

Maintenance



Owned by [Anita Yang](#), created with a template ...

Last updated: [Oct 20, 2023](#) • See how many people viewed this page

Introduction

Artefact maintenance is a crucial aspect of our product development. By properly maintaining artefacts, it ensures that our team can collaborate, resolve issues, and knowledge share more effectively. This document outlines the guidelines and our practices for artefact maintenance throughout the product development.

Artefacts Overview

Our project encompass a diverse range of artefacts, which form the foundation of our project development. They include:

- Source code files
- Internal Documentation
- UI/UX Designs
- User stories + Requirements
- Bug issues
- Deployment Artefacts
- Version Control Commits

Maintenance Guidelines for Artefacts

Storage and Organisation

- Ensure that artefacts are kept up to date and stored in correct directories.
 - For code:
 - Push to repository whenever new features are implemented
 - Accompany commits with meaningful and insightful commit messages.
 - If a feature is complete, merge into main branch. If a merge conflict occurs, schedule an ad-hoc meeting as soon as possible and manually resolve the conflict.
 - Please ensure files are added to correct (or most logical) directory
 - Please ensure coding conventions are followed to ensure clarity and consistency.
 - For documentation (i.e., non-code artefacts):
 - Add pages to relevant directories (if possible). Try to avoid standalone documents to avoid clutter.
 - If changes occur, update documentation as soon as possible.
 - Please also add a version and outline what changes were made, to the page's version table (version tables found here: [Docu](#)
[mentation Version Table](#)).
- Ensure that redundant artefacts are deleted as soon as possible to minimise clutter and enhance readability.
- Ensure that only authorised personnel have access to stored artefacts.

Ownership and Accountability

- Ensure that at least one person is tasked to maintain an artefact (i.e., code reviews, documentation updates) to allow artefacts to be kept up to date.

- Whoever is assigned to maintain an artefact is held accountable to make any necessary updates or changes to the artefact. Otherwise, the responsibility must be explicitly transferred to another team member who has the capacity to maintain the artefact. This ensures that no artefact is “left behind” and all progress is properly tracked.

Artefact Retrieval

- Provide access instructions on how team members can access and retrieve artefacts when needed.
- As mentioned in “Storage and Organisation”, artefacts should be grouped and structured in a manner that is easy to find and maintain.

Review and Approval

- For code:
 - Before merging into main branch, undergo a code review process to ensure no bugs are present in the logic.
 - As mentioned above, if merge conflicts occur, schedule a meeting to resolve manually.
 - If any problems occur, communicate as soon as possible to fellow group members. If possible, document the issue on our issue tracking system ([Vis-CAT](#)) such that it is addressed.
- For documentation:
 - Proof read documentation before publishing or submitting. This will allow it to be more readable and consequently, maintainable.
 - If any problems arise, schedule a meeting or communicate ad-hoc with fellow group members via communication channels, as soon as possible.
- Schedule client meetings frequently to ensure that product meets expectations and requirements.

TL;DR

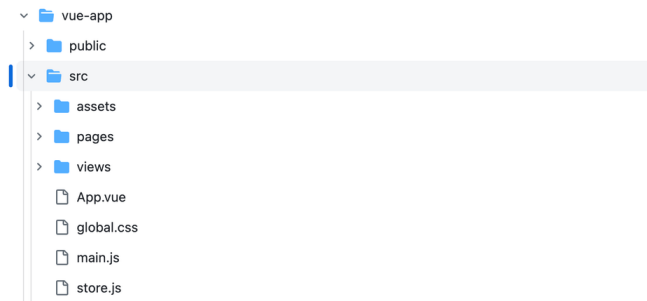
- Please update documentation whenever changes occur. In particular, update meeting notes, designs or requirements, and technical information whenever necessary.
- Please be consistent with agreed-upon coding conventions. Regularly review code to ensure quality is met, issues are addressed, and coding standards are complied with.
- Remove redundant content to minimise clutter.
- If problems occur with either coding or documentation, please share with the group and document this in our issue tracking system.

Our commitment to proper maintenance

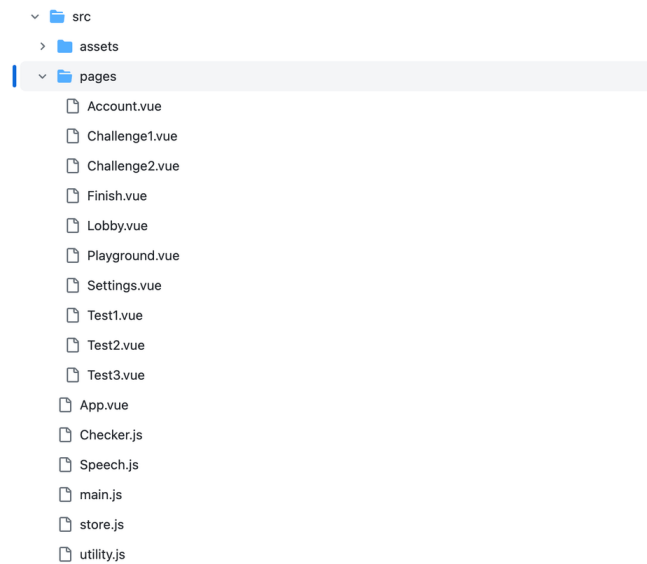
Our systematic approach to maintaining our project artefacts can be seen in our repositories and project documentations. Some key examples, as of Sprint 3 (end of week 12), are listed below:

Code repository

- Source code files and other supporting materials (i.e., .png, .mp3, .svg, .ttf, GIF files) are all organised into a well-structured hierarchy of folders within the GitHub repository. This allows files to be more easily found or maintained during the implementation process, and also allows file paths to be more easily determined when coding (since related files are grouped together).
 - In Sprint 3, we made some slight updates to the structure set in the previous sprint. Specifically, we reorganised the “pages” directory to only contain “.vue” files, and move all “.js” files into the main “src” directory. By doing this, we can make it clearer which files are responsible for handling Javascript logic and which ones are Vue components. This improves readability and makes it easier for developers to navigate the project, ultimately enhancing maintainability.



Sprint 2: source code files are found in “pages” directory, and supporting materials are found in “assets” directory.



Sprint 3: Vue files are found in “pages” directory, supporting materials are found in “assets”, and javascript files are placed in main src directory

- Throughout our project, we frequently committed changes. Frequent commits have enhanced maintenance; this is because as features are implemented across several commits, if some errors or compatibility issues occur, they can easily be rolled back without affecting the development progress. Furthermore, frequent commits provides historical insights into the project's evolution.

 **195 commits**

As of Week 9, this is the number of commits on the main branch (including merged commits)

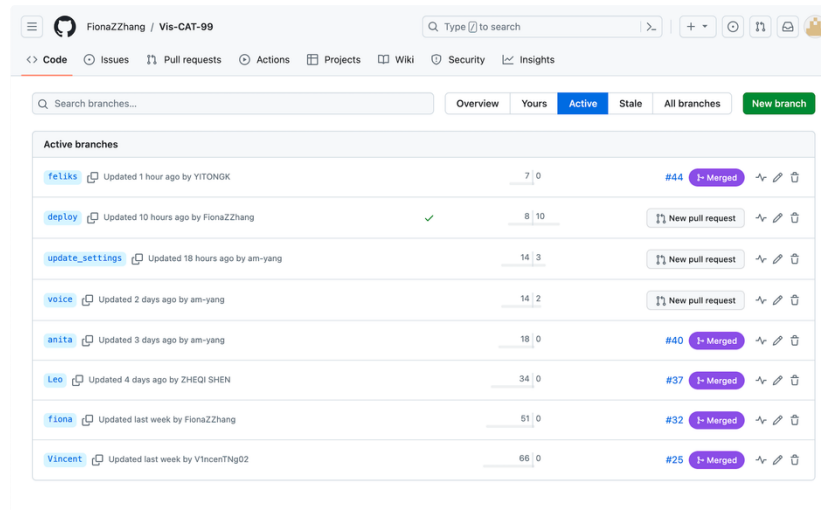
 **269 commits**

As of Week 12, this is the number of commits on the main branch (including merged commits)

- Our commits are also accompanied with meaningful, insightful and coherent commit messages. As a result, they have provided context and insights into why/what specific changes were made, which helps fellow group members understand the rationale behind past code

decisions. This also makes it easier to make necessary updates or review the code in the future. An example of this can be found [here](#).

- Furthermore, Git branches are now use-case oriented. It is worthy to mention that in Sprint 1, Git branches were created for each group member (i.e., we had only had 5 sub-branches, each representing the workload assigned to each group member). While this allowed each team member to contribute to the project development, it lacked description and made it difficult to understand the changes made across different branches. Hence, from sprint 2 onwards, following advice from our mentor, we decided to open and use branches that strictly relate to use cases. This allows for more easy rollbacks, aids understanding of branch contents, and a reduced number of merge conflicts.



- We also widely followed pre-established coding rules (found here: [Coding Standards](#)), which have allowed our source code to be more easily understood and maintained. In particular, we followed:
 - Naming conventions: we used descriptive and meaningful names for variables, functions, files, and other identifiers. This allows items to be identifiable and therefore more maintainable (see *fig 1*).
 - Proper indentation to improve code readability.
 - Code commenting: This was added as necessary, which helps to explain complex logic, provide context, and allow code to be more easily updated or extended in the future. (see *fig 2*)
 - Addition of a Global CSS file, which can be found [here](#). We used this file to store elements that appear multiple times across multiple pages, which helped us reduce repetitions and redundancy in our code. This makes our code more readable, and as a result, more extensible in the future.

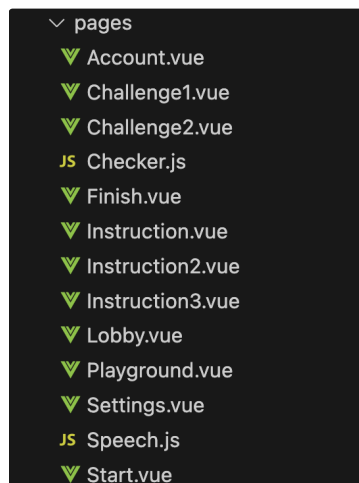


Fig 1. Example of our informative file naming

```
// check whether the tester has drawn the correct pattern for any task
export function checkCorrectness(originalPattern, requirement, userInput) {
  if (originalPattern.length !== userInput.length) {
    return false;
  }

  // unify data type to from array to list
  const targetPattern = originalPattern.map(int => Number(int));
  const userPattern = userInput.map(char => Number(char));
  // change cell id list to coordinate list
  const originalPath = toCoordinateList(targetPattern);
  const userPath = toCoordinateList(userPattern);
}
```

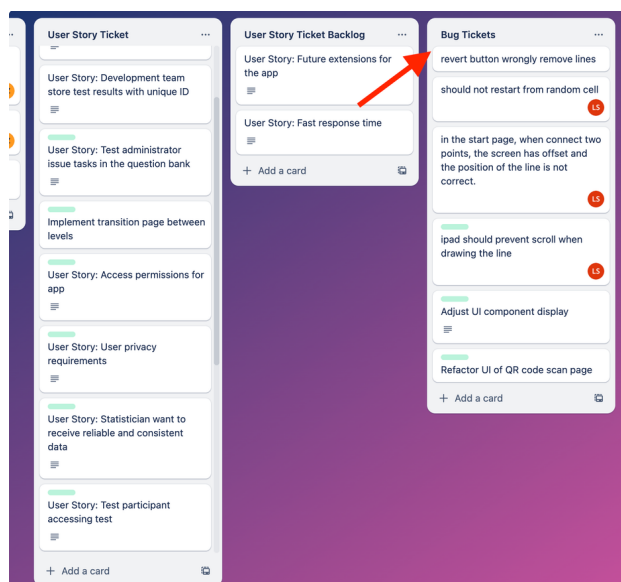
Fig 2. Sample snippet of our use of informative comments

Testing and Quality Assurance


- To preface, our testing plan can be found here: [Testing](#). This artefact is written in a very coherent manner with a very readable layout. As a result, it is easily accessible, which allow group members to quickly reference and facilitate the relevant self-tests to verify the functionality of the software.
- Furthermore, to achieve more comprehensive bug reporting and tracking, we employed real users during our testing phase, and asked them to work through our product. To better document user feedback, we asked users to report them on a Google Form we prepared (found [here](#)). This ensured that issues/feedback are properly recorded and tracked for resolution. Specifically, users were asked to undergo the following processes:
 - Users were asked to first fill out some personal information before completing the test, which are all non-identifiable information. This helped us gauge the general test user demographic, and how these user characteristics may be reflected in their product feedbacks.
 - After completing the test, users were tasked to complete the latter half of the form. This included answering questions about the usability of our product, and how it can be improved in future iterations.
 - By the end of the testing phase, we were able to obtain several helpful responses. They were mostly positive and approved of our UI design. While we did receive some criticism, it was widely helpful and allowed us to consider the issue of seamlessness, which we had not thought about before. More information about our UX feedback can be found here: [UX Report](#).

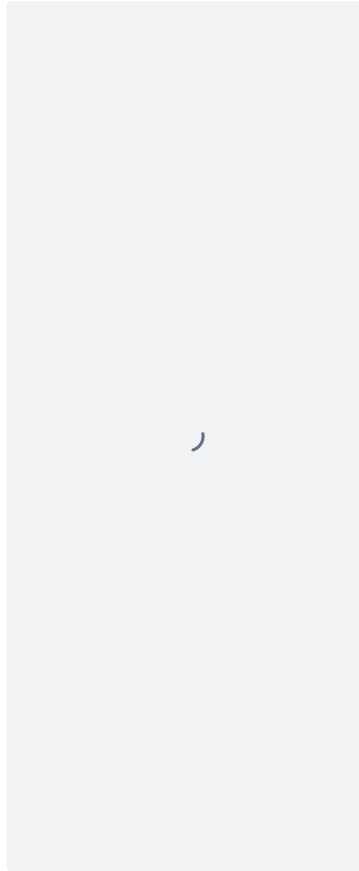
Addressing bugs and issues

- If any issues occur, or is detected by team members, they are encouraged to discuss it via real-time communication channels (WeChat/Slack), and also document the problem on Trello through a bug ticket. This has ensured that any outstanding problems being addressed as soon as possible and supported the maintenance of our development artefacts.



Artefact Retrieval and Accessibility

- For seamless navigation and reference, our commonly-used artefacts can be found on a single Confluence document, which in turn aids maintenance. This page can be found here:  [Wiki](#).
- Our internal documentation artefacts are also categorised into logical groups based on their types and purposes. Specifically, we created a well-structured hierarchy of folders within our Confluence, which has in turn promoted documentation updates and reviews due to its intuitive and easy-to-find structure.



*Demonstration of internal
documentation structure*

- As mentioned in the above section, our source code files are also categorised into logical groups, which also promotes the accessibility of our artefacts.
- We also provided instructions on how team members (or the client) may access and retrieve artefacts when needed. This has allowed all members to seamlessly partake in product development without experiencing many onboarding issues. Key examples include:
 - A detailed, step-by-step instruction on how to compile and run our source code files found on our GitHub README (found [here](#)).
 - A detailed, step-by-step instruction on how to deploy our project (found [here](#)).
- Access permissions: It is also worthy to mention that we set up our artefacts such that only group members (or those with an invite link) have access to our project artefacts. This will prevent unauthorised changes being made to our artefacts from unauthorised individuals. This safeguards the integrity of the project and enhances our ability to properly maintain our documents.

Communication and Feedback Mechanism

- We endeavour to continuously improve our standards of artefact maintenance. To do this, we aim to continuously facilitate team communications and further discover areas of improvement.
- In addition, as we are students that are still learning the conventions of software development, we prioritise feedback given to us by our subject mentor.

Future improvements and goals (as of Sprint 3)

Now that we have completed all business requirements at the end of sprint 3, there are several key activities we will focus on for ongoing maintenance:

- Security audits: we will be conducting security audits periodically to stay vigilant about security threats and system vulnerabilities. This will be especially crucial if more sensitive data is transmitted in the future, which may make the system more prone to interceptions.
- Performance monitoring: we will monitor the performance of our product and ensure that the system remains fast, reliable and responsive as the number of users increases in the future.
- Compatibility monitoring: we will regularly test our product with new devices, browsers, and operating systems. This will help us ensure that our product remains compatible and up-to-date with the latest technology updates.
- User Feedback: We will continue gathering user feedback to ensure that the product can continuously satisfy the evolving needs of the user.

Conclusion

By properly maintaining artefacts through adhering to established guidelines, we have ensured that our artefacts remain organised, up-to-date, and accessible to the entire team. Such efforts have allowed us to stay on track, minimise confusion, and consistently surpass sprint goals.