!!!! NOTE !!!!!! This is the old Version! For an updated version, visit Version 2

NYC Planning Labs Guides

Best practices, how-tos, and methods by which Labs encourages and achieves an awesome, successful working environment

---

Methodologies

Core Values

We strive to work in a modern way and use best practices on all our projects. Here’s what that means to us:

- **Open by Default**
  Planning Labs is committed to openness and transparency, and all activities will be open by default.

- **Build With, Not For**
  Planning Labs is a design partner, working closely with its customers in visioning and agile development. Customers are expected to provide resources for the duration of a product build, and will be heavily involved in sprint planning, testing and acceptance.

- **Ship Early, Ship Often**
  Planning Labs' preferred projects will yield a minimum viable product in 4-6 weeks, providing a rapid cadence and the opportunity to work on diverse projects and problems across the agency.

- **Document and Disseminate**
  Planning Labs will place a high priority on documentation and outward messaging in support of its mission to promote modern technology best practices, and will manage its own content to include general information about the team's mission and work program, blogging, docs, and social media engagement.

Read the full text of our charter on GitHub: https://github.com/NYCPlanning/labs-charter

Scrum

Our agile software development process is based on the Scrum framework. It's an iterative, incremental approach which allows us to optimize predictability and control risk through transparency, inspection, and adaptation.

All Planning Labs team members and customers should be familiar with Scrum: http://www.scrumguides.org/scrum-guide.html

Product Owner: All Labs projects have customers—planners at DCP who are real users of the tools we build, and who are responsible for maximizing the value of Labs' work. It's important that the customer knows they're the Product Owner in the Scrum framework, and that Labs "builds with, not for." (Some projects have a working group of customers. In this case, one person leads that group as the Product Owner.)

Sprint Cycle: We usually use a one-week sprint cycle, with sprint planning on Monday mornings, and live demo on Friday afternoons.

Sprint Planning: At the beginning of every Sprint, we meet with our customer to define a goal for the Sprint and determine what work will be done.

Daily Scrum: Once a day, the Labs team has a 15-minute standup meeting. We keep it short. Any detailed conversations happens afterwards. During this quick meeting, we each answer 3 questions:

1. What did I do yesterday?
2. What am I going to do today?
3. Is there anything blocking me from being productive?

Sprint Review / Demo: At the end of every Sprint, we invite the whole agency to review the week's work at our demo. Keeping this is open to all is an effort to increase transparency and evangelize our way of working. We like to say, "Demos, not memos."

Retrospective: After each demo, the Labs team retires to a quiet space where we discuss how the Sprint went. What did we achieve? How effective were we? What went well? What could we do better? We keep a running document of notes from these retrospectives so that we can look back at our progress as a team. Being such an extremely agile team, there's an implied task when using these guides:

1. Reevaluate and update the guides with any discovered improvements

---

Development environment

Technologies

- **Node.js** — https://nodejs.org/en/ (current version 8.11.3)
  We strive to use the latest LTS version of Node.js as the standard version for local development. All projects using Node should be compatible with this version. When a new LTS version is released, the team should upgrade their machines together, and projects should be updated to work with the new version.

- **Yarn** — https://yarnpkg.com/lang/en/
  We prefer `yarn` for installing JavaScript dependencies. It builds dependency graphs quickly. Although npm increased its speed in later releases, many of our projects use of `yarn.lock`, so we continue to use it until there's reason to switch to npm.

- **Ember.js** — https://www.emberjs.com/
  Ember is a javascript framework that handles routing, components, dependencies, data modeling, etc for single-page applications.

- **Mapbox GL** — https://www.mapbox.com/mapbox-gl-js/
  A next-generation web mapping library, which allows for the use of vector tiles, complex symbology, 3D, advanced viewport control, etc.

- **Foundation for Sites** — https://foundation.zurb.com/sites/docs/
  A styling and layout framework that we use for our frontend applications. Although Foundation includes JavaScript components, we use it mostly for Sass in Ember apps.

- **Express** — https://expressjs.com/
  A Node HTTP server, we use express heavily for building JSON APIs to support our single-page applications.

- **Docker + Dokku** — http://dokku.viewdocs.io/dokku/
  Dokku is an abstraction of docker that allows for easily deploying containerized applications and services by pushing a git command. It's open source, and very similar to Heroku. For more, read our blog post on deploying with dokku.

- **Netlify -** https://netlify.com
  Netlify is a build and hosting site for static sites and single-page applications. We use it to build and host our ember, react, and gatsby applications

- **PostGIS** — https://postgis.net/
  The spatial extension for PostgreSQL databases. This technology is under the hood in Carto, but we also use it directly. It allows us to store geometries in the database and use spatial queries (like `ST_Intersects()`, `ST_Buffer()`, etc) to do spatial processing on the fly.

- **OpenMapTiles Server** — https://openmaptiles.com/server/
  An open source tile server that provides OpenStreetMap-based vector tiles. These tiles form the "basemap" of all of our maps.

- **Maputnik** — https://maputnik.github.io/
  Open Source GUI for MapboxGL Style Editing. We built labs-maputnik-dev-server so that we can open any map in our dev environment in GUI to try out style rules. This blog post has more about how we use Maputnik

- **Airbnb JavaScript Style Guide** — https://github.com/airbnb/javascript
  Our team uses the popular airbnb eslint rules for all javascript code. Each project may have unique exceptions, but the airbnb rules give us a good baseline for code styling.

- **Git Tower** — https://www.git-tower.com/
  A GUI for local git commands which helps with commits, branching, merging, pushing, pulling, rebasing, etc

- **Ansible** — https://www.ansible.com/
  Ansible playbooks are "infrastructure as code"—configuration rules that can be

programmatically applied to virtual servers. These rules will govern ssh access to machines, firewalls, security updates, what happens on restart, etc. Storing them as code means we can apply new rules across all of our infrastructure without a lot of manual configuration.

Third-party services

- **GitHub** — https://github.com
  Cloud git-based code repository, with identity, issue tracking, etc. All labs projects are maintained in the nycplanning github organization.
  - Offboarding: Ensure user is removed from the organization, as well as from the Outside Collaborators tab.
- **Slack** — https://nycplanning.slack.com/
  Beyond day-to-day team communication, we use slack for automated alerts on code commits, Pull Requests, and Deployment. See the #labs-bots channel for github, updown.io, and other alerts.
- **DigitalOcean** — https://www.digitalocean.com/
  Cloud hosting. We spin up virtual servers here, and use it for floating IPs, firewalls, and document storage (similar to Amazon S3). Tag any new Droplets (Servers) with `labs`, and our Ansible playbooks will find them.
  - Offboarding:
    - Remove from the team
    - Remove SSH key(s)
- **Carto** — https://carto.com/
  Cloud-based spatial database, JSON api, and tile service. Carto is our “instant spatial backend”, and frontend applications often communicate directly with the carto APIs.
- **Waffle.io** — https://waffle.io/NYCPlanning/
  Kanban-style wrapper for github issues. We set up a waffle board for each project, and use it to track sprints. Each issue moves across the board from *To Do*, to *In Progress,* to *Done*.
- **CircleCI** — https://circleci.com/
  Continuous integration/deployment of `develop` and `master` branches to staging and production environments respectively. Setup instructions.
  - Onboarding: Invite user to circleci team.

    - Offboarding: Inherited from GitHub. ←- need to confirm

- **Airtable** — https://airtable.com/
  An intuitive cloud database, airtable allows you to quickly set up tables with controlled inputs, add public form input, etc. We use it to store project content for our website, allowing for quick updates without modifying the codebase.

- **mLab** — https://mlab.com/
  Cloud-hosted mongodb database.

- **GoDaddy** — https://www.godaddy.com/
  The domain registrar and DNS host for planninglabs.nyc

- **Google Analytics** — https://analytics.google.com/
  Tracks site usage, acquisition, behavior, bounce rates, etc. We set up analytics on all Labs sites.

- **Google Drive** — https://drive.google.com/
  Used to share documents, presentations, and other files

- **TweetDeck** — https://help.twitter.com/en/using-twitter/tweetdeck-teams
  Twitter aggregator, allows for anyone on the team to make use of the @nycplanninglabs twitter account.

- **Updown.io** — https://updown.io/
  A simple and inexpensive uptime monitor. It checks our services every few minutes, and fires off slack messages if things are not returning the expected responses. We use the updown.io API to power a public status page at https://planninglabs.nyc/status/

- **Medium** — https://medium.com/nycplanninglabs
  We maintain a Medium publication for team blogs. All Labs team members are expected to contribute one article per month.

- **Dependabot** — https://dependabot.com/
  Automated dependency updates for things like NPM packages. (Note: This can become noisy and sometimes lead to unwanted upgrades. It's best used for security-only updates or turned on during specific maintenance periods.)

    - Offboarding: Inherited from GitHub.

New service checklist

- Ensure there is one user per person

- Ensure every user has two-factor/multi-factor (2FA/MFA) enabled; enforce this where possible

Repository best practices

This starter app provides templates for readmes, PRs, and issues:
https://github.com/NYCPlanning/labs-starter-app
The PR template encourages documentation & tests as part of a proper PR

…
be aware of lingering PRs. Don't open a PR w/o checking existing ones… help w/ conflicts
delete branch after merging…

…issue template…
…README…

…protected branches…
…it should be required that PRs have `develop` merged in…

…we should add a "topic" to the repository called `labs`...

Linting with eslint

As a general rule, we use airbnb eslint rules for all javascript code.
https://www.npmjs.com/package/eslint-config-airbnb

To add eslint to a project, create a file called .eslintrc in the root of your project. This json file tells the linter which rules to apply, exceptions, etc. (for ember apps, this file formatted a bit differently and is called ".eslintrc.js"

ZoLa's .eslintrc.js file

You will also need to add a linting extension to your text editor, which will markup your text and show lint errors that need to be addressed.

Ignore rules - Use sparingly! The rules are there for a reason, we should have a good reason to ignore them. You can ignore eslint rules by adding // eslint-disable-line to a line, or adding a global ignore to .eslintrc.

Tests should run the linter - ember's built-in tests will automatically run eslint rules. For express/koa or other projects, you will need to add a test that runs the linter. See labs-layers-api's eslint tests for an example of running eslint as a mocha test.

How we use branches

- `master` —
- `develop` —
- `[123]-[issue-name]` —

Naming conventions

Application name is important, it will be reused in many places, and may be prefixed, extended, etc.

We should strive to keep repo names and dokku application names similar. Prefix all repository names with `labs-`. No suffix implies the repo for the frontend app, all other repos should be suffixed

Repo:

- `labs-{appname}`
- `labs-{appname}-api`
- `labs-{appname}-somestaticdata`
- `labs-{appname}-somescripts`

Dokku `appname` should match repository name, minus `labs-`

Maintenance

We must be proactive about repo maintenance… quarterly/etc

- Check Repo Permissions - Are there external contributors, teams, or individuals with access that do not need it anymore?

- Dependency updates - Run `yarn outdated`. Patch versions can usually be upgraded automatically using `yarn upgrade-interactive`. Test minor and major version changes, leave a note for things that can't be upgraded at the moment.

- Delete stale branches. All repos should have master and develop, and any branches that are related to open PRs.

- Scrub issues, all issues should have at least one label (bug, enhancement, etc). Close duplicate, irrelevant or "wontfix" issues.

- PR Cleanup - Get to “inbox zero” on pull requests. Determine whether PRs should be merged or closed, and always leave a note.
- README … all repos should have at least the following sections in their readme.md… …
- LICENSE, CONTRIBUTING ...

SSL/HTTPS

All NYC Planning Labs apps and services must be made available via HTTPS, and should redirect HTTP traffic to HTTPS

- To accomplish this for our dokku environment, we use the dokku letsencrypt plugin. SSH to the dokku server, and enter ‘dokku letsencrypt {appname}’, and follow the prompts
- You can quickly auto-renew SSL certs for all dokku apps using `dokku letsencrypt:auto-renew`
    - For services running on other servers behind nginx, we use letsencrypt’s certbot to create SSL certificates. There is also an nginx plugin for certbot that will automatically add the certs to the nginx config files (and provide a redirect for HTTP traffic)
- Updown.io includes a check for SSL expiration, and will issue a warning via email and slack when a cert is 7 days from expiration. ← This is a good time to run autorenewals

---

Deployment

Frontend Apps

Frontend apps are deployed via Netlify. To deploy an app, follow these steps:

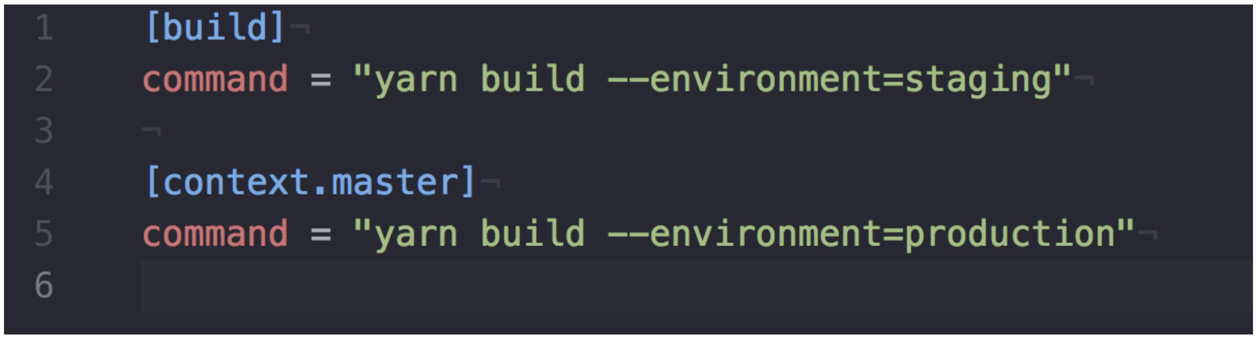
Create Netlify App via the UI

1) Login to netlify.com with labs_dl
2) Add the repo via the netlify UI
    a) click "New site from Git"
    b) choose "GitHub

    c) auth if prompted
    d) select the NYCPlanning GitHiub org
    e) choose the project
    f) Set build command to “npm run build”
3) Change site name to match the repository name (labs-zola, labs-factfinder, etc)
4) Master should be the production branch by default. Confirm this in Settings > Build & deploy > Deploy contexts
5) Set deploy previews to “Automatically build deploy previews for all pull requests” (it should be set to this by default)
6) Enable branch deployment for develop (under deploy contexts section)
7) Enable slack notification into #labs-bots on “Deploy Succeeded” (you will need the webhook url, option is under “deployment”)

Repo Changes Necessary for Netlify

1) If production and staging require different build commands, add netlify.toml to the root of the repo with contextual commands. These settings will override those set in the netlify UI. (This is useful if production and staging should talk to different APIs which we often do in our ember apps)



```
[build]
command = "yarn build --environment=staging"

[context.master]
command = "yarn build --environment=production"

```

2) Single page apps need to redirect all non-index routes to the index route. This is accomplished by adding a _redirects file in the root of the built site directory. We use an ember addon to help with this.

Run ‘ember install ember-cli-netlify’ create .netlifyredirects file with the following rule:
“/*   /index.html   200”

Commit and push all of the above changes to develop to github, which will trigger a netlify deployment

Custom Domains

Netlify sites are all hosted at {appname}.netlify.com. Netlify allows for custom domain names, but we want to keep all of our DNS config separate (because we don’t have control over the .gov domains, which are all currently pointing to a VPS).

Custom domains for netlify apps should be configured by creating an A record for the domain that points to labs-01.planninglabs.nyc.

Once the DNS is pointing to labs-01, add an nginx config with proxy-pass rules that will proxy the netlify site for production and staging (see google doc below)

Use certbot to configure SSL on the production and staging domain names. (see google doc below)

Google Doc with steps for moving dokku-hosted frontend apps to netlify (January 2019)

APIs

Express APIs are deployed using dokku.

Onboarding & offboarding checklists

New team members

When new people join the team, we're responsible for making sure they're set up properly. *(Not all items are applicable to every new team member.)*

An existing team member should:

- Send a Slack invitation to the new team member's email address, and add them to appropriate channels
- Make sure the new team member's email address is added to the Labs_DL@planning.nyc.gov distribution list
- Make sure a MacBook is issued to the new team member
- Give the new team member access to the DCPguest network
- Invite the new team member to the NYCPlanning GitHub organization and grant them access to applicable repositories

- Make sure the new team member has generated and linked their SSH key for their GitHub account
- Give SSH access to the new team member
- Grant the new team member access to applicable third-party services
- The new team member should set up an account with medium.com, and an existing team member should add them to the nycplanninglabs medium publication as an **editor.**

The new team member should:

- Join our Slack workspace, join the appropriate channels, and choose a good avatar
- Set up their dev environment on their MacBook
- Accept the invitation to join the NYCPlanning GitHub organization
- Make sure applicable third-party services can be accessed
- Create a pull request on our website's repository, adding themself to the "Our Team" section of the About page
- Schedule one-on-ones with all team members to meet, learn, and ask questions
- Review and improve these guidelines

New clients

When new clients work with us for the first time, they should understand what our process requires of them as a Product Owner.

- Read over our methodologies guides, familiarizing themselves with Scrum and the responsibilities of the Product Owner role
- A Slack invitation should be sent to the client's email address, they should be added to appropriate channels, and choose a good avatar
- The client should create a GitHub account if they do not have one
- The client should be granted access to their product's GitHub repository so they can maintain the Product Backlog (they should also be sent a Waffle.io link)

Offboarding

When team members leave, in addition to standard HR offboarding procedures, we are responsible for updating our team's internal workflows, processes, and security.

- See service-specific offboarding tasks in "Third Party Services" above

---

Product lifecycle checklist

The following describes the ideal lifecycle of a typical Labs project. It outlines what *should* be done from the start of a project, and the state that *must* be attained by production apps that have real users.

1. **Open from the start**
   "Open by Default" is a core value of Planning Labs. From a project's very start, a public repo should be created on GitHub (see Repository boilerplate)

2. Add the project to the "Labs Repo Tracking" Airtable

3. Use the skeleton map app to easily incorporate all the latest integrations and configurations (CircleCI deployment, favicons, labs-ui, labs-layers-api, labs-ember-mapbox-composer):
   https://github.com/NYCPlanning/labs-map-skeleton-app
   3.1. If you choose not to use this, you'll still want to copy over specific files, including favicons, circle-ci config, and config/environment.js
4. **Set up Dependendabot**
   At least security updates, for all dependency files in each repository.

5. **Set up continuous integration**
   As early as possible, a nascent version of the project should be deployed at {app}.planninglabs.nyc so that people can start using it.
   …During sprints, the \`develop\` branch should regularly/iteratively be deployed to the staging environment. At a minimum, \`develop\` should be deployed before every sprint review session (demo). A good practice is to use continuous integration to deploy to staging when \`develop\` changes.
   …{app}.planninglabs.nyc

6. **Set up eslint**
   We use Airbnb's eslint rules for all javascript projects. This usually involves installing eslint and the eslint-config-airbnb-base package, and configuring the app to use it.

(Add a *yarn test* command to express projects, for ember apps, ensure *ember test* is also applying eslint compliance) Developers may also need plugins for their text editors to apply eslint errors and warnings in the code.

7. **Get a domain**
   Once a product name is determined, a planning.nyc.gov subdomain is obtained (see Request subdomain DNS updates from DoITT).

8. **Uptime monitoring**
   Add monitoring for the new app and any associated services to updown.io.

9. **Google Analytics**
   Before a production environment is live, Google Analytics should be added.

10. **Share the production URL**
    Since Labs projects are open from the start, the .gov domain is publicly accessible but may or may not be publicized. At this point, the Product Owner might choose to share the project with beta users.

11. **Iterate on staging**
    Sprint work should continue to happen on the `develop` branch which is deployed to the staging environment.

12. **Production release**
    Subsequent sprints should be accepted or declined by the Product Owner. If accepted, the `develop` branch should be promoted to `master` which is deployed to the production environment.

13. **Social media sharing images**
    Before the product is publicized Twitter card and Facebook Open Graph meta info and images should be added.

14. **Publication**
    When the Product Owner declares public readiness, the app is publicized by the agency (links on DCP website, press release, Tweets, etc).

See also: DoITT's Technical ~~Vendor~~ Resources page.

Infrastructure

Offboarding: Remove user's ssh access to servers using labs-infrastructure ansible scripts

See here.

Incidents

Things happen. After any incident (security, downtime, etc.), hold a blameless post-mortem.

Operations how-tos

Add an external visitor to the building visitor/guest list

- Email Receptionist_DL@planning.nyc.gov and include the visitor's full name and approximate arrival time (e.g. *"Jane Doe will be visiting us at 1pm. Can you please add them to the Building Visitor's list? Thanks!"*)

Access email on an off-network machine

1. Go to Outlook Web App at https://csmail.nyc.gov/
2. Login with your LAN ID (e.g. "DCP\S_Hudson")

Access DCP Commons on an off-network machine

1. Go to https://planning.ra.nyc.gov/
2. Login with your LAN ID
3. Select "Planning - Windows Users Role" at the prompt to choose a role
4. If you see the message "Your OS/platform is not supported for this component" select "Home" in the header
5. DCP Commons (and other internal sites) is selectable in the list of Web Bookmarks

Access CityTime on an off-network machine

1. Follow the steps to access DCP Commons from an off-network machine
2. On the front page of DCP Commons, there's a link to CityTime

Request subdomain DNS updates from DoITT

1. Go to CityShare (available as a link on DCP Commons)

2. Go to “Citywide Service Desk - My Desk” (Link in the right sidebar)
3. Click “Submit a Request”
4. If you have never used DoITT’s CityWide Service Desk before, you’ll need to call them to create an account for you. Their number is: 212-692-4357.
5. The login is… weird! The instructions are wrong/unclear. It says use your LAN ID, but it actually requires the local-part of your email (e.g. cwhong instead of c_whong). It also says to use your agency acronym, but it requires the nyc.gov subdomain:
    a. Login: {email-local-part}@planning (e.g. cwhong@planning)
    b. Password: {lanpassword}
6. Choose “Application” from the “What does this request closely relate to?” dropdown
7. In the “Please provide additional details…” textarea, enter DNS record instructions for pointing to Labs’ floating IP on DigitalOcean (e.g. *“Please create an A record: {mynewsubdomain}.planning.nyc.gov pointing to XXX.XXX.XXX.XXX.”*)
8. Before submitting the form, test and double check the IP address you are providing
9. Submit the form (you should get an email verification)
10. Now play the waiting game

Managing projects on planninglabs.nyc

We use airtable as an “instant backend UI”, it stores the data and gives us a highly-accessible way to modify text, add images, etc. Airtable has a JSON API, which we route through an express API to serve projects data to planninglabs.nyc.

1. Go to https://airtable.com/ and create an account
2. Have a team member send your account an invite to the Airtable shared workspace.
3. On the “Bases” page, choose “Labs Live Projects” (if you do not have access, have someone share the base with you)
4. Update the projects accordingly:
    - All changes are instantly visible on planninglabs.nyc
    - The order of projects in Airtable effects their order on the website (the newest project should be at the top)

- The `type` field determines if/where projects appear on the projects page. It's good practice to add the `type` field last (or keep it blank while editing other fields). This avoids the project being visible on the front end while changes are being made on the back end.
  - `feature` — A featured card that links to a project specific page, which requires `thumbnail`, `url`, and `description`
  - `resource` — "Developer Resources" section
  - `current` — "In the Works" section

Set up CircleCI

1. Go to https://circleci.com/ and sign in with your GitHub account
2. Switch to NYCPlanning in the organization drop-down (top left) if it's not selected
3. Click "Add Projects" button on the left to see the list of available repositories
4. Find the repository and click "Set Up Project." Note: the user must be an "owner" of the GitHub organization in order to do this.
5. Add the "deployer" SSH private key so CircleCI can deploy to Labs server infrastructure. This can be found in the shared 1Password account.
   a. Click the gear icon (upper right corner)
   b. Click "SSH Permissions" (left sidebar)
   c. Add the key
   d. Leave the hostname blank
6. Add any environment variables that may be necessary to run the tests
   a. Click the gear icon (upper right corner)
   b. Click "Environment Variables" (left sidebar)
7. In the project repository, create a folder called `.circleci` and add a file called `config.yml` (a starting template can be found here)
   a. Be sure that the node version is correct
   b. Install any environment-specific dependencies by adding a `run` step:

```
- run: sudo apt-get install libgif-dev
```

8. Commit changes, and push the appropriate branch to the GitHub repository (this should trigger a build on CircleCI)

Add HTTPS/SSL to a dokku app

1. SSH to the dokku server
2. Run `dokku letsencrypt {appname}`

Note: We have set up an autorenew cron job using `dokku letsencrypt:cron-job --add`, so it should not be necessary to manually renew expired certs. See the docs for the letsencrypt dokku extension.

Delete old attachments from Slack

When there are too many attachments, users will start to see warnings when attaching images & files. Run the script below to clear old ones.

1. Globally install slack-delete-files
2. Go to https://api.slack.com/custom-integrations/legacy-tokens to get an API token. (Maybe only admins for the slack workspace can do this delete, we should confirm this)
3. Run `slack-delete-files --age=60 --pinned --token={your token}`

Update Version Number for npm addons

- How to check the latest published version number:
    1. Create an npm username by navigating to https://www.npmjs.com/
    2. Click on your avatar on the top right of the screen
    3. Click Profile Settings
    4. Click on nyc-planning-labs under Organization (all team members need to be added to the organization)
    5. The latest published version number is underneath the package titles

- Versioning (currently working on this)

-npm version (major, minor, patch)

    -path upgrade
    -semantic versioning
    -patch version for minor change
    -npm version patch

- How to publish an updated version in your terminal
    1. In your terminal navigate to the correct repository
    2. Type: npm login
    3. Enter your username, password, and email
    4. Type: npm publish (this should publish the latest version of the package that is in your local repository)
    5. To check the version, type: npm version

Add Google Analytics to a New Project

How

Book a Room

How

Yarn Link

How

Set Up Browserstack

How

Dokku

How

Netlify

How

Digital Ocean

How

Navicat

How

Ember Testing

How to test one component at a time: `ember test --server -m "Integration | Component | map-form" `