Goal

The goal of this document is to provide a general description of the key aspects that the Dataverse SPA Minimum Viable Product (MVP) must meet.

SPA MVP

Functional

Figma User Flows

- Collection
  - View mode
  - Create
  - Edit General Information
- Dataset
  - View mode
  - Create
  - Edit metadata
  - Publish
- File
  - View mode
  - Upload Files

Summary

For the MVP we are going to focus on the Dataset and File, since these elements are included in a collection (dataverse).

To simplify this first version of the product we are going to focus on developing the create and view pages. For the update/edit pages, only the Edit metadata form is going to be included, leaving the rest of the update/edit pages for future versions. Also, the page view will contain only the elements activated by default, the functionality activated via API won't be included for this MVP.

Apart from the Dataset and File views, we are going to include the collection (dataverse) page since it serves as a Dashboard to visualize a list of the collection elements.

Dataset

For the MVP we are going to focus our efforts in the Dataset and File, assuming we already have a Dataverse. For the Dataset we are going to develop the following pages:

- Create (see img. 1)
- View (see img. 2). For the view page we will leave the following elements:
    - Title
    - Thumbnail
    - Citation
    - Access Dataset button
    - Description, Subject, License
    - Files tab
        - Upload Files button
        - List of Files
    - Metadata tab
        - Export Metadata
        - Citation Metadata
- Publish (see img. 3).
- Upload Files (see img. 4).
- Edit Metadata (TODO add img)

Figma frames

Image 1. Form to create a Dataset.

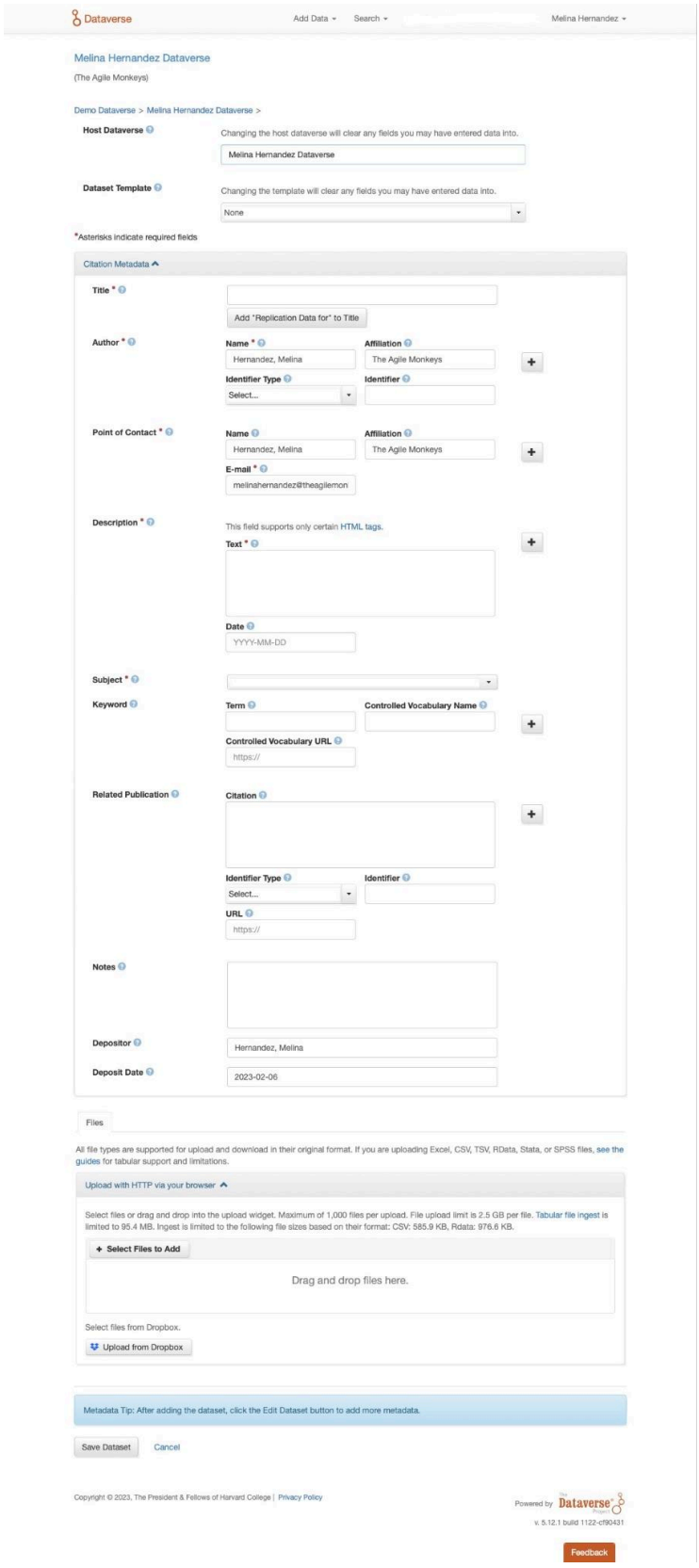


Image 2. View Dataset page.

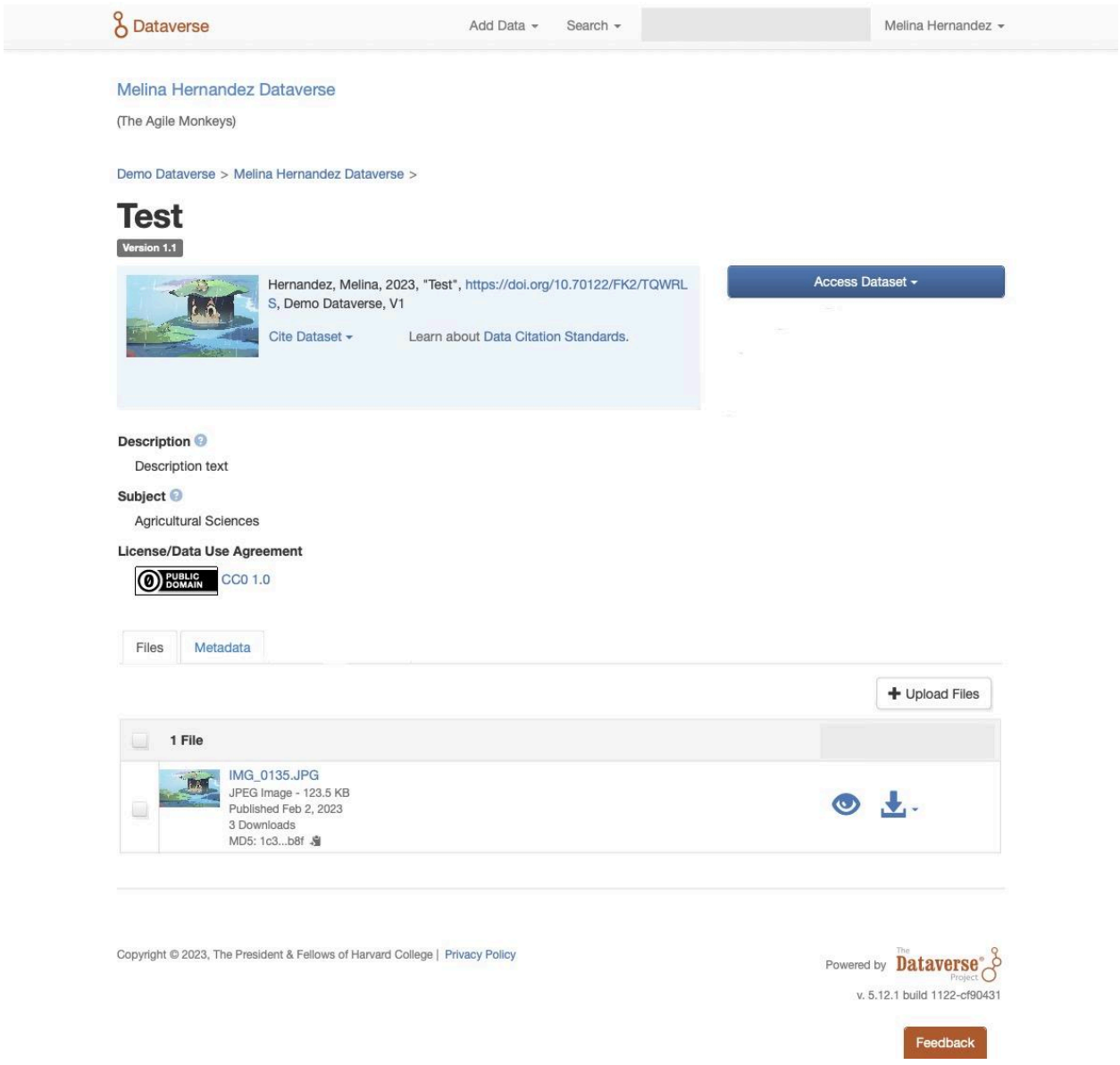


Image 3. Publish Dataset Modal.

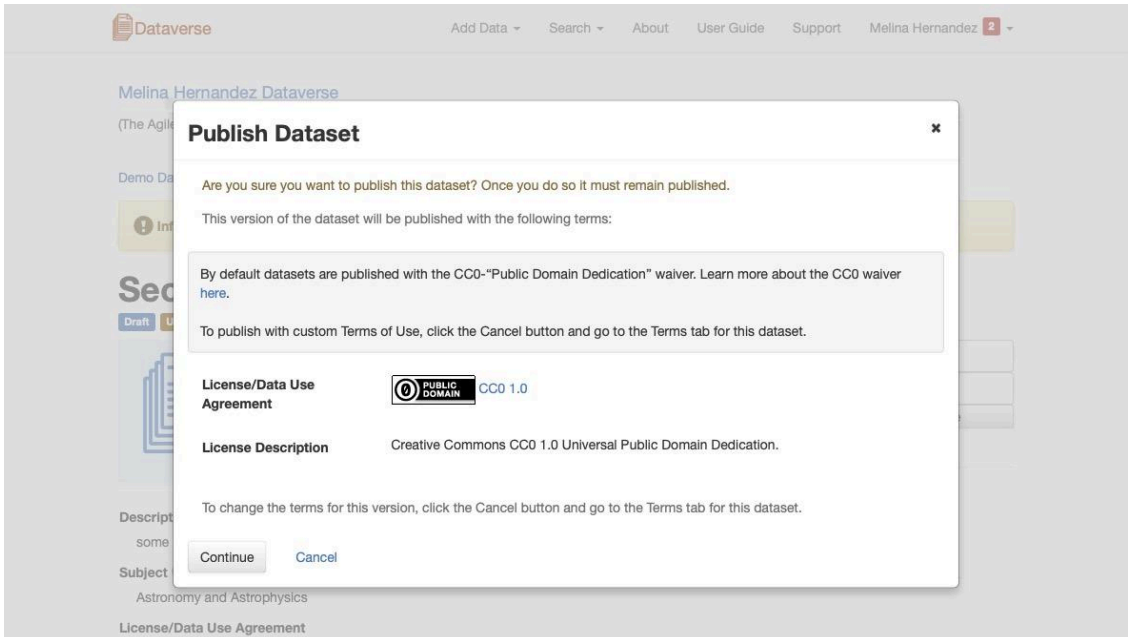
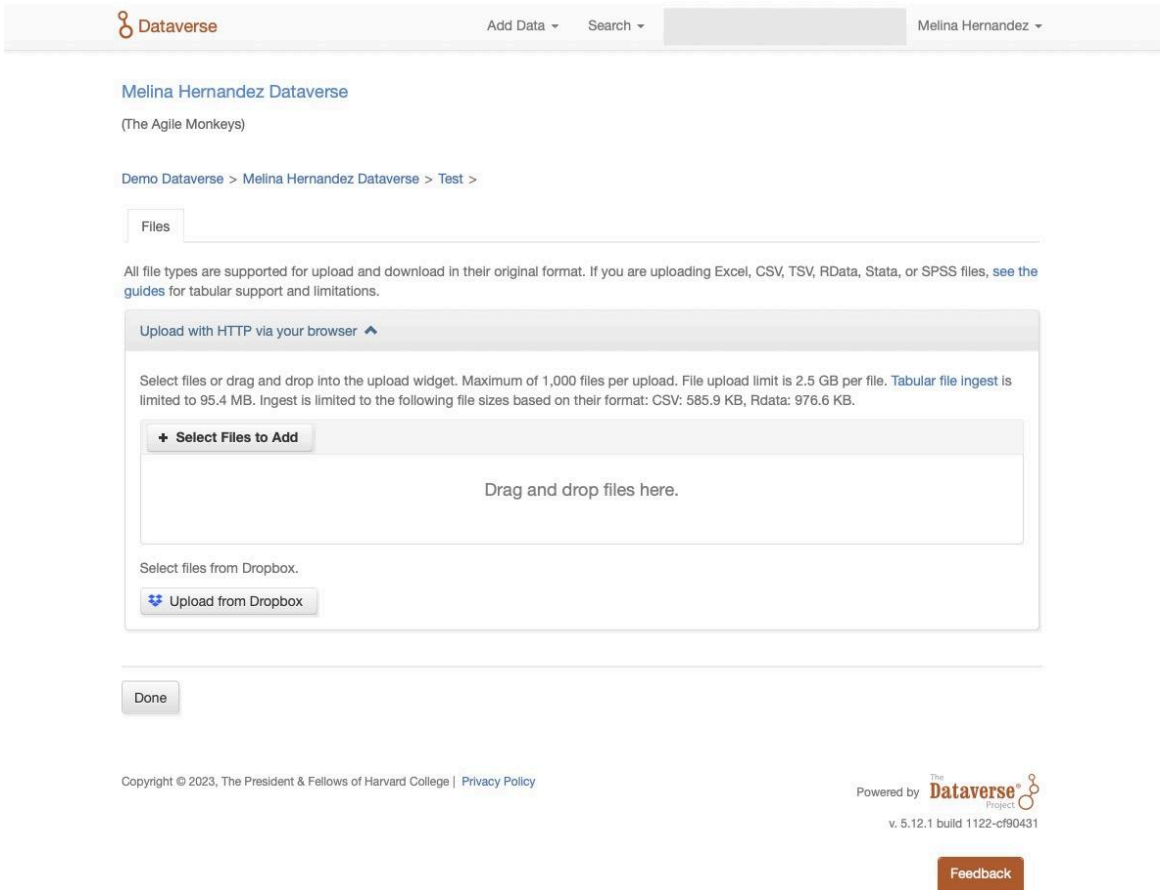


Image 4. Form to upload files to Dataset.



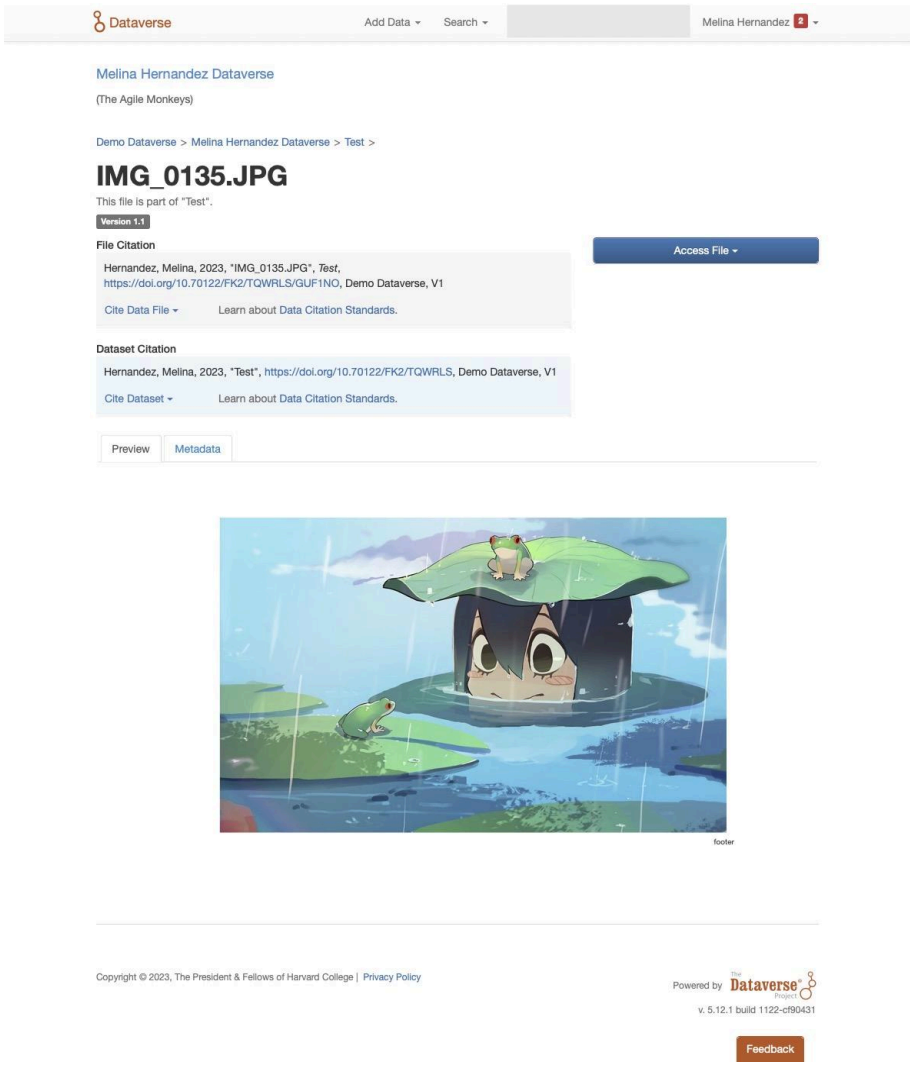
File

A collection of files creates a Dataset. For the File we are going to develop the following pages:

- View (see img. 5). For the view page we will leave the following elements:
    - Title
    - File Citation
    - Dataset Citation
    - Access File button
    - Preview tab. **Without the Explore button**
    - Metadata tab
        - Export metadata button
        - File Metadata

Figma frame

Image 5. View File page.



Collection (dataverse)

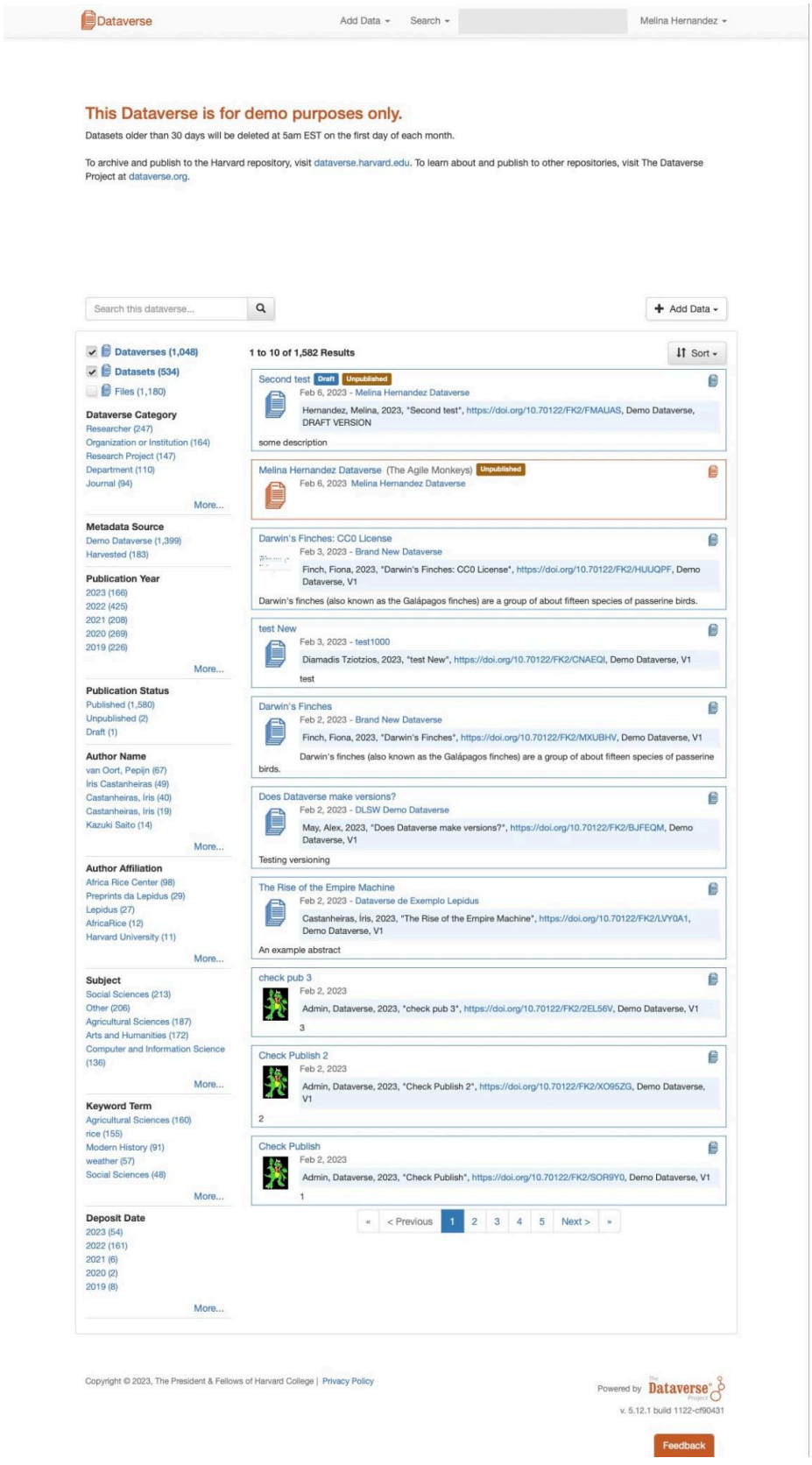
We want to include the collection (dataverse) view in the MVP because it's a fundamental part of the application. It serves as the Home and as a Dashboard at the same time. It allows access to the main features while serving as the visualization of the collection elements.

From the current state of the collection page we want to add the following features to the MVP:

- View (see img. 6).
    - Header with the following sections
        - Add Data
        - Search
        - Sign Up
        - Login
    - Title and Description
    - Search input **without the Advanced Search**
    - Column with the list of filters and categories
    - List of Datasets and Files **without the Dataverses**
    - Add Data button **without the Add Dataverse option**

Figma frame

Image 6. View Dataverse page.



Non-Functional

Summary

For the new SPA, and in particular for the MVP, we must take into account other aspects not directly related to the Dataverse functionalities, but with non-functional aspects.

- Testing
- Security
- Infrastructure & Deployment
- Modularity
- Accessibility
- Internationalization
- Performance

Below we describe the scope we expect to achieve for each of the above requirements in the MVP.

Testing

Testing is a fundamental aspect to consider for the SPA. In order to achieve optimal coverage and ensure the proper functioning of the SPA, we will consider different types of tests and frameworks:

- End-to-end UI Testing: using Cypress framework
- Unit and Integration Testing: using Jest framework + Testing Library
- Accessibility Testing: possibly using https://storybook.js.org

For the MVP and in its first iterations (possibly in its first milestone), it is important to set up these tools to make sure that we have everything we need to properly develop functionality with the associated tests.

In particular, in relation to the overall test coverage, we will establish a minimum test coverage for any repository commit to accept it as mergeable. As we are building an application from scratch, we have a good opportunity to establish a test coverage percentage high enough to ensure optimal overall test coverage. We can use tools such as Coveralls for this purpose, which is already used in Dataverse's main repository.

In addition, we may use other static code analysis tools for other purposes, such as code formatting.

All these tools will be ultimately applied sequentially together within a Continuous Integration (CI) pipeline, which can be implemented using GitHub actions and/or Jenkins.

Although we are going to include automated tests for the UI, we will take into account the manual QA process that will be done to test and ultimately approve and merge changes into the SPA codebase. For this reason, we must pay special attention to the title and description of any issue or associated PR, as well as think about user-oriented issues when creating them, in order to help the reviewing and QA process as much as possible.

Accessibility (a11y) is an important aspect to take into account. As discussed in the early stages of this project, the reimplementation of the Dataverse Frontend is a great opportunity to improve the existing accessibility. We can create accessibility tests as well, including tools such as https://storybook.js.org. This tool has other useful features that we can also use (Generating docs, easy testing, easy components reutilization, etc.).

Infrastructure & Deployment

Containerization

Considering the recent efforts (See #9353 and its related issues) to introduce Docker into the Dataverse ecosystem and its advantages for both the development and deployment of applications, we will consider containerization from the early stages of the SPA development.

Despite this, for local development React already offers a node server that has proven to be very practical during the PoC development. And for deployment, since the built SPA application is about static files, there may not be great benefits in containerization. For these reasons, the real benefits of containerizing/dockerizing the SPA must be evaluated in more detail.

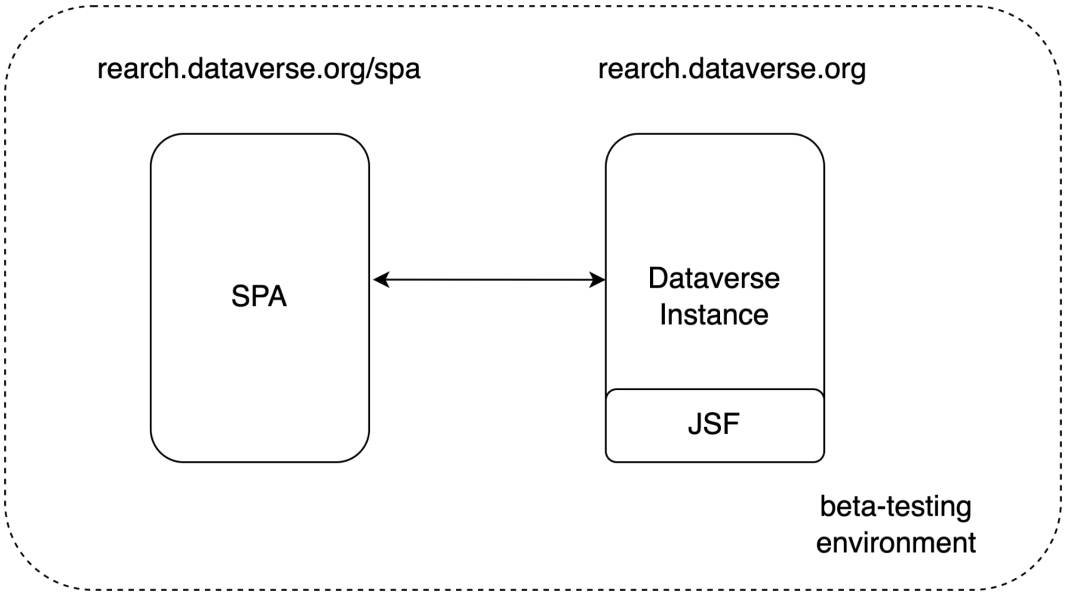
Nevertheless, what we certainly know will be very useful is the Dataverse container-based development environment. The environment will make frontend development easier by abstracting the backend part to an easily installable docker-compose service setup.

Environment

Since the new SPA is going to be accessible only by certain stakeholders and not to the entire community, we will need a remote “beta-testing” environment accessible only to authorized testers. This environment will have the latest version of the SPA deployed so that it can be tested by the team.

For example, if we are developing version “0.3.0” of the SPA, in the “beta-testing” environment we will have version “0.2.0” deployed. Find out more about versioning in the Versioning section.

The “beta-testing” environment must be well differentiated from another deployed Dataverse environment. Since a requirement of the SPA is to make it interoperable with JSF, the “beta-testing” environment must contain not only the SPA application, but also an actual Dataverse instance serving as its API service, which will also contain its JSF frontend (to interoperate with).



The deployed Dataverse version (in the previous example at: rearch.dataverse.org) must be compatible with the deployed SPA version. For example, if the deployed SPA version is “0.2.0” and that version requires a particular extended endpoint to work, which is added in Dataverse “5.14.0” version, the deployed Dataverse version should be “5.14.0”.

Current related issue: #9347

Continuous Delivery (CD)

Although we are using a beta-testing environment, the MVP must have a Continuous Delivery (CD) mechanism to automate the deployment of the SPA code to the beta-testing environment when there is a new beta version ready to test. This mechanism must be applicable to a real production environment, by only changing the deployment target environment.

This mechanism will be able to be executed isolated or connected to the CI process described above, to achieve a complete CI/CD flow. As for CI, we can use GitHub Actions and/or Jenkins jobs.

Security

All traffic to the SPA, and from the SPA to the Dataverse API must be done over HTTPS/SSL for the involved applications (both the SPA and the Dataverse backend) deployed on the beta-testing environment.

Authentication

Given the ongoing work to redesign Dataverse Authentication, for the SPA MVP we will not implement new authentication flows, such as the OIDC-PKCE flow proposed in the new authentication model. Instead, and in order to speed up the development of the SPA functionality, we will reuse the current authentication model based on the Dataverse backend sessions.

Dataverse backend sessions are related to a JSESSIONID cookie which is settled at login time, regardless of the chosen login mechanism (builtin, OIDC, Shib, etc). Since both the SPA and the Dataverse instance share the same domain, the JSESSIONID cookie is shared as well.

Therefore, to be able to use the SPA as an authenticated user, it will be necessary to first log in via JSF login page. We have already discussed the potential security risks related to this authentication model (CSRF). Since the beta-testing environment is a closed environment, these risks are greatly minimized.

For session API authentication to work, it is necessary to enable this mechanism within the Dataverse API authentication logic. Given the security risks and experimental nature of this feature, the session API authentication will not be active by default in any Dataverse installation. Instead, we are going to wrap this feature in a feature flag. This work is already in progress - see PR #9290, which depends on PR #9299.

However, while we are not going to implement production-oriented authentication flows in the SPA MVP, we must consider the future OIDC-PKCE flow in the MVP development, so we must prepare the SPA internal design to properly evolve to this future phase when the new Dataverse Backend authentication model is finally implemented.

Internationalization

Dataverse has users all around the world so we need to take care of the internationalization. This can be done mainly by translating the texts of the UI according to the user location. To achieve that we are going to implement i18n in the SPA from the beginning, in the first steps of configuring the React application.

For the moment, we only need the bases to work with i18n. Internationalization is an extense issue to be continued after the MVP milestones, but it is important to have i18n configured from the beginning because translated texts affect most of the components of the application. For the scope of the MVP only the English language would be configured.

Some of the most popular i18n libraries for React are react-i18next with 2.3M weekly downloads and react-intl with 1.2 M weekly downloads. Both of them have great documentation and are good candidates for implementing the internationalization solution. However react-i18next is more popular and it already has integrated solutions to detect the user's language.

Apart from that, we need to take into account that there is already a solution written in Java for i18n. For the MVP we should try to reuse as much as we can from this implementation.

In a Post MVP effort, Weblate can be integrated for greater control and management of the different languages and translations.

Modularity

Dataverse API Client module

Since the early discussions about modularity in the Dataverse SPA, we suggested the idea of creating a Dataverse API Client module / library, which would encapsulate the Dataverse API access logic so that it can be reused by other applications other than the Dataverse SPA.

Currently there is already an npm module for this: https://www.npmjs.com/package/js-dataverse. Actually, this module is not heavily used and is a bit old. We need to analyze it in more detail to decide whether to build on top of it and continue its evolution, or redo it from scratch considering the design and functionality that the module already provides.

Since this module would be useful to any other web application that requires access to the Dataverse API, we will start building it from the beginning of the SPA MVP development. Since it is decoupled from the SPA, it will have a separate repository and versioning.

UI modules

UI components and styles may be good candidates for extracting them to libraries or modules of base UI resources, which can be helpful for the community by potentially being reused in the development of custom UIs.

For this kind of modules, we can build them in an emergent way, not from the beginning, identifying the UI elements we consider reusable as we develop and include them into the SPA MVP.

Third-party UI integrations

Since MVP will not cover functionalities that need to integrate with third-party applications, either through iframes or custom components, we won't cover this aspect of modularity for now. We will do it in post-MVP iterations.

Versioning

The SPA is going to have its own versioning system since the deployment is going to be detached from the backend. This will allow us to know what has been modified from one version to another. The Semantic Versioning standard although it was designed with APIs in mind can be applied also to a frontend application with a few changes.

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make significant UI redesigns.
2. MINOR version when you add functionality.
3. PATCH version when you make bug fixes.

We removed the word compatible from the numbers definition in Semantic Versioning because for the frontend it doesn't make much sense. For the frontend, compatibility would mean that the user understands the new UI compared to the previous version. If the core components of the application remain the same then it would be compatible with the previous UI version. This means that all new features would be included in MINOR while there is no big UI redesign.

The version should be stored in the package.json itself because it is read during the build. At the same time versioning could be used for the catching, as we know browsers catch the compiled files so when a new version is released there can be problems with users not seeing the latest changes. If we add the version to all the build files, then they would be updated on the browsers everytime a new version is released. Although that is not completely true, since an SPA does not refresh the page when navigating through it, the new versions won't be automatically updated unless the user refreshes the page.

To follow this versioning systematically it is recommended to use a branching strategy to automate it. This branching strategy would depend on what point we want to get with the CI. Gitflow Workflow works for long-lived branches to manage feature branches, but it requires a really organized collaboration and can generate a lot of conflicts, it can be more challenging to work with a CI/CD approach. Trunk-based is now the standard since the branches are oriented to live less and the changes are integrated immediately hidden by feature flags, is the best way to work with a Continuous Integration development.

For the Release Notes we can automatically generate them since the changes should be described in the merged PRs. GitHub provides Github Releases that automatically generates the notes and can be configured to use the labels as a way to classify the PRs changes, for example we could configure a label for the current milestone. We can also create a GitHub Workflow to create the automatic Release Notes but this may be out of the MVP scope.

Performance

Performance can be hard to track in SPA applications since they do not refresh the page. Most of the performance tools would track only the initial page load but not the navigation to other routes in the application. For this reason performance should not be measured in the same way as with traditional websites.

Rendering

For the first load the SPA needs all compiled files and initial information, which means it would take time to launch. Common metrics would be worse than for a MPA because of the SPA nature, but this doesn't mean that first load performance is impossible to be optimized. One of the most common practices is **Lazy Rendering**, this means rendering only the parts that are visible for the user in the initial view port. This practice is useful if the SPA spends a lot of time for the initial rendering. This concept can be taken to API calls, as Lazy fetching, so those data fetchings taking longer can only be triggered when the user gets to that part of the application. You only render what is actually needed.

Another approach would be to render the whole page but **assigning priorities**, so the view port is finished for the user's first interaction, while the rest of the application is still rendering, this is a good approach if the page is large.

Another aspect to be considered regarding the performance is the **Layout Shift**, it affects the user experience as well as the page metrics. It is really common in SPAs since there are many elements dynamically rendered. It should be taken into account every time a conditional render is included.

For the renderization of large lists or large tables we could use **Virtual Lists** that only renders the part of the list that the user is visualizing at the moment, rendering the information once the user scrolls the page until that part. For React one of the most popular libraries would be React Virtualized.

For the scope of the MVP these practices should be considered, even if the application is not large enough to have performance issues with the initial render, any component that takes longer to render should be a candidate for these optimizations from the beginning, before there are more components.

Monitoring

One of the easiest ways to identify performance issues in a SPA is monitoring the components renderization during the development. The React Profiler is a powerful tool to identify performance bottlenecks. It should be always checked while developing a React application because most of the performance issues in SPAs appear from components re-rendering too many times without no one noticing it. For example there could be infinite loops for a wrong use of the React hooks but from the user perspective the application would seem to be working as expected. Tracking and monitoring the SPA state during the development is key to ensure the best performance.

File Upload/Download

Files management is an important part of the Dataverse, the performance has to be great even for large files. For the SPA MVP this should be taken into account to ensure the best performance from the client side.

Files can be uploaded via HTTP request to the server or via presignedURL directly to the S3. From the frontend perspective some libraries for the requests have integrated React Hooks to check the progress of the file upload, this progress can be used to show the user a progress bar of the upload process.

This is going to be a work for both the backend and frontend to find the best optimization for the files upload.

Pagination

Datasets can have a great amount of files which means possible performance issues. The community has reported slow loading screens for large datasets [TODO - attach any issues or comments].

Larger datasets visualization needs to be paginated to improve the response times. This issue should be approached since the beginning of the SPA development. This may require the backend to return paginated responses as well.

From the SPA perspective, if there is no pagination provided by the API then large lists can be efficiently rendered with Virtual Lists (see Rendering section).

Caching

Caching could be one of the major improvements to the performance of the application since most of the Datasets data is not going to change frequently. Caching can be a huge issue and it may be out of the scope of the MVP, but we can make some considerations from the beginning.

Async calls are usually managed in React with the useEffect Hook which means that the API requests are repeated each time the component is re-rendered, even if there is no new information in the server. API requests can be cached to improve performance using libraries like React Query.

In React there is a simple Hook for memoization of variables to avoid unnecessary computations on re-render, the useMemo() Hook.

If at some point of the development is needed to save data on the localStorage then there are tools to caching this data. Specifically for Redux storage the Reselect library can memoize selectors.

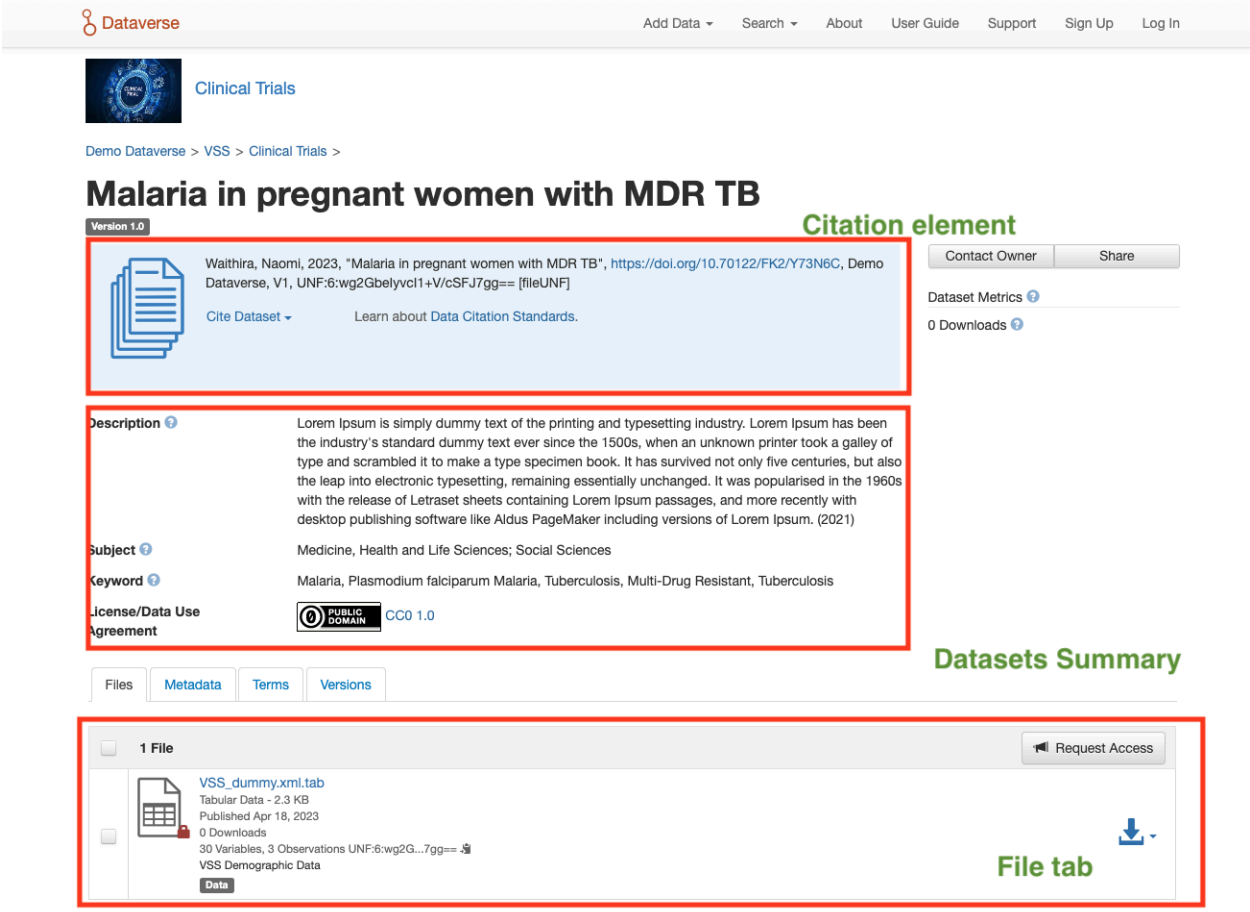
Process for replicating UI features

Note: This process starts to be applicable in Milestone 2, where we start replicating the Dataverse UI features on the SPA, specifically the Datasets page.

UI pages have different components, sections and display modes, so the goal of this process is to ensure that we properly plan and divide the UI feature replication work, and that the issues we create are well scoped, avoiding creating big ones that can be hard to estimate.

A Dataverse page can have different visualization modes (view, edit, or create). The process for replicating a page starts by replicating its associated view mode, followed by its create and edit modes, if they exist.

Conceptually, modes can be treated as separate pages, which have different UI elements dependent on different Dataverse API endpoints to perform CRUD operations. Taking the Datasets page as an example, and focusing on the View mode, we can identify different UI elements:



In the image above we have only identified three elements, as an example, but we could identify more. The goal is that each of these elements can be developed independently, in an end-to-end way:

- [UI] Create the React UI component (and child components, if necessary) and related logic.

- [Data access & API] This step is optional, depending on whether the associated API operations have already been implemented in the js-dataverse module or not.

    - Create the necessary use cases for the particular element in the js-dataverse package.

    - Create / extend the associated Dataverse API endpoints: This step is optional, and applies when the API has deficiencies in terms of returned payloads, write operations, or optimizations.

The points above can be translated into issues. So potentially, the development work of a particular UI element breaks down into the following issues to create:

- The parent issue. For example: ***Create the citation element of the Datasets page***

  This issue is created in the dataverse-frontend repository. Within its scope, in addition to the development of the UI logic (React code), the analysis of the data access logic to develop the element is also considered. Once the analysis is done, the following spike issues may be created, in case it is necessary to develop data access logic:

    - Js-dataverse spike issue. For example: ***Create a use case for retrieving citation data.***

      This issue is created in the dataverse-client-javascript repository.

    - (Optional) Dataverse API spike issue. For example: ***Extend the citation endpoint payload for including "X" field***

      This issue is created in the dataverse core repository.

  The result of the analysis for a specific UI element must be added to the parent issue, as well as the links to the spike issues in case they are created. It is important that the different issues are correctly linked to each other, since they will be created in different repositories and this allows a better monitoring of the workflow.

  A parent issue can be closed when the UI development is done, and the associated spike issues are closed.

GitHub issue template (To add to the repository):

```
## Overview of the Feature Request
```

```
### Mockup of the requested element
### Process action items
_Find out more about the action items in the [process
definition](https://docs.google.com/document/d/1vOSsIGGCbgal4f_lSp1yI
nQifBcmo5vZ-Z9pfkVDyG4/edit#heading=h.e6y69fx5m61r)_
- [ ] Analyze required data access logic
- [ ] Document associated API endpoints
- [ ] Develop the UI element
- [ ] Add the element to the page and storybook
- [ ] Create spike issues if required and add them to _related open
or closed issues_
## What inspired the request?
## What existing behavior do you want changed?
## Any brand new behavior do you want to add to Dataverse?
## Any open or closed issues related to this feature request?
```

The endpoints associated with each element will be documented to keep track of the relationship between the SPA and the API. This can be very helpful for future developers.

There may be elements within a page that share data access logic (same API endpoints). To enhance autonomy in the development of the elements, they will be developed without taking into account the possible duplication of js-dataverse calls (API calls).

Once a page is finished, we will optimize the associated elements to avoid duplication of API calls, through caching or component communication mechanisms. (See: Caching).

Milestones

Milestone 1: The Set Up

This milestone will be focused on generating the tools for the future work. At the same time we should test those tools against a simple page. We can test the end to end with a simple Layout page, having only the header, title, description and body structure, without getting into too much detail.

Completed

- **[issue #1]** SPA React Skeleton (Initial estimate: 2 weeks)
    - Create SPA with Create React App with the TypeScript template
    - Set unit testing libraries Jest and Testing Library
    - Set up e2e testing with Cypress

    - Set internationalization with i18next
    - Set linter and formatter
    - Setup the React Router to navigate through the SPA pages
    - Setup SASS and Bootstrap for the CSS stylesheets

- **[issue #10]** : Vite migration (3 days)

- **[issue #3]** Set up Design System UI Library with StoryBook (Initial estimate: 1 week)
    - Setup StoryBook to document UI components and add accessibility tests to the UI

- **[issue #5]** Establish the QA process for the SPA (Initial estimate: 1 week)
    - Define QA guidelines for the review of the SPA pull requests

- **[issue #2]** Configure the dataverse-frontend Github repository (Initial estimate: 0.5 weeks)

- **[issue #19]** Add a test coverage measurement mechanism (1 day)

- **[issue #7]** The Layout component (Initial estimate: 1 weeks)
    - Create an initial Layout component as a wrapper for all the subsequent page components of the SPA. This Layout component styles the header + body + footer of all the rest of the SPA pages.

- **[Deliverable: Infra & Deployment Setup]** Design and configure the underlying infrastructure and the deployment of the application on a remote environment. (Initial estimate: 1 week)
    - [issue #17] : Add a mechanism for automated deployment to S3
    - [issue #18] : Add a mechanism for automated deployment to Payara

- [issue #20] : Upgrade Storybook from v7-beta to v7

[The next milestones are tentative and likely to change]

Milestone 2: Dataset page on View Mode

Now that the SPA skeleton and tools have been set, we can start developing a core feature: creating a Dataset. This Dataset feature will only include the essential features described in the document above: view, create, publish and edit metadata.

At the same time we are going to develop the Dataverse Page to display the Datasets.

Completed

- **[issue #27]** Analyze Storybook Testing Capabilities for UI Testing Integration

- **[Deliverable: API connectivity]** Review/Redo the Dataverse client npm module implementation - This would depend on whether the module has to be completely redone or not

    - [issue #13] Analyze js-dataverse module and decide whether to build on top of it or redo it from scratch

    - [issue #39] Create a npm module for a Dataverse API client so the frontend application and other community applications can use it

- **[issue #53]** Log In / Log Out mechanism
    - [issue #9531 (dataverse)] Add a Log Out endpoint available when the session API auth feature flag is enabled
    - [issue #56 (js-dataverse)] Add use case to get the current authenticated user given an auth cookie

- **[issue #65]** Automated dev environment for Dataverse

- **[issue #76]** Migrate remaining Vitest tests to Cypress component tests

- **[issue #12]** Add version number to bottom right of each page

- **[Deliverable: Design System]** Create the UI basic atoms such as buttons, tabs, form inputs and add them to the Storybook
    - [issue #14] : Identify UI elements for the Dataverse Design System and create issues for their development
    - [issue #28] : Create a npm module for the Dataverse Design System so the community can use it for their applications and future plugins
    - [issue #32] : Create the buttons and dropdowns of the design system
    - [issue #30] : Create the design tokens of the design system using SASS variables
    - [issue #38] : Create the alert element of the design system
    - [issue #37] : Create tooltips of the design system
    - [issue #36] : Create the modal element of the design system
    - [issue #35] : Create the accordion of the design system
    - [issue #34] : Create the tables of the design system
    - [issue #33] : Create the navigation elements of the design system
    - [issue #31] : Create the form elements of the design system
    - [issue #40] : Add mdx template for the Storybook documentation of the design system elements

    - [issue #45] : Create the ButtonGroup of the design system
    - [issue #47]: Create Cypress component tests for existing UI components
    - [issue #52]: Replace in coverage check Vitest by Cypress component tests

To be completed

- **[Deliverable: Dataset]**
    - Dataset [View Mode : Info]
        - [issue #58] Create the boilerplate of the Dataset section with Dataset title.
            - ~~[issue #71] Spike - Analysis~~
            - ~~[issue #78] Spike - Frontend~~
            - [issue #89] [Spike - Backend] Backend support for UI components related to the dataset information
        - [issue #59] Create the citation block of the Dataset
            - ~~[issue #72] Spike - Analysis~~
            - ~~[issue #82] Spike - Frontend~~
            - [issue #89] [Spike - Backend] Backend support for UI components related to the dataset information
        - [issue #60] Create the Dataset Summary
            - ~~[issue #73] Spike - Analysis~~
            - ~~[issue #83] Spike - Frontend~~
            - [issue #89] [Spike - Backend] Backend support for UI components related to the dataset information
        - [issue #62] Create the Dataset Metadata Tab
            - ~~[issue #75] Spike - Analysis~~
            - [issue #84] Spike - Frontend
            - [issue #89] [Spike - Backend] Backend support for UI components related to the dataset information
        - [issue #89] [Spike - Backend] Backend support for UI components related to the dataset information
            - [issue dataverse#9588] [Spike - API] - Extend Dataset API endpoints to support the Dataset page (SPA)
            - [issue js-dataverse#59] [Spike - js-dataverse] Add new Dataset use cases to support the Dataset page (SPA)

        - [issue #95] [Spike - Frontend] - Invoke js-dataverse use cases in the Dataset info UI components and integrate all components in the Dataset page

        - [issue #106] Improve the containerized development environment

    - Dataset [View Mode : Files]

        - [issue #61] Create the Dataset File Tab
            - ~~[issue #74] Spike - Analysis~~

            - [issue #94] Spike - Frontend

                - [issue #107] [Spike - Frontend] [0/2] Create the File Tab of the Dataset - Analysis

                - [issue #104] [Spike - Frontend] [1/2] Create the File Tab of the Dataset page - Display data

                - [issue #105] [Spike - Frontend] [2/2] Create the File Tab of the Dataset - Add filters, checkboxes and actions

            - [issue #90] Spike - Backend

        - [issue #63] Add all downloading options to Dataset page in view mode

- [issue #100] Refactor DatasetSummary to use the same translations files as the DatasetMetadata

Milestone 3: Dataset page on Create and Edit Modes

- Create new Dataset form (1 week)
- Edit Dataset metadata form (0.5 week) - Should be the same as the create form
- Create the Publish Dataset modal (0.5 week)
- Setup Continuous Integration (CI) mechanism (1 week)

Milestone 4: Collection

The Dataverse Page represents a collection of Datasets. In this milestone this page is going to be developed as well as the file upload..

- File
    - API extension and optimization + js-dataverse extension (3 weeks)

    - File page (1 week)
    - File Upload (1 week)
    - File Previewers (iframes) (1 week)

- The Dataverse Page

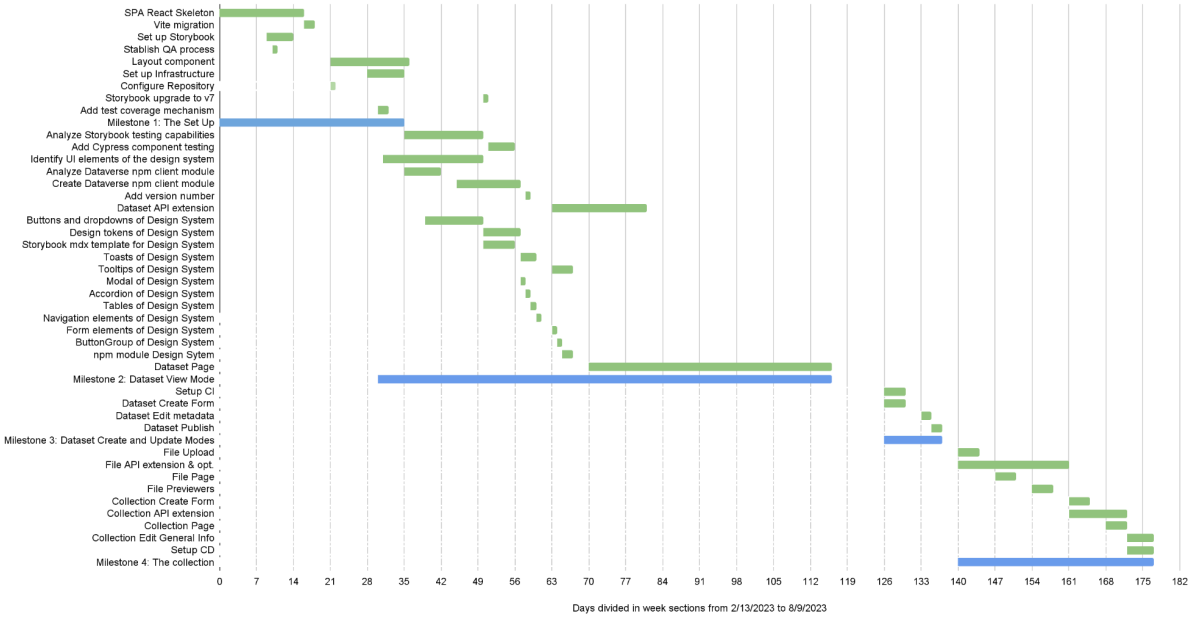
    - UI elements:
        - TODO
        - Add breadcrumbs to the Layout component

    - API extension + js-dataverse extension (2 weeks)
    - Collection page (1 weeks)
    - Create new Collection (1 week)
    - Edit Collection General information (0.5 weeks)

- Setup Continuous Delivery (CD) mechanism to automate the deployment of the SPA code to the beta-testing environment. (1 week)

Milestones estimation charts

Milestones Best Scenario



Milestones Worst Scenario