Tab 1

1. GTM Process
    a. DEVDOCS Durable Team SME List
    b. Attend GTM meetings
    c. Implement changes/updates (usually from the PM or CSS tickets)
2. Beta access
    a. Adding someone to a beta project
    b. Some PMs require approval before we can add them.
3. Feedback slack channel
    a. Dev-center-feedback
    b. Also add the following channels:
        i. dev-documentation slack channel
        ii. Dev-docs-team
        iii. Join the BigCommerceDevs slack workspace
4. GitHub Issues
    a. Respond to developer or merchant issues
5. GitHub Repos
    a. Bigcommerce /docs
    b. Bigcommerce /api-specs-ssot (private, beta repo)
    c. Bigcommerce /developer-center (for creating menus)
6. JIRA tickets
    a. The current backlog is 107 tickets. This is from 170 tickets about 4 months ago.
    b. All new tickets have Tina's name. I have been working on my tickets and Tina's tickets.
7. Style Guide
    a. Devdocs Style Guide Checklist
    b. Devdocs Style Guide
8. How to contribute to devdocs
    a. Contributing to BigCommerce Developer Documentation
    b. Diataxis templates
    c. Updating DevDocs Content*
        i. The information is marked as "outdated", but the practices are still applicable. The main departure is that the *dev-docs* repo is no longer used. Use the *docs* repo (linked above) instead.
9. https://www.bctoolbelt.com/

Testing/Tools

Testing/Tools

BigCommerce Spec Editor

Purpose: To verify YML/YAML docs content against BigCommerce build tools.
URL: developer.bigcommerce.com/spec-editor

1. Copy doc into editor to check for errors
2. Verify behaviour of changed section(s)
    a. If the body is missing (from request or response) but an example is present, there is a schema error, usually an unresolvable data override.
    b. Some errors require a page reload to resolve. Use the Swagger editor to resolve the editor, then reload the page and re-paste the code
3. Check for unexpected section impacts
4. **NOTE:** This editor is the most accurate for verifying behaviour on BigCommerce DevDocs, regardless of how it displays on the Swagger Editor.
5. **NOTE:** This editor doesn't rebuild on fatal errors. If a block of red text shows an error, you will need to reload the page.
6. **NOTE:** This page doesn't save sessions. Do not navigate away from this page without having your work saved elsewhere.

BigCommerce MDX Editor

Purpose: To verify MD/MDX docs content against BigCommerce build tools.
URL:

1. Copy doc into editor to check for errors.
2. Verify behaviour of changed section(s).
3. Check for unexpected section impacts.
4. **NOTE:** This editor is not currently public.
5. **NOTE:** This page doesn't save sessions. Do not navigate away from this page without having your work saved elsewhere.

Swagger Editor

Purpose: To verify YML/YAML docs content against OpenAPI/Swagger standards.
URL: editor.swagger.io

1. Copy doc into editor to check for errors
2. Verify behaviour of changed section(s)
3. Check for unexpected section impacts

CircleCI

Purpose: To monitor builds and check previews.

Login: Login with your BigCommerce Github account. This may require secondary verification.
URL: https://app.circleci.com/pipelines/github/bigcommerce/developer-center/
Other Pipelines: `storefront` and `api-proxy-java` are used for GraphQL updates.

1. Open CircleCI developer-center pipeline.
    a. Updates from developer-center repo will come under the github username of the person who made the pull request.
    b. Updates from docs repo will come under Mark's github username
2. Identify the build related to your update
    a. Updates from developer-center will have branch and commit/pr info
    b. Updates from docs will have to be opened to confirm commit info
3. For navigation updates, net new docs, and some other items, the progress can be viewed in the `preview_deployment` section.
    a. Once the `Deploy Project Artifacts to Vercel` test runs, the build is ready for preview.
    b. Preview is found at `https://developer-center-[HASH]-bigcommerce-dx.vercel.app/`
    c. The `[HASH]` is available as part of the build process. If you can't find it, at minimum it's available in the `Deploy Project Artifacts to Vercel` test on GH
    d. **NOTE:** If you have access to the Vercel project, you can monitor builds and their progress without relying on the CircleCI progress.
4. For doc updates, the build progress is mostly an indicator of when the page will be live.
5. If an update fails, it may be re-run from beginning or from last error. A subsequent failure should not be re-run, but the error report should be checked.

Release Notes

Release Notes

1. Release notes are published every Thursday around noon.
2. How to submit a release note
    a. Locate the relevant CSS Epic ticket.
        i. In a comment, tag the Dev Docs lead as well as the PM or durable team lead that should provide the final review and approval.
        ii. Share the details that you want the release notes to include.
        iii. All approved updates should be in by Wednesday afternoon so they appear in Thursday's publication.
3. Improve the Developer Release Notes
4. Improving the Changelog Process
5. Publishing Release Notes
    a. Gather report of merged pull requests from relevant repositories
        i. `/docs` for regular content updates
        ii. `/api-specs-ssot` for beta progress
        iii. `/catalyst.dev` for Catalyst updates
        iv. `/developer-center` for navigation and core updates
    b. Create a new draft in the Dev Center Noticeable project. This requires an admin to add your account to the project.
    c. Get reviews from relevant stakeholders
    d. Publish draft
    e. **NOTE:** This has been known to lag behind publication. If release notes do not publish within an hour, a pull request may need to be merged to force a rebuild of the release notes page.

Github Note:

The default template for Pull Requests in `/docs` contains fields for

- What Changed?
    - Explanatory content in present tense to describe what changed and why
    - Short, to the point.
- Release Notes Draft
    - Content communicating the update for release notes
    - Ideally structured like current release notes
    - Intended as a draft, to be revised as necessary

Github

Github

Repositories

- References and guides, generally - /docs
    - All yaml/yml files are api specifications.
        - Structure follows Swagger format (see swagger.io for details)
        - Introductory content is in primary `description` field
        - Sub-navigation is configured through `tags`
            - Each tag is its own sub-navigation item
            - API endpoints/methods may be configured to display on multiple tags
        - If root navigation title matches a tag, its endpoints display above all sub-navigation items.
        - Any yml/yaml files in a `models` folder are intended as reusable components, not as standalone docs
    - All md/mdx files are guides and explainers - some of which are non-public facing.
        - Format is technically markdown + XML
        - Our system includes several default and custom REACT components
            - <Callout type="[severity]"> for callout bubbles about important info.
            - <Tabs items={['title 1', 'title 2', …]}> for tab groups
                - <Tab> for tab content. Only use within <Tabs>
            - <Steps> for outlining procedural content
            - See Nextra Built-Ins for more information
    - Most json files are structural
    - Almost all other filetypes are structural
- Basic navigation and structure, some guides - /developer-center
    - Each yaml/yml file that displays publicly should have an entry in a .json file somewhere here.
    - Each md/mdx file that displays publicly should have an entry in a .json file somewhere here.
    - Navigation structure in this repo almost mirrors that of the public site with some exceptions
- Beta docs - /api-specs-ssot [requires special permissions to access]
    - Structure is similar to /docs
    - Edits should be made following the same process as /docs
- GraphQL
    - Storefront - /storefront [requires at least `all-private-repos-pull-only` access. demands `dev-docs-team` access to merge PRs]

        - The only file we should be touching is main/resources/graphQL_descriptions.properties to update the descriptions that show up in the reference file.
        - Create a pull request following the same rules as other repos.
        - Approver can be a storefront team member or a `dev-docs-team` member.
        - CircleCI Pipeline must be watched to confirm successful build and testing or else the PR cannot be merged.
        - Once all PR checks complete, merging will cause a cascade of build notifications.
        - Once all `production` notifications are in, the next build of the dev centre will refresh the reference page.
    - Admin & Accounts - /api-proxy-java [requires at least `all-private-repos-pull-only` access. no further access granted]
        - The only files we should be touching are
            - modules/accounts/graphql/conf/accounts_graphql_descriptions.properties
            - modules/stores/graphql/conf/stores_graphql_descriptions.properties
        - Create a pull request following the same rules as other repos.
        - Approver **MUST** be from the PAPI team.
            - Reach out in `#api-platform-rfcr` for assistance if no action is taken
        - CircleCI Pipeline must be watched to confirm successful build and testing or else the PR cannot be merged.
        - Once all PR checks complete, merging will cause a cascade of build notifications.
        - Once all `production` notifications are in, the next build of the dev centre will refresh the reference page.
- ~~Catalyst docs [current] - /catalyst.dev~~
- Images [NOT GITHUB] Google Cloud Platform [requires permission to update, but not to access]
    - Add images here.
    - Pull URL using the overflow menu

Teams

BigCommerce makes heavy use of Github Teams, each of which has its own files, repos, and processes. Most content in the `/docs` repository is managed and maintained by devdocs, and in particular the following three teams:

- `dev-docs` (https://github.com/orgs/bigcommerce/teams/dev-docs)
    - Approver access to `/docs` and `/developer-center`

- `dev-docs-collaborators` (https://github.com/orgs/bigcommerce/teams/dev-docs-collaborators)
    - Write access to `/docs` and `/api-specs-ssot`
- `dev-docs-team` (https://github.com/orgs/bigcommerce/teams/dev-docs-team)
    - General read access to `/docs`, `/developer-center`, and `/api-specs-ssot`
    - Write access to most of the above

We also have the team `all-private-repos-pull-only` (https://github.com/orgs/bigcommerce/teams/all-private-repos-pull-only), which grants access to all BigCommerce repositories. We need this to be able to research relevant information in non-docs repositories.

Processes

Whenever possible, the following best practices should be followed for Jira tickets

1. One Pull Request per ticket per repository
2. Pull Requests should be marked with appropriate labels
    a. `sme-review-needed` for PRs that would be otherwise ready for publication
    b. `ready-for-review` for PRs whose content has been fully committed and require only revision
    c. `awaiting-css-cue` for PRs that have been reviewed and are otherwise ready to publish
    d. `do-not-merge` for PRs that are meant primarily as drafts or as “on-hold” updates
    e. Other labels as necessary
3. Pull Requests should be titled `[PROJECT]-[TICKET] - [Summary]` e.g. “DEVDOCS-777 Fix product name limit”
4. Pull Requests should be associated with a branch whose name matches the PR title, e.g. DEVDOCS-777

The following best practices should be followed with file creation/deletion/relocation

1. New files should be created within `/docs` roughly in the same file structure as their public counterparts e.g. `/docs/docs/start/guides` pages should correspond with `/docs/start/guides/*` files.
2. File movement should be paired with corresponding replacement actions within `/developer-center` in its `_meta*.json` files and file structure.
3. Updating URLs should be paired with building new redirects in `/developer-center` following the steps in the repo README.md
4. Deleting files should be paired with corresponding URL removals and redirects in `/developer-center` per above
5. Archiving files should be treated as file movement

Pull Requests

- When possible, work locally to create the full pull request as a single commit
- Provide a concise commit message connected with the changes made
- Once a pull request is configured, fill out the Pull Request Template before saving it
- Before submitting a Pull Request for review, verify that the content displays correctly in relevant preview tools (see Testing/Tools)
- When reviewing a Pull Request, use the “Suggestion” feature to provide corrections if you have them. Otherwise, provide clean comments on the lines affected by using the + button at the beginning of the line/lines affected.
    - Multi-line notes can be added by clicking and dragging to highlight the group of lines affected
    - **NOTE:** multi-line notes cannot include both removals and additions
- When handling revisions, the best practice is to commit or deny groups of suggestions first. When denying a suggestion, be sure to include clear reasoning why
- Once all reviews and revisions are managed, allow the Github Actions checks to run before merging.
- **NOTE:** All pull requests require review at minimum by one person from the team

Issues

We generally do not engage with issues unless the submitter includes clear guidelines about what is necessary to fix their issue.

Beta Management

Beta Management

Beta Tools

Quick and painless tool page for adding/deleting betas and assigning users to them.

- Beta Tools Page (requires admin access).
    - Add users to this list to give them access to betas
    - If a user is already present, you can add or remove betas for them as necessary
    - When adding a user to a beta, advise they may need to log out and back in for the beta to show up
    - **NOTE:** The email addresses provided MUST be the email associated with the user's github account
- Beta List Page (requires admin access).
    - Add betas to this page to make them show up in account pages
    - The name given here should be the root folder used in the repo
        - Display Name has caps and/or spaces (for professionalism)
        - Repo Name has lowercase and hyphens (for URL access)
    - A rebuild of Dev Center will sometimes be necessary to give access to docs
- /api-specs-ssot (requires Github access).
    - Follow guidance in the repo README for file structure
    - Add new beta folder following the convention outlined in existing betas
- Beta Access Request Doc.
    - Names are added by project managers and account managers
    - Current process is to search the tools page for users, then add them to the betas requested
    - Once a user has been added to a beta, update the status column of their entry

Creating a new Beta Docs

1. Merge the new docs into `bigcommerce/api-specs-ssot` following the correct process.
2. Create the beta at https://developer.bigcommerce.com/beta/tools
3. Create new beta entries in `bicommerce/developer-center`
    a. `_meta.tsx` at the path where the beta is delivered
    b. A new page in `content/beta/_meta.tsx`
    c. Links to any API references in `scripts/prebuilds/configs/_metaBetaApiContents.json`
    d. Links to any descriptive content in `scripts/prebuilds/routes/docs.json`
4. Add the Beta to the chip dropdown in Beta Project and Customer List

[Deprecated] Documentation Process

Documentation Process

General steps

1. Jira ticket comes in (feedback, task, or GTM) with a requested update.
    a. If the ticket contains linked doc(s), identify the files for the doc(s) in either `bigcommerce/developer-center` or `bigcommerce/docs` repository.
    b. If the ticket does not link a specific doc, identify the relevant doc based on the context of the request. Then return to (a)
    c. If a doc cannot be identified from the ticket, request further context from the ticket's submitter.
2. Once the relevant files have been identified, make relevant updates to the file
    a. API Specifications are in `.yaml` or `.yml` files.
    b. Guides and reference docs are generally in `.md` or `.mdx` files
3. Submit a pull request including the relevant changes
    a. Pull request should include the following, at minimum
        i. Relevant Jira ticket
        ii. Changes made in summary
        iii. Draft for release notes
    b. Pull requests made in `bigcommerce/developer-center` should also include
        i. Rollout/Rollout steps
        ii. Testing identifying the CircleCI build[CI]
4. Request review from an SME and a team member.
    a. For requests with a wider impact, review should also be requested from PSE
5. Revise as necessary.
6. Once revisions are complete, Squash and Merge the pull request.
    a. After merging, the pull request will be pushed to the CircleCI pipeline. Await the build's completion.
    b. Once the build is complete, verify a successful update and mark the ticket as Closed. The Jira automations will mark the ticket Done.

Net New, Relocation, or Archival process

1. Generate new content via the General Process.[Arch]
2. Create file links in `bigcommerce/developer-center` for the change.
    a. Net new pages require the path to be built in `/pages` and a `_meta.json` file written to include the Menu data, the file structure, and links to the relevant files in `bigcommerce/docs`. (see below).
    b. Relocations require changes to the path(s), the `_meta.json` file(s), and the redirects scripts (see below).

    c. Archivals require deletion of the path(s) and changes to the redirect scripts (see below).
3. Create a pull request for the relevant changes.
4. Request review from a team member.
5. Revise as necessary.
6. Once revisions are complete, Squash and Merge the pull request.
    a. After merging, the pull request will be pushed to the CircleCI pipeline. Await the build's completion.
    b. Once the build is complete, verify a successful update and mark the ticket as Closed. The Jira automations will mark the ticket Done.

Updating or creating `_meta.json`

Meta files provide three main features in Nextra: structure, order, and metadata.
For all new `_meta.json` files, the minimum setup is

```JSON
{
  "index":{
    "title": "Page Title",
    "docUrl": "path/to/page/file.mdx"
  }
}
```

New entries may be added by following the format of the index page.

The `index` entry displays at the path where you find the `_meta.json` file, while other entries use their key for the path slug.

The order of entries in the `_meta.json` file determines the *order* they appear in the side navigation menu, and the `title` provided determines *what* is displayed.

Updating the redirects

Redirects are handled via a bloom filter. Follow the instructions provided in the repository's `README.md` file. This will require cloning the repository locally and running a custom PNPM script.

Notes:

[CI]:

When a pull request is made in `bigcommerce/developer-center`, it cannot be merged until a successful CircleCI build is made and tested. This is generally automated, but it means updates requiring changes in `bigcommerce/developer-center` must be built at least twice.

[Arch]:

Archivals and path updates should only partially complete the general process, as Squash and Merge operations will incur 404 errors on the Developer Center if the relevant changes haven't been made in `bigcommerce/developer-center` around the same time. The merge steps should be

1. Generate pull request in `bigcommerce/docs`
2. Generate pull request in `bigcommerce/developer-center`
3. Merge
    a. For net new, merge `bigcommerce/docs` then `bigcommerce/developer-center`
    b. For archivals and relocations, merge `bigcommerce/developer-center` then `bigcommerce/docs`

Failure to follow this process will result in some downtime for the affected pages.

Documentation Process

Documentation Process

General steps

7. Jira ticket comes in (feedback, task, or GTM) with a requested update.
    a. If the ticket contains linked doc(s), identify the files for the doc(s) in either `bigcommerce/developer-center` or `bigcommerce/docs` repository.
    b. If the ticket does not link a specific doc, identify the relevant doc based on the context of the request. Then return to (a)
    c. If a doc cannot be identified from the ticket, request further context from the ticket's submitter.
8. Once the relevant files have been identified, make relevant updates to the file
    a. API Specifications are in `.yaml` or `.yml` files.
    b. Guides and reference docs are generally in `.md` or `.mdx` files
9. To preview how a `.md` or `.mdx` file will look in the developer center copy and paste the content into the mdx editor.
    a. If it does not look correct make any changes and copy and paste back into the editor.
    b. If changes are made again in the editor you will have to copy and paste back into the mdx editor.
    c. **NOTE:** This is a pain point that is explained further in the Pain Points tab. This adds extra time to development by not being able to preview locally in our development environment.
10. Submit a pull request including the relevant changes
    a. Pull request should include the following, at minimum
        i. Relevant Jira ticket
        ii. Changes made in summary
        iii. Draft for release notes
    b. Pull requests made in `bigcommerce/developer-center` should also include
        i. Rollout/Rollout steps
        ii. Testing identifying the CircleCI build[CI]
11. Request review from an SME and a team member.
    a. For requests with a wider impact, review should also be requested from PSE
    b. If the reviewer does not have access to GitHub and/or asks for a preview link then you will need to copy and paste the markdown content into a google doc and send them a link to review.
        i. Once changes are approved and agreed upon in the google doc, you'll have to revise as necessary and ensure you copy and paste the updated content back into the markdown file.
        ii. **NOTE:** This is a pain point that is explained in the Pain Points tab. This is due to the repositories being separate. A lot of the teams (especially

catalyst) want to see a preview link of the document as it will be in the developer center. This is not possible right now with the current setup and causes extra time of copy/pasting and setting up reviews in other places.

12. Revise as necessary.
13. Once revisions are complete, Squash and Merge the pull request.
    a. After merging, the pull request will be pushed to the CircleCI pipeline. Await the build's completion.
    b. Once the build is complete, verify a successful update and mark the ticket as Closed. The Jira automations will mark the ticket Done.

Net New, Relocation, or Archival process

7. Generate new content via the General Process.[Arch]
8. Create file links in `bigcommerce/developer-center` for the change.
    a. Net new pages require the path to be built in `/content` and a `_meta.tsx` file written to include the Menu data and the file structure in `bigcommerce/docs`. (see below).
    b. Relocations require changes to the path(s), the `_meta.tsx` file(s), and the redirects scripts (see below).
    c. Archivals require deletion of the path(s) and changes to the redirect scripts (see below).
    d. All updates of this type require an update to the prebuilds rewrite file at `/scripts/prebuilds/routes/docs.json` in order for pages to load correctly.
9. Create a pull request for the relevant changes.
10. Request review from a team member.
11. Revise as necessary.
12. Once revisions are complete, Squash and Merge the pull request.
    a. After merging, the pull request will be pushed to the CircleCI pipeline. Await the build's completion.
    b. Once the build is complete, verify a successful update and mark the ticket as Closed. The Jira automations will mark the ticket Done.

Updating or creating `_meta.tsx`

Meta files provide three main features in Nextra: structure, order, and metadata.
For all new `_meta.tsx` files, the minimum setup is

```typescript
import type { MetaRecord } from "nextra";  // To make the datatype available locally
```

```
const meta: MetaRecord = {.                // To define an instance of the
datatype
  "index": {                               // For the root path where the
_meta.tsx file is found. This can be mappped
                                           // to any file in the repo so long
as the mapping is defined in the prebuild.
    "title": "Overview"
  },
  "status-codes": {                        // For a sub path without any deeper
nesting. This can be mapped to any file as well
    "title": "Status codes"
  }
};

export default meta;                       // To make the path available in the
tree
```

New entries may be added by following the format of the `intro-to-bigcommerce` page shown, though some other fields may be required for follow-on entries.

The `index` entry displays at the path where you find the `_meta.tsx` file, while other entries use their key for the path slug.

The order of entries in the `_meta.jtsx` file determines the *order* they appear in the side navigation menu, and the `title` provided determines *what* is displayed.

Generally the filename is inferred from the path slug provided added to the path structure where the `_meta.tsx` file is found.

The example `_meta.tsx` file addresses a top-level navigation item. Deeper paths require slightly less configuration up front, but still require the full general file structure.

Updating the prebuild rewrite file

Entries in the rewrite file are strictly for the purpose of identifying a path in `bigcommerce/developer-center` with a specific file in `bigcommerce/docs`. The structure for a given line is

```JSON
"fully/resolved/path/to/page/file.mdx":
"https://raw.githubusercontent.com/bigcommerce/docs/main/docs/fully/resolved/path/to/page/file.mdx"
```

With the exception of certain docs such as betas or docs found within `bigcommerce/developer-center` already, which are referenced by their path within the structure of the `content` folder.

Updating the redirects

Redirects are handled via a bloom filter. Follow the instructions provided in the repository's `README.md` file. This will require cloning the repository locally and running a custom PNPM script.

Notes:

[CI]:

When a pull request is made in `bigcommerce/developer-center`, it cannot be merged until a successful CircleCI build is made and tested. This is generally automated, but it means updates requiring changes in `bigcommerce/developer-center` must be built at least twice.

[Arch]:

Archivals and path updates should only partially complete the general process, as Squash and Merge operations will incur 404 errors on the Developer Center if the relevant changes haven't been made in `bigcommerce/developer-center` around the same time. The merge steps should be

4. Generate pull request in `bigcommerce/docs`
5. Generate pull request in `bigcommerce/developer-center`
6. Merge
    a. For net new, merge `bigcommerce/docs` then `bigcommerce/developer-center`
    b. For archivals and relocations, merge `bigcommerce/developer-center` then `bigcommerce/docs`

Failure to follow this process will result in some downtime for the affected pages.

Review Process

Review Process

Reviews in general are broken down into three groups, each with a different approach, each dependent on the files being reviewed.

Verify the Pull Request Contents (all file types)

- Open the pull request related to the Jira ticket.
- Verify the checks completed successfully.
- Switch to the Files tab and read through the Diff generated for the pull request
    - Red lines are changes. Highlighting on red lines indicates specific changes.
    - Green lines are the new versions. Highlighting on green lines indicates specific changes.
    - Red lines without matched green lines are fully removed.
    - Green lines without matched red lines are fully new.

YAML Specifications

As covered in the Documentation Process, YAML files are set up for API Specification files. These are structured data that is read, formatted, and output for the end user. Our review process for this includes a few steps and a few tools.

Verify a successful render

- Once the diff has been verified, click View File to see the full version.
- Copy the file into one of two tools (preferably both).
    - editor.swagger.io - general swagger format tool. Errors in this editor should be addressed immediately.
    - developer.bigcommerce.com/spec-editor - BigCommerce specific swagger format tool. Each endpoint should be checked to verify the updates do not break the document.
        - This has been broken intermittently since the update to Nextra 4
- Once the rendered version is confirmed, move on to the next file(s).

MDX Guides and References

As covered in the Documentation Process, MDX files are set up for guides and references. These are basic markup documents meant for cross-coding into HTML using a conversion tool built into the developer center.

Verify a successful render

- Once the diff has been verified, click View File to see the full version.
- Copy the file into one of two tools (preferably both).
    - markdownlivepreview.com - general Markdown format tool. Will not handle custom tools, but works for most GFM style Markdown.
    - https://developer-center-f1yhvcd54-bigcommerce-dx.vercel.app/mdx-editor - BigCommerce specific Markdown format tool. Handles all custom tools and available Markdown features along with Nextra components.
        - This really needs to be moved to a live build like the spec editor. If it stays on the staging build it's on, there's a nonzero chance it could be archived or deleted at some point in the future.
- Any issues that arise in these tools should be addressed as requested revisions.

JSON structure files

Generally, we only use JSON for navigation structure (see notes in the two Documentation Process tabs). The main concerns are

- Is it valid JSON? (use a JSON inspector tool)
- Is it valid for its role (verify against local JSON)

TSX structure files

With the migration to Nextra 4, we use TSX for our navigation structure and depend on a single JSON file for mapping files from the `bigcommerce/docs` repository.

Our full file-specific review for this is whether the Dev Center builds successfully and the menu works as expected. For reviewers with more experience in NextJS, a closer review of the object structure may be done.

Finalise a Review Status (all file types)

- If all files have been verified, approve or request changes.
    - Request changes either by explicitly marking it in the review selection or by adding suggestions and comments to specific lines or groups of lines. (see the general github information)
    - Approve with relevant comments as necessary
- Notify the person managing the pull request.

Pain Points

Pain Points

Some pain points exist in the documentation process, from the authoring stage to the review stage and even to publication. A few of those are outlined below.

Split Repository

Concept

Our documentation is split between two separate repositories on GitHub. One is for a ‘static site’ build (`bigcommerce/developer-center`) and the other is for ‘dynamic content’ (`bigcommerce/docs`). Changes to page structure, path structure, or archive content all require updates in both repositories.

Pain Point

Splitting documentation in this manner causes a few issues in active documentation.

- Local development builds are impossible.
    - Running the developer center locally would allow immediate previews of the updates being made, allowing for more accurate expectations for the output.
    - Because the static site repository links directly to the dynamic content repository, there’s no clean way to reference local files in a development environment. So a local build is impossible.
    - A reviewer on the team cannot pull a branch, run the local build, and preview the pull request contents.
- Preview environments require an active build, which takes at minimum 20 minutes.
    - Preview environments are only generated for the static site repository, so only new, relocated, or archived doc changes get a preview environment.
    - Preview links generated require a full rebuild if any changes are made to the dynamic content repository, adding another 20 minute wait.
- Updates that require new, relocated, or archived content require coordinated merging of pull requests to avoid errors on the live Dev Center.

CircleCI Build Failures

Concept

Due to the nature of the build process, failures can happen for a number of reasons, from 429 errors to out-of-memory errors to unexplained timeout errors. These build failures cause two issues: no preview link, and extended deployment time debt.

Pain Point

- When CircleCI returns with 429 errors, builds have to be paused until the GitHub call limit refreshes.
    - There is no indicator of when that call limit will refresh, so this adds 20 to 40 minutes minimum to deployment time.
- When out-of-memory errors are returned, builds have to be restarted.
    - This is not a guarantee of a successful deployment, only a second attempt.
    - This adds 20 minutes minimum to deployment time.
- When timeout errors are returned, builds have to be restarted.
    - This is not a guarantee of a successful deployment, only a second attempt.
    - This adds 20 minutes minimum to deployment time.
- **NEW.** Playwright tests have been failing after otherwise successful builds, which prevents deployment.
    - When a CircleCI build and test is successful, builds are automatically deployed. Playwright failures require manual deployment.
    - **MANUAL DEPLOYMENT MUST BE DONE BY AN `ENGINEER` WITH VERCEL ACCESS.**