It's only 9:00 AM, so if I want to do both things, I have to make some strategic moves forward and have it all wrapped up by no later than 11:00 AM so I can shift gears and make a good showing on my first management of a client meeting in some time. But it feels really good to be this engaged in technical work again. That old tech-craft feeling is back.