SE 4485: Software Engineering Projects

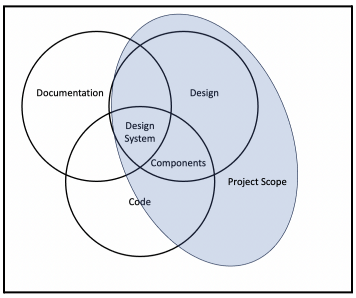
Spring 2024

Final Report

Group Number	3
Project Title	Storybook POC Continuation with Chromatic and Storybook GPT
Sponsoring Company	ARGO
Sponsor(s)	Ponchai Reainthong, Kevin Roa, Raisa Gonzalez
Group Members	1. Lillie McMaster 2. Lauren Nguyenphu 3. Alina Khan 4. Hamdiya Abdulhafiz 5. Iman Sheriff 6. Timothy Naumov

Executive Summary

The following report is the culmination of the Software Engineering capstone project executed by the third group at the University of Texas at Dallas in the Spring of 2024. The project was sponsored by ARGO data members Ponchai Reainthong, Kevin Roa, Raisa Gonzalez as a continuation proof of concept, with the intention of measuring the feasibility of creating and using a Storybook GPT, using ChatGPT-4, to enhance and maintain the documentation inside a Storybook component library. The project ran from January 26th till May 3rd and was completed by Lillie McMaster, Lauren Nguyenphu, Alina Khan, Hamdiya Abdulhafiz, Iman Sheriff and Timothy Naumov.



The project recommends that ARGO continues to use the developed Storybook GPT with the enhanced Storybook documentation, deployed with Chromatic and Github workflows to ensure continuous development, testing, and re-use. During the project's lifecycle, the team enhanced the existing Storybook by adding live Figma integrations, additional components, and dynamic theming for each component.

The project was structured in an agile iterative manner, using Notion to track sprints, work assignments, and development. In contradiction to this, the project fell under waterfall deliverables given by the capstone Professor Wong. These deliverables include the Project Management Plan, Requirement Specifications, Architecture Document, Detailed Design Document, and the Test Plan. The Project Management plan defined the team organization, contemplated the risks analysis of the project, and defined standards in Scholastic Dishonesty, meetings and quality expectations. In the Requirements Specifications, stakeholders like the companies designers, developers, and customers were defined and placed in use case models. The functional and nonfunctional requirements are laid out, as well as our 8 defined use cases. In the Architecture documentation, we define and rationale our architectural styles used in the system: factory method pattern, decorator pattern, and the facade pattern. These are connected to our three systems, the component library, Storybook GPT, and the GitHub Actions. The Detailed Design document gives a comprehensive outline of the Graphical User Interface design, coupled with the Static and Dynamic model of the system. Finally, the test plan outlines testing for newly introduced components, as well as entire system test cases.

Figure 1.2. Storybook Front End

In conclusion, the Storybook Continuation POC sponsored by ARGO and developed by Group 3 was a comprehensive look at the power of Custom GPTs, alongside powerful Storybook integrations and enhancements, can significantly reduce time spent developing new and improving existing components. These tools are not only going to remain automatically upgraded with the improvements in technology, but will also provide consistent automatic documentation and reuse of components throughout the use of the Storybook Library.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Purpose and Scope

The product is a continuation of the ARGO component UX library. The project entails development of a comprehensive GPT to aid in the creation of component creation for the Component library, for seamless integration within Storybook. The objective of the project is to streamline and enhance component creation. Within the defined scope, beginning in January 26th 2024 and ending May 6th 2024, the project aims to achieve the creation of a repository of stories utilizing GPT, designed to address company challenges.

1.2. Product Overview

The product is designed to aid creators within the ARGO team with a comprehensive suite of tools and resources for crafting components. Leveraging the power of GPT, it offers an intuitive experience for generating component designs within both Storybook. The product opens up a myriad of possibilities for scenario-based usage. Including rapid prototyping, iterative design refinement, or addressing specific company challenges.

1.3. Structure of the Document

This document will be structured systematically for readers' clarity and accessibility. The table of contents outlines where contents have been placed. Our Introduction provides an overview and includes sections on Project Management Plan detailing project organization, risk analysis, and resource requirements. Requirement Specifications outline system requirements, while Architecture discusses architectural choices. Design covers GUI and detailed design models, and the Test Plan outlines testing strategies. Evidence of Configuration Management and considerations for Engineering Standards and Constraints are provided, followed by Additional References and Acknowledgments.

1.4. Terms, Acronyms, and Abbreviations

npm - Node Package Manager
GPT - Generative Pre-trained Transformer
MUI - Material UI
GUI - Graphical User Interface
UX - User Experience
QA- Quality Assurance
UC- Use Case
TC- Test Case
CSS- Cascading Style Sheets
HTML- Hypertext Markup Language
POC- Proof of Concept
UML- Unified Modeling Language
GUI- Graphic User Interface

2. Project Management Plan

2.1. Project Organization

The project team is composed of 6 Software Engineering Seniors attending the University of Texas at Dallas. The group leader is Lillie McMaster, determined by luck. The other five teammates are Alina Khan, Lauren Nguyenphu, Iman Sheriff, Hamdiya Abdulhafiz, and Timothy Naumov.
We have several ARGO team members on our side during this project. Ponchai Reainthong will be the leader we report to most often. We will also be teamed with Raisa Gonzalez, and Kevin Roa. Also, we are given the contact of Micheal Hube to email with any additional questions as they are raised. The positions and contact information of each member is listed below. For questions and communications, Ponchai asked us to email him, Kevin, and Raisa all attached in the email.

Name	Position	Email	Phone #
Ponchai Reainthong	UX Director	ponchai.reainthon@argodata.com	972-866-3568
Kevin Roa	UX Engineer I	kevin.Roa@argodata.com	972-866-3470
Raisa Gonzalez	UX Program manager	Raisa.Gonzalez@argodata.com	--
Micheal Hube	UX Designer	michaelhub@argodata.com	--
Lillie McMaster	Group 3 Lead	lnm190001@utdallas.edu	682-213-0134
Lauren Nguyenphu	Group 3 Member	Lan190005@utdallas.edu	469-408-9620
Alina Khan	Group 3 Member	Alina.Khan@utdallas.edu	310-863-0411
Hamdiya Abdulhafiz	Group 3 Member	hxa180025@utdallas.edu	469-422-9863
Iman Sheriff	Group 3 Member	ims190002@utdallas.edu	206-376-1311
Timothy Naumov	Group 3 Member	timothy.naumov@utdallas.edu	469-605-6677

Table 2.1. Group Members and Sponsors

2.2. Lifecycle Model Used

Agile (Scrum)

- We will focus on developing the project in manageable increments. Each increment will aim to deliver a component or functionality of the project.
- Each sprint will have a 1-2 week timeframe and will include a set of deliverables. These will be planned out during our weekly meetings to effectively break down our milestones.
    - Sprint Planning: At the start of each sprint, we will review the project backlog and select tasks for the current sprint. We will then allocate these tasks to each team member.
    - Sprint Execution: During the sprint, each member will work on their allocated tasks. Progress will be updated during team meetings.

    - Sprint Review and Feedback: At the end of each sprint, we will conduct sprint reviews. This will involve presenting deliverables and progress, collecting and discussing feedback from ARGO, and adjusting the plan for the next sprint based on this feedback.
- 2-3 weekly meetings:
    - Tuesday/Thursday: Group meeting via Discord/On Campus
        - Discuss current sprint progress and individual updates
    - Friday: Meeting with ARGO via MS Teams/ARGO Conference Room
        - Conduct sprint reviews, share progress, and collect feedback
- We will collect feedback and integrate it into the subsequent sprints to continuously improve our processes and outputs.

Rationale

We chose an agile approach due to its adaptability and responsiveness to changing requirements. An iterative development will allow for regular feedback incorporation to ensure that we consistently align our work with expectations and project goals. As our project facilitates creativity among our team, this approach will allow us to innovate while maintaining a structured framework to meet our goals.

2.3. Risk Analysis

Risk	Severity	Likelihood	Reduction Strategy
Team member illness	Medium	High - cold, flu, and covid are prevalent in the late winter and early spring.	Communicate with the team and adjust the schedule if needed. Maintain and document all code in case any work needs to be handed off.
Team member academic commitments	Medium	Medium - team members may have other projects, conferences, interviews, etc.	Communicate with the team and adjust the schedule if needed. Maintain and document all code in case any work needs to be handed off.
Team dynamics	High	Low - with the reduction strategy in place, our team is set up to work well together.	Meetings are held twice a week to ensure the team is working well and towards the established goal. Work will be delegated by the team lead. Will reach out to the TAs if needed.
Dependency on shared and paid resources	High	Low - permissions and accesses granted to a shared email for the entire team.	Will contact the team at ARGO if any issues arise.
Errors with infrastructure technology	Medium	Low - technologies being used have redundancy and backup systems in place to mitigate any unforeseen errors	Keep up with documentation and requirements regarding technologies used.
Merge conflicts	Medium	Medium	Utilize branches to separate workload, regularly pull changes from the main branch to ensure code is up to date. Decompose coding objectives into small and focused tasks, leading to more frequent commits with fewer changes in each.

Table 2.2. Risk Analysis

2.4. Software and Hardware Resource Requirements

The Software Requirements include:

- React
    - React is a library that is used to build user interfaces. Our components are written in React and would be implemented in a React application. Many of the other software requirements in the system work within the React ecosystem therefore it is the most compatible library to be used in this project.
- TypeScript

    - TypeScript is a programming language built on top of Javascript that enables strict typing of data structures. This enables better maintainability throughout the development of the library and enforces coding standards that improve the development process.
- HTML
    - HTML (Hyper Text Markup Language) is the underlying language used when react components are rendered in the DOM. React is essentially a layer of abstraction on top of HTML that improves the performance and development of user interfaces.
- CSS
    - CSS is used throughout the components as needed. These Cascading Style Sheets allow developers to further customize the behavior of our UI components as well as the look and layout of the page.
- React MUI
    - MUI is a component library written in React that exports a large amount of commonly used UI components. These components serve as a base for our component library to avoid additional complexity. Most of the components that we export from the ARGO component library inherit behaviors and properties from the MUI components, including their TypeScript types.
- NPM
    - We publish the updated component library to NPM (Node Package Manager), where developers can install the package and consume the components in their React applications. NPM is the standard javascript package manager where we also install many dependencies that are used within the component library.
- Git (GitHub)
    - Our code is version controlled with Git through GitHub. Our work is organized on branches and pull requests are created when features are ready to be merged. Team members have to review pull requests to check for regressions or other bugs that may have been accidentally introduced in the changes. After changes are approved, pull requests are merged into the main branch and GitHub Actions are triggered to deploy the updated component library to NPM as well as the latest storybook documentation on Chromatic.
- Chromatic
    - Chromatic serves as both a platform to review regression testing within the component library as well as a cloud provider for hosting the Storybook documentation. When we first integrated Chromatic into the component library, each component was captured, and new changes were compared to the current stable snapshot of each component. This is how we catch regressions in components and ensure that no changes to the component library break any production environments.
- Storybook
    - Storybook is the primary mechanism for documenting the components in the component library. Each component has a separate page to view the component in isolation. In this view, the developer can experiment with variations of the component with the component controls as well as viewing different themes (such as light and dark theme) as well as the Figma mockups that the component was developed from.
- ChatGPT

    - The StorybookGPT was created through ChatGPT where components and stories are uploaded to be used as context in responses. When the GPT is prompted for new stories given a component, GPT-4.0 along with the uploaded files are used to generate a Storybook story that can be pasted into the component library.
- Figma
    - Figma was provided to us by the ARGO team as design specifications for components. Various component styles are defined in Figma including colors, shadows, sizes, fonts, etc. Implemented components must adhere to the specification and Storybook enables developers to compare implemented components with their mockups side by side.

The Hardware Requirements include:

- Computer
    - All of the component development must be done on a computer.

2.5. Deliverables and Schedule

The primary activities of this project will involve contributing to the existing component library order to improve various aspects of the development process as well as keeping up with course assignments to facilitate the development of the project. Outside of the development of new components and improving the package deployment process, our team will explore how we could create a custom GPT to document components in Storybook. Below is a list of possible deliverables for this project:

- Automate new version numbers with automatic release process using semantic-release https://github.com/semantic-release/semantic-release
- Use TS props-parsing to automatically show components customization in Storybook.
- Explore the use of chromatic for component testing and review processes.
- Create custom GPT for storybook story generation.
- Develop new components from MUI: https://mui.com/material-ui/all-components/

Deliverable	Dependencies	Estimated Time	Allocation of people	Rationale
Project Management Plan	Project overview from Argo kickoff	1 week	6	We clarified expectations and went over the project overview at the Argo kickoff.
Requirements Documentation	Project Management Plan	5 business days	6	After finalizing the project management plan, we need to define what we will accomplish this semester.
Architecture Documentation	Requirements Documentation	5 business days	6	We need to make some important decisions

				regarding the architecture of the component library
Detailed Design Document	Architecture Documentation	7 business days	6	A detailed design document would require us to evaluate our requirements closely and determine the appropriate design.
Test Plan	Detailed Design Document	5 business days	6	As we get closer to the final presentations, we need a collection of tests to demonstrate our work.
Final Project Demonstration	Test Plan	7 business days	6	After completing all deliverables and course assignments, we create a live demonstration of our features and improvements.
Final Project Report	Test Plan	7 business days	6	Along with the final project demonstration, we will create a report to document the completed project.

Table 2.3. Deliverables Schedule

2.6. Monitoring, Reporting, and Controlling Mechanisms

Our team decided to use Notion to document tasks and generate reports for progress tracking. Through Notion projects, we can manage tasks and monitor our progress with a roadmap. Throughout the semester, the roadmap will progress as we contribute new features to the repository. Additionally, we will create burndown charts which will be used to track our progress over time. This will indicate how much work has been done and how much is left to do.

2.7. Professional Standards

Scholastic Dishonesty:

Team members are expected to uphold academic integrity by avoiding any form of scholastic dishonesty such as plagiarism, cheating, or copying others' work. However, if a team is to leverage outside resources or utilize any tools to help them complete their work they should ensure that they credit/cite the work appropriately. Refer to Appendix A for more details.

Rationale: Scholastic dishonesty undermines the team's credibility, and these were the terms discussed with ARGO's sponsors regarding the companies on scholastic dishonesty and plagiarism.

Meeting Schedule:

Team members should actively participate in scheduled meetings and be punctual at all meetings. They should also come prepared with necessary materials, questions, updates or completed sections. If a team member has an extenuating circumstance that prevents them from being present at any meeting, they should inform the group and project sponsors as soon as possible (preferably 24 hours in advance). The team member should also ensure that they take the necessary actions so that they are able to catch up with their work.

The schedule is as follows:

Tuesdays	10:00 AM - 12:00 PM	Informal Team Standup
Thursdays	11:00 AM - 12:00 PM	Required Team Standup
Fridays	1:00 PM - 2:00 PM	Required Sponsor Meeting

Rationale: Meetings are set in advance and are crucial for the team to collaborate, communicate, and make important decisions. Therefore, all team members should attend to make sure everyone is on the same page regarding important topics.

Quality Expectations for Tasks and Deliverables:

Team members are required to produce high-quality work that meets the specified criteria and expectations.

Rationale: Producing quality work is important in order to achieve the team's objectives and maintaining a positive reputation with our sponsors. High-quality deliverables reflect professionalism, competence, and dedication to the project.

2.8. Evidence all the artifacts have been placed under configuration management

The configuration management tool will be Notion. Each project document given by UTD Professors and TA's as well as components and deliverables to ARGO will be delivered through this platform. Meeting notes and attendances will be taken here as well.

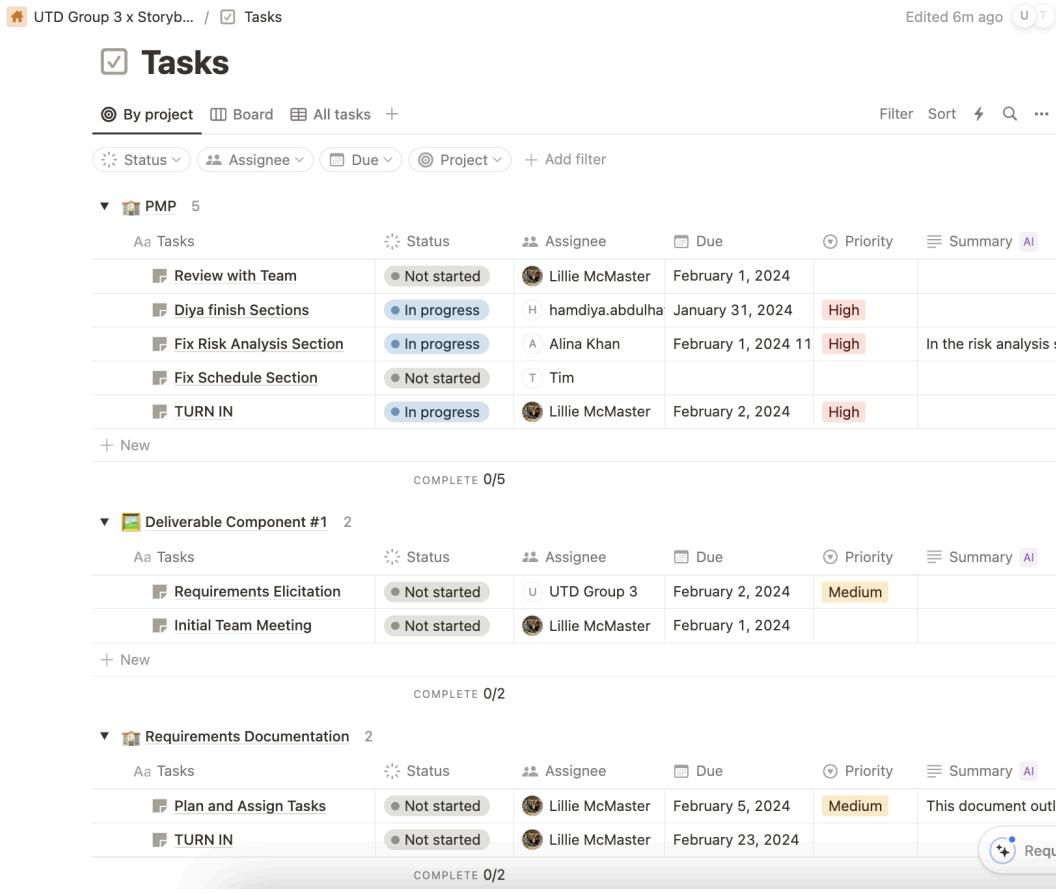


Figure 2.1. UTD Group 3 x Storybook POC Notion Screen shot #1

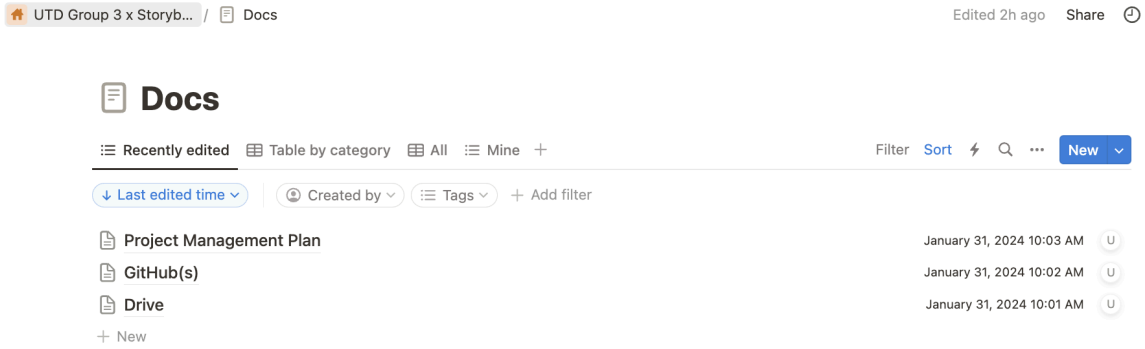


Figure 2.2. UTD Group 3 x Storybook POC Notion Screen shot #2

2.9. Impact of the project on individuals and organizations

Individual Impact:

This project not only has a tremendous impact on all the individuals involved but it also extends its influence to broader societal realms, fostering innovation, efficiency, and collaboration in software development practices. The project's impact on individuals and society is profound and multifaceted. By streamlining their process and offering effective tools to increase productivity, it empowers developers. By using automated story generation with StorybookGPT, we have significantly reduced the time from concept to deployment, and have also increased efficiency, and the overall developer experience. With the integration of Figma, developers and designers can easily create, refine, and test components in real time. Additionally, this project introduced new technologies that many of the team members had little to no experience with; therefore, this project provided an invaluable opportunity for skill development and growth.

Organizational Impact:

The Storybook POC Continuation project is set to bring transformative changes to ARGO's approach to component development and documentation. Traditionally, component development was fragmented, leading to inconsistencies. With the introduction of StorybookGPT, ARGO can standardize the creation and documentation of UI components, ensuring consistent design languages and adhering to the same quality standards. Additionally, with the reduced time required to document components, it not only accelerates the development process, but also cuts down on operational costs associated with the component lifecycle management. The use of Chromatic as a centralized system with Storybook allows for easy management of shared components across projects. This reusability prevents the need for recreation of similar components, saving time and resources.

The implementation of this project is important for enhancing ARGO'S design system. By having a more integrated and automated environment for component development, ARGO sets a precedent for future projects and establishes a framework for maintaining standards in design and functionality. This project is expected to lead to more efficient development processes and an enhanced user experience.

3. Requirement Specifications

3.1. Stakeholders for the system

The stakeholders for the system are:

- ARGO designers
- ARGO developers
- Customers

3.2. Use case model for functional requirements

Functional Requirements

- The system allows designers to review, customize, and finalize components.
- The system enables frontend developers to access and review documentation created.
- The system supports component library developers in reviewing design specifications and writing code for components.
- The system allows input parameters for component stories and generates stories accordingly.
- The system enables component library developers to commit code changes to the component

library repository.

- The system allows component library developers to create pull requests for component changes.
- The system integrates components into the component library after implementation.
- The system supports iterative refinement of components until they meet specifications.

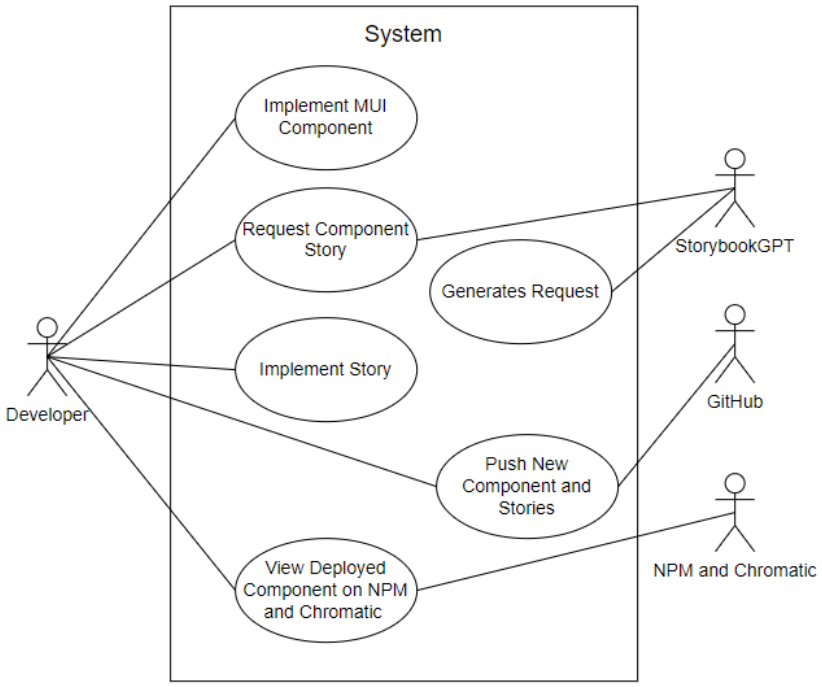


Figure 3.1 Use Case Model For Functional Requirements.

3.3. Graphic use case model

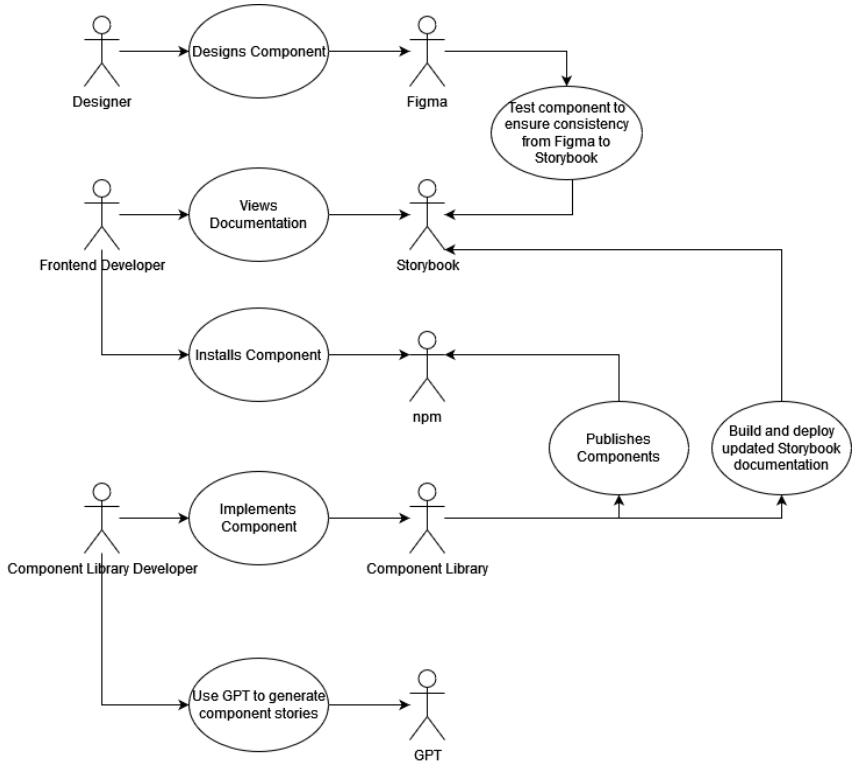


Figure 3.2. Graphical Use Case Model for Functional Requirements

3.4. Textual Description for each use case

UC1: Designs Components	
Participating Actors	Designer, Figma
Entry Condition(s)	A list of components intended for use are created.
Normal Flow of Event	Designer reviews the list of components intended to be designed. Designer designs and customizes the components using figma. Designer reviews, and finalizes designed components.
Exit Condition(s)	The components have been designed and finalized.
Exceptions (Alternate Flow of Events)	The designer has to go back and design another component because one is missing.
Special Requirements	All designed components must be easy to use, and its functionality must align with users expectations and behaviors.

Table 3.1: Use Case 1

UC2: Views Documentation	
Participating Actors	Frontend Developer, Storybook
Entry Condition(s)	Documentation is created.
Normal Flow of Event	Frontend Developer opens up documentation and is able to view it. The front-end developer reviews the documentation and it's approved.
Exit Condition(s)	Documentation is viewed, and approved.
Exceptions (Alternate Flow of Events)	There are changes that need to be made in the documentation.
Special Requirements	Documentation must follow necessary standards and guidelines.

Table 3.2: Use Case 2

UC3: Implement Components	
Participating Actors	Component Library Developer, Component Library
Entry Condition(s)	Component design is finalized and the Component Library Developer is assigned to implement the component.
Normal Flow of Event	Component Library Developer will review the design specifications and write the code for the component. The code is tested to ensure components are compatible and adjusted as needed. Developer will commit the code to the Component Library. A pull request is created for the component.
Exit Condition(s)	Component is successfully implemented into the Component Library.
Exceptions (Alternate Flow of Events)	If the component does not meet specifications, it is reworked until it passes all tests and requirements.
Special Requirements	Component implementation must be documented and follow coding standards and guidelines

Table 3.3: Use Case 3

UC4: Generate Component Stories

Participating Actors	Component Library Developer, StorybookGPT
Entry Condition(s)	Component has been implemented and is available in the component library. The component library developer is ready to create documentation for the component.
Normal Flow of Event	The Component Library Developer prepares the input parameters for the component stories and inputs them into StorybookGPT. StorybookGPT will generate stories based off of the parameters. The developer will review and approve the generated stories, which will then be added to Storybook for the component.
Exit Condition(s)	Component stories are generated, reviewed, and integrated into Storybook.
Exceptions (Alternate Flow of Events)	If the stories are insufficient, the Developer may need to refine inputs to regenerate stories.
Special Requirements	The input to StorybookGPT must be detailed to ensure accuracy and quality of the generated stories. The component library developer must validate the generated stories to match the intended function and design.

Table 3.4: Use Case 4

UC5: Publishing Components	
Participating Actors	Component Library, npm
Entry Condition(s)	New components are implemented
Normal Flow of Events	When new components are implemented, a new pull request is created on GitHub. After reviewing the changes, the pull request will be merged and a GitHub action will be triggered. This action will determine the next package version number from the commit history and publish the built package to the npm registry.
Exit Condition(s)	The package has been successfully deployed to the npm registry with the updated version number and newly implemented components built in.
Exceptions (Alternate Flow of Events)	Component library developers can manually trigger the release process if necessary.
Special Requirements	A new package number has to be set otherwise duplicate package

	version number errors will occur in the release process.

Table 3.5: Use Case 5

UC6: Build and Deploy Storybook Docs	
Participating Actors	Component Library, Chromatic, Storybook
Entry Condition(s)	New components are implemented
Normal Flow of Events	When a pull request is merged with new components implemented, another GitHub action is triggered which builds the new Storybook documentation and deploys it to Chromatic.
Exit Condition(s)	The updated Storybook documentation is built and deployed to Chromatic.
Exceptions (Alternate Flow of Events)	Developers can build local development storybook servers for others to see components that are in progress.
Special Requirements	Chromatic has to pass regression tests with newly implemented components as well as verifying that components match designed components in figma.

Table 3.6: Use Case 6

UC7: Install Component Library	
Participating Actors	Component libraries
Entry Condition(s)	Package is published
Normal Flow of Event	User opens the system and navigates to the component library installation page.Component library is selected to install. Prompts necessary configuration or customizations to be applied and the system downloads and installs the selected component library. The system confirms the installation of component libraries.
Exit Condition(s)	Selected component libraries are installed.
Exceptions (Alternate Flow of Events)	If the selected component library is not available or cannot be installed, the system displays an error message and does not proceed with installation.
Special Requirements	System must have necessary permissions to install component libraries.

Table 3.7: Use Case 7

UC8: Test Components from Figma	
Participating Actors	Figma, test components
Entry Condition(s)	Figma project open with test components created
Normal Flow of Event	A test is initiated for the component in Figma for the selected component. Figma runs the test and verifies the component's functionality and also provides the test results. The results from Figma get reviewed.
Exit Condition(s)	Test results are provided from Figma
Exceptions (Alternate Flow of Events)	If a test component is not found in the Figma project, notification is given that the testing process is halted. If a test fails, Figma provides specific information about the failure.
Special Requirements	Figma project with test components created.

Table 3.8: Use Case 8

3.5. Rationale for your use case model

The use case model consists of several actors who are interacting with the different entities within the system. We separated developers into front end developers and component library developers to isolate how component library developers implement and document new components for front end developers to consume. While the component library developers largely contribute to the component library with the help of GPT, their work is propagated through the system after the contributions are built and released to the storybook deployment on Chromatic and npm. From there, front end developers are able to install package updates from npm and view the live storybook documentation for changes to components. When new storybook deployments are released, Chromatic can be used to test backwards compatibility with previous versions of the package while also ensuring that component stories adhere to design specifications from Figma.

3.6. Non-functional requirements

In the Non-functional requirements, we are creating two separate systems that will work hand in hand. “The System” refers to the UTD-ARGO-II Storybook repository we are improving from the first half of the UTD-ARGO Spring 2023 project. “The GPT system” refers to the ARGO Storybook GPT we are producing using ChatGPT 4.0.

Usability

A. The system shall provide an interface for creating, viewing, modifying, and deleting components.
B. The system shall provide the ability to use its components within other ARGO projects.
C. The GPT system shall provide prompts to streamline creating new stories.
D. The GPT system shall provide the ability to document each component deployed.

Maintainability

A. The system shall ensure that components will be compatible with future version upgrades.
B. The system shall house all components within the same repository.
C. The system shall be built using open source libraries.
D. The system shall automatically deploy new changes to the library using Conventional Commits.
E. The GPT system shall be built to ensure usage of proper design standards.
F. The GPT system shall utilize the latest stable version of ChatGPT.

Extensibility

A. The system shall be able to accept the creation and modification of components.
B. The system shall propagate new changes across component implementations.
C. The GPT system shall update to the latest stable updates of ChatGPT.
D. The GPT system shall utilize new features of stable updates of ChatGPT.

Scalability

A. Both systems combined shall prove the scalability of the proof of concept for ARGO.
B. The system architecture shall be designed as stateless, allowing instance modifying without affecting functionality.
C. The system should be resilient to individual component failures.

4. Architecture

4.1. Architectural style(s) used

Factory Method Pattern

The creation of components is best described using the Factory Method Architectural Pattern. Material UI is leveraged to create and define basic components in TypeScript. The Storybook GPT system serves as a factory to create different variations of the basic components.

Decorator Pattern

This pattern is used to add functionality to existing objects. Our system utilizes the existing Material UI library as the basis for components, but builds upon it to meet our needs. Storybook allows developers to modify a component's properties through its graphical user interface. The Storybook GPT stories that are generated can be considered to be adding functionality to the existing base components.

Facade Pattern

The GitHub Actions workflow and its configuration server as a facade for more complex processes such as releasing to npm and deploying to Chromatic.

4.2. Architectural model

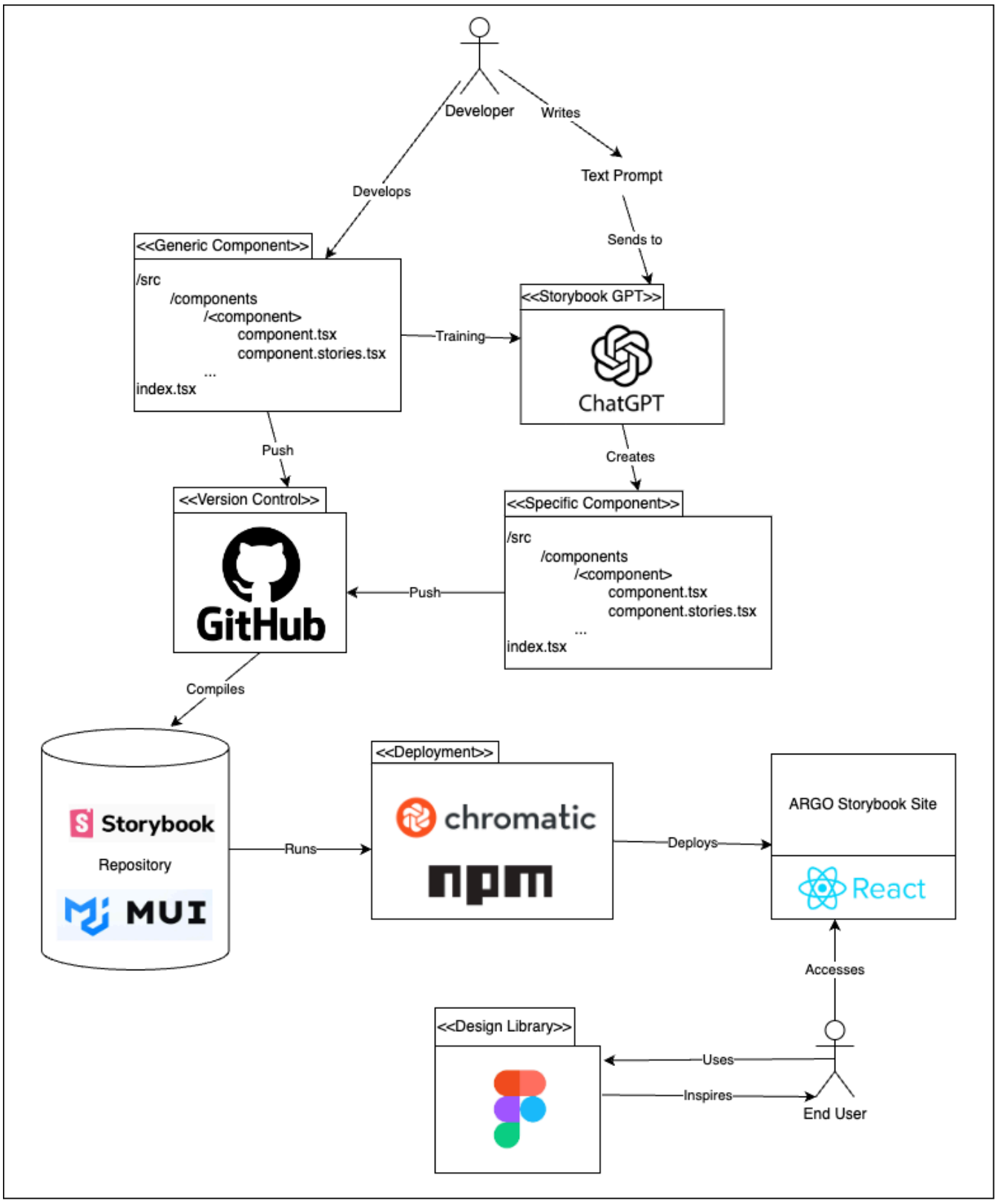


Figure 4.1. Architectural Model

4.3. Technology, software, and hardware used

This project requires a variety of technologies to be used together across the system. We will identify several perspectives where different entities will interact with the technology stack differently. The technologies grouped by system are listed below:

System	Technology
Component Library	TypeScript, React, Storybook, MUI, VS Code, GitHub, Figma
Storybook GPT	GPT-4, Storybook, React, MUI
GitHub Actions (CI/CD)	npm, semantic-release, Storybook, Chromatic, GitHub-hosted runners, Rollup

Table 4.1. Technology, Software, and Hardware Used

Component Library:

The component library requires the use of several technologies that work in tandem to export components into a package. The component's are written in TypeScript with React and we are using MUI components as a base. We use VS Code as our development environment and GitHub as our version control. Each component has a story written for it using Storybook which should adhere to design specifications defined in Figma.

Storybook GPT:

The GPT uses GPT-4 for the LLM and we have provided many existing components and stories for context which are written using React with MUI as well as stories through Storybook. The component library developers interfaces with the StorybookGPT through ChatGPT.

GitHub Actions (CI/CD):

When pull requests are merged into the master branch, a GitHub Actions workflow is triggered to release the latest updates to npm and Chromatic. These workflows are executed through the GitHub-hosted runners which are virtual machines. When building and releasing the latest component library, the new package has to pass visual tests before the package is built using Rollup and then released using semantic-release. Semantic-release updates the package version automatically and then publishes the latest components to npm. Additionally, another workflow is triggered which builds the latest Storybook documentation and deploys it to the hosted instance using Chromatic.

Communication between application server and database server:

While our project does not involve direct communication between an application server and a database server, we are leveraging tools that interact with database servers. One instance of a database server is publishing our components to npm. When releasing the package in the GitHub Actions workflow, the built component library is sent to be stored in the npm registry (which could be considered a persistent storage). Additionally, when the Storybook documentation is deployed using Chromatic, the static files are hosted on a web server to be distributed when users access the documentation. Although this is not considered a database server, there is a communication in our workflows that releases our latest changes to another system that stores the latest version of the component library as well as the documentation.

4.4. Rationale for your architectural style and model

Generic and Specific Component Subsystems

Generic and Specific components consist of a component.tsx Typescript file, and component.stories.tsx Stories Typescript file, and an optional component.css file. The Typescript file is the React component file, defining the structure, functionality, and behavior of the component. The Typescript file is also where you would define your components props and state. The Stories Typescript file is specific to Storybook. A "Story" or "Stories" represent the visual state or configuration of the component, allowing multiple behaviors and conditions of the component. The optional CSS file defines styling for the component, such as layout, typography, colors, or other design elements. Styles can also be defined in the Typescript file, but maintaining a separate Cascading Style sheet can make the component easier to maintain. These files use the Storybooks ecosystem, and additional storybook documentation can be found online.

More specifically, our system defines Generic and Specific Component Subsystems. Generic components use the MUI library to define a typical react component, such as a button, slider, and more. The MUI library contains dozens of basic react components, alongside their coding program. A developer would use tools such as MUI to develop a component. In a Specific Component Subsystem architectural flow, the developer would describe a variation or more specific component to the trained Storybook GPT. The GPT would create either a Stories Typescript as a variation of an existing component, or create a Stories Typescript and a Typescript file for a brand new component.

Storybook GPT Subsystem

The Storybook GPT Subsystem is a deliverable of the ARGO Storybook POC. During the POC, we (as one time developers) use the Generic Components as input to train the expected output of the Storybook GPT. Once completed, the purpose of this trained GPT is to accept developer text prompts and create Specific Components based on learned expectations. The Storybook GPT Subsystem will use GPT-4 for the LLM.

Version Control and Component Library

Once the developer creates, runs, and successfully tests the Generic and Specific Components, they are pushed to the Component Library using version control. Our team uses a GitHub repository for hosting our Storybook repository. The Storybook repository maintains a combination of developer created components to be quickly used and reused by the End User and closely tied to the design library. GitHub Actions workflow automates releases of the Repository system to the latest update of npm and Chromatic. An additional automated workflow is used to build the latest Storybook documentation and deploy it to the hosted Chromatic instance.

Deployment and ARGO Storybook Site

Once the automatic GitHub Action workflow is used, it deploys the ARGO Storybook site. This React site is accessed by the End User. In this instance, the End User is another layer of developer but not exclusively an individual that differs from the Developer user. While this stakeholder may be the same or a unique individual, the intention between accessing the React Site is to utilize, reuse, variant, and implement using a component stored within the site. This site is intended to be used to save the end user much time in creating components. Instead of programming, testing, inserting, designing, and deploying every component for a page, the developer accesses the site to grab a component that is already

programmed, designed to match specifications in styling, and tested. In a very specific use case, the end user accesses the site to find a component to use for their site. They may notice a component needs to be varied in order to work within their system better. In this case, the End User now becomes the developer that writes a text prompt to the GPT. The architecture is followed all the way down to deploying the site. Now, the new variant will be not only developed quicker using the Storybook GPT, but also conveniently documented for reuse in the repository.

Design Library Subsystem

The Design Library Subsystem is a combination of Documentation, Design, and Code. This system shall be used by the ARGO team as a front end tool for using new components, complying with ARGO design standards, and overall streamlining the developer experience.

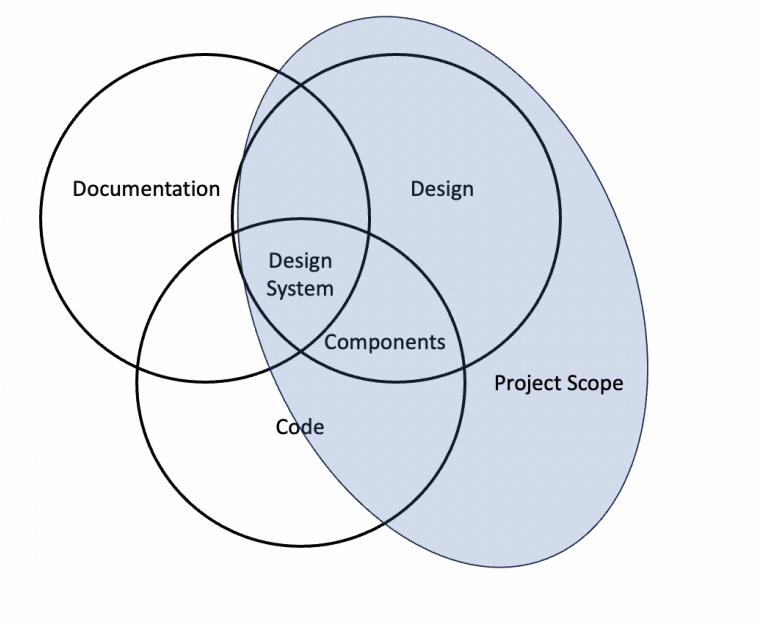


Figure 4.2. Design Library Subsystem

4.5. Traceability from requirements to architecture

Functional Requirements: based of use cases from requirements documentation

Functional Requirement	Subsystem implemented for the requirement	Subsystem relation to the requirement
The system allows designers to review, customize, and finalize components, ensuring that all designed components are easy to use and align with user expectations and behaviors.	Figma ,Storybook, Component Library, npm	Figma allows designers to gain inspiration. Whereas, Storybook allows for components to individually be showcased with a modular and reusable design.
The system enables frontend developers	Storybook	Enables frontend developers to access and

to access and review documentation created, ensuring adherence to necessary standards and guidelines, and approval upon review.		review documentation created within Storybook, ensuring adherence to necessary standards and guidelines, and approval upon review.
The system must support Component Library Developers in reviewing design specifications and writing code for components.	Component Library, MUI, ChatGPT	Subsystems assist in facilitating the reviewing of design specifications and writing code for components.
The system should allow input parameters for component stories and should generate stories accordingly, also allowing developers to review and approve the generated stories.	Storybook GPT	Storybook GPT assists in facilitating the input parameters for component stories, generating stories accordingly, and enabling developers to review and approve.
The system must enable Component Library Developers to commit code changes to the Component Library repository.	GitHub, npm	npm provides the functionality for version control and code management, allowing developers to commit code changes to the Component Library repository.
The system should allow Component Library Developers to create pull requests for component changes.	GitHub	These subsystems facilitate the creation and management of pull requests, allowing for collaboration and review of component changes before merging.
The system must successfully integrate components into the Component Library after implementation.	Component Libraries, npm, Chromatic	Component libraries handle the integration of components into the library after they have been implemented and approved, ensuring that they are available for use within the system.
The system should support iterative refinement of components until they meet specifications.	Figma, Chromatic, React	Figma, Chromatic, and React collectively support iterative refinement. Figma provides design tools for refining components, Chromatic facilitates visual testing and feedback loops, and React allows for implementation adjustments based on specifications.

Table 4.2. Traceability from Requirements to Architecture of Functional Requirements

Non-Functional Requirements

Non-functional Requirement	Subsystem implemented for the non-functional requirement	Subsystem relation to the Non-functional requirement
Usability	Figma	Figma ensures usability with an intuitive interface for designing and customizing components.
Maintainability	GitHub	GitHub allows the system to be easily maintained and updated over time.
Extensibility	GitHub, Storybook	Storybook provides extensibility by accommodating future enhancements or modifications without significant changes.
Scalability	React	Ensures the system can handle increased workload and user demand effectively.

Table 4.3. Traceability from Requirements to Architecture of Non-Functional Requirements

5. Design

5.1. GUI (Graphical User Interface) design

Storybook Controls

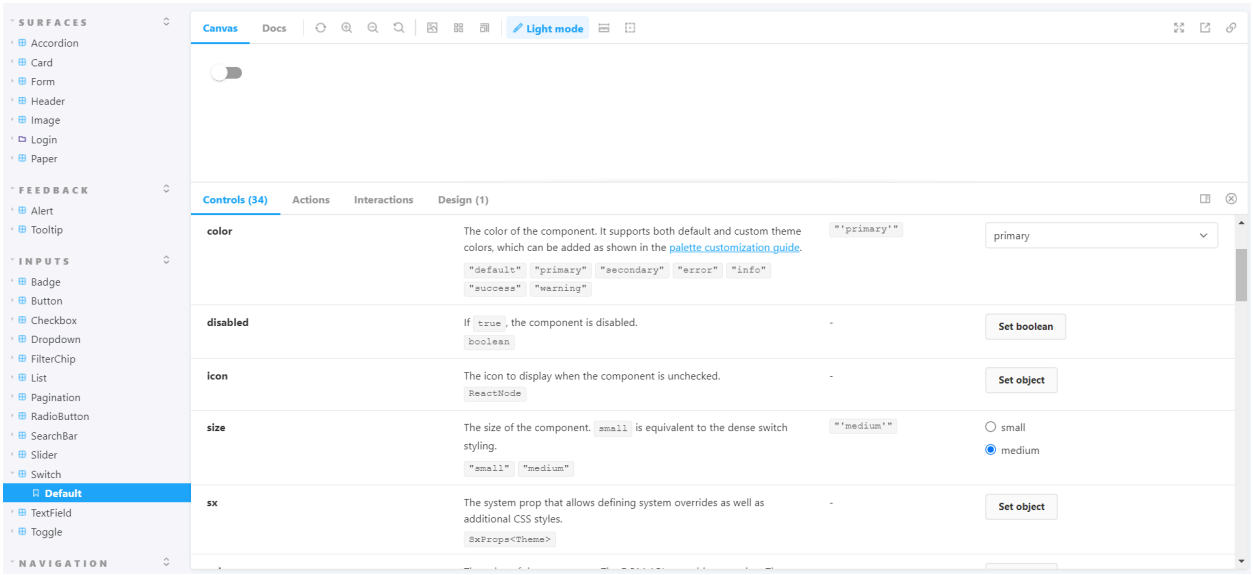


Figure 5.1: Switch Default Controls

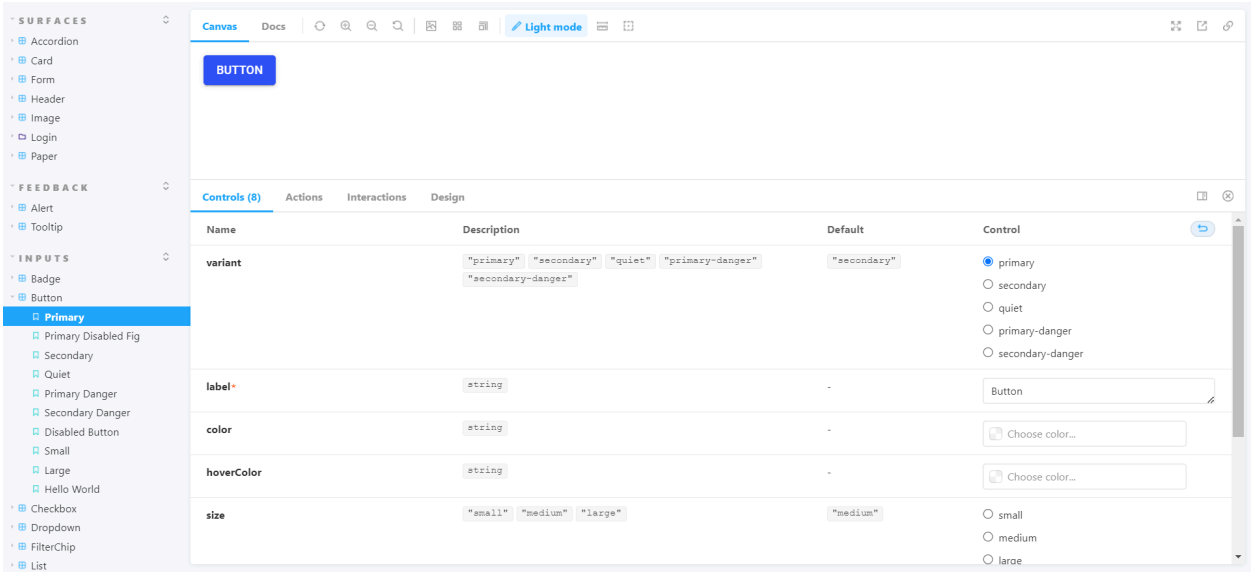


Figure 5.2: Button Primary Controls

The storybook controls are a Graphical User Interface that allows for developers and designers to modify component configurations and view how each component can react to different properties. The storybook controls were extended to include more options that closely align with the original behavior specified in the Material UI component library.

Storybook with Figma Integration

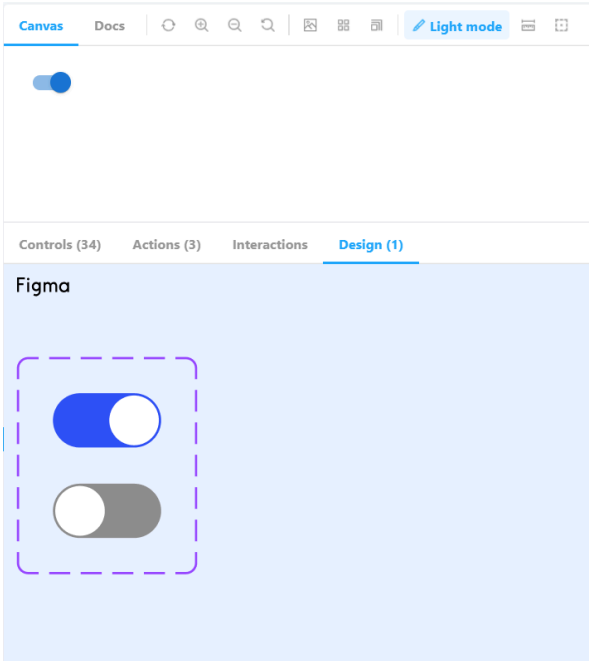


Figure 5.3: Switch Design Figma Integration

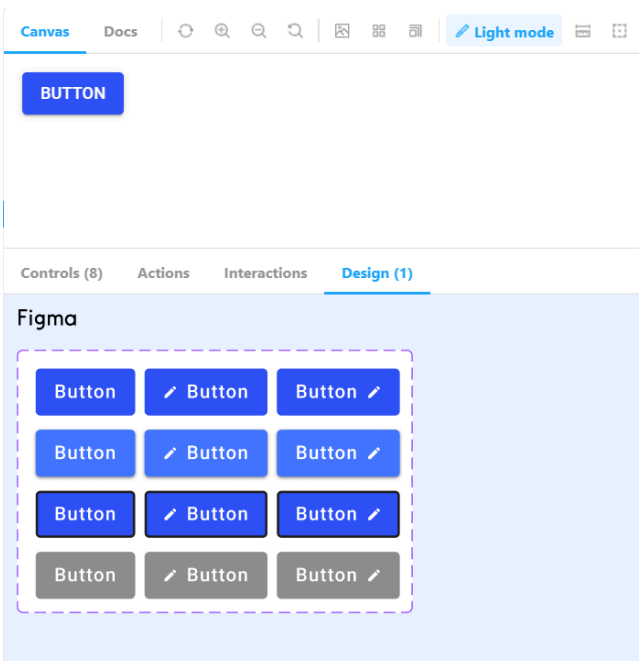


Figure 5.4: Button Design Figma Integration

The Storybook Figma integration enables viewers to cross reference how components were implemented with the original specifications defined in Figma. The components are linked through Storybook and the Figma integration allows for easy access to how each component was mocked up in Figma.

Figma button specification

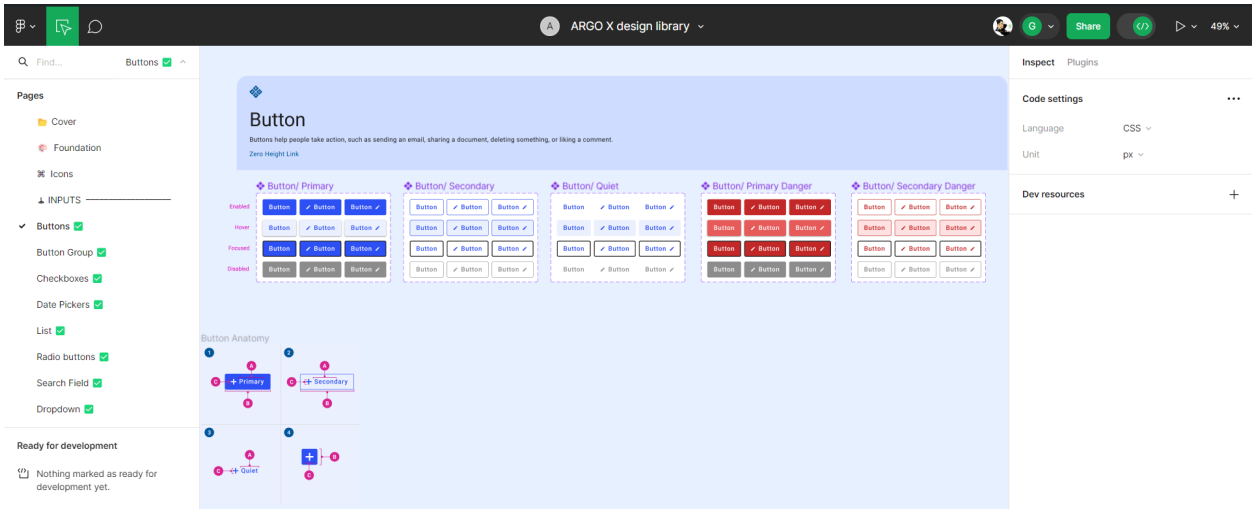


Figure 5.5: Figma Button Specification

These designs were provided by our Sponsor, Argo, and detail how each button component should behave under different configurations such as disabling the button or using a primary or secondary button.

Figma Theme Specification

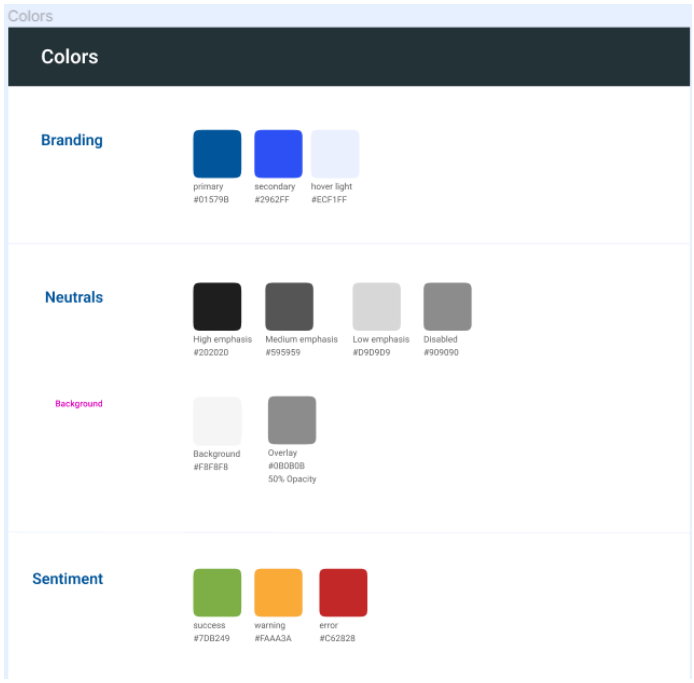


Figure 5.6: Figma Color Specifications

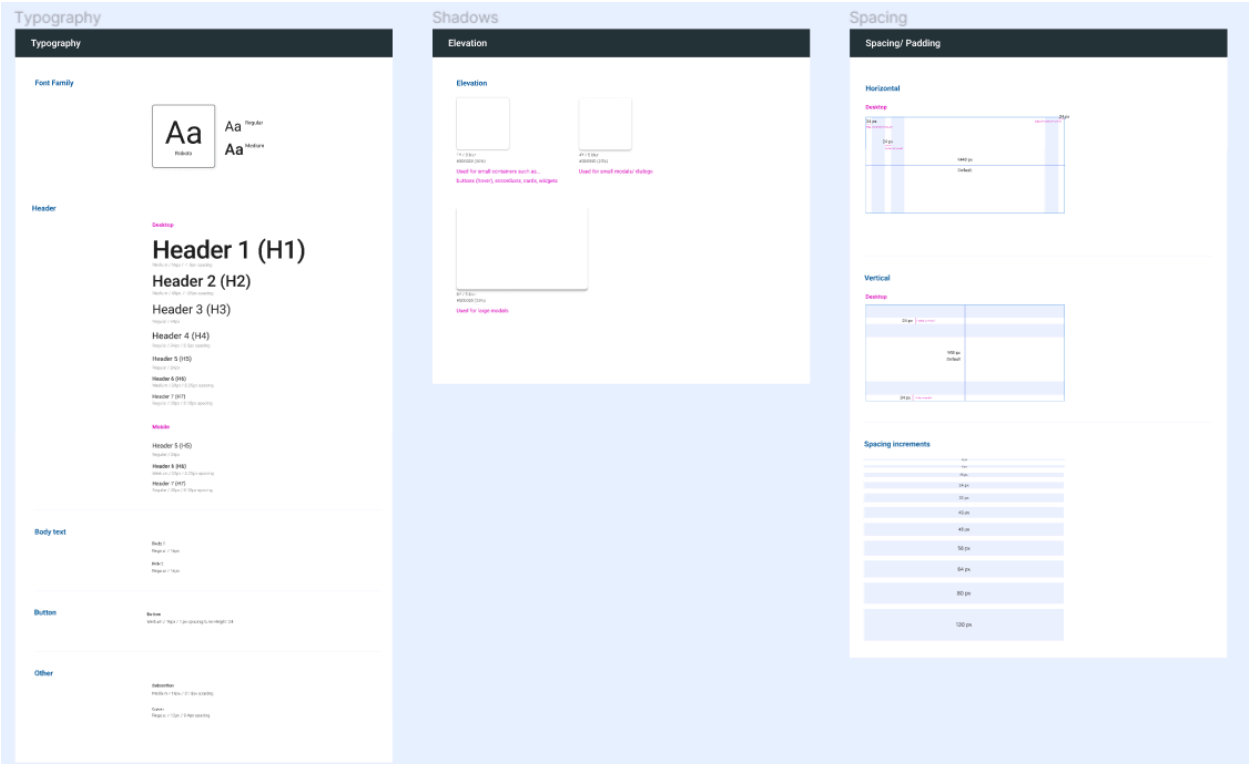


Figure 5.7: Figma Typography, Shadow, and Spacing Specification

This extensive specification defines what colors should be used in the component library theme as well as detailed theme specifications such as font, font weights, shadows, etc.

Figma Composite View

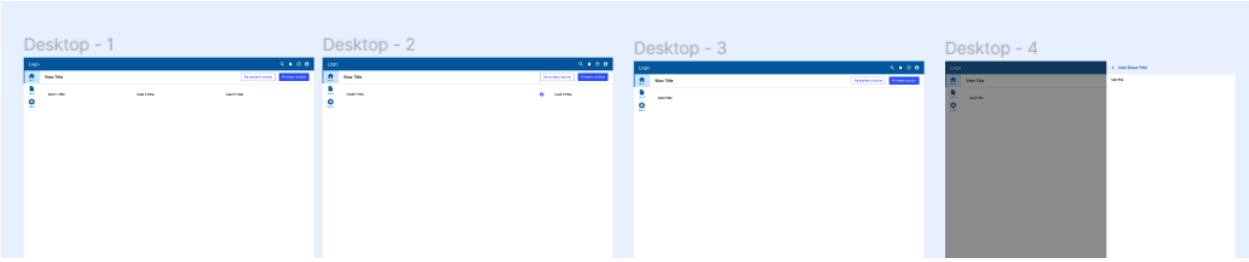


Figure 5.8: Figma Desktop View

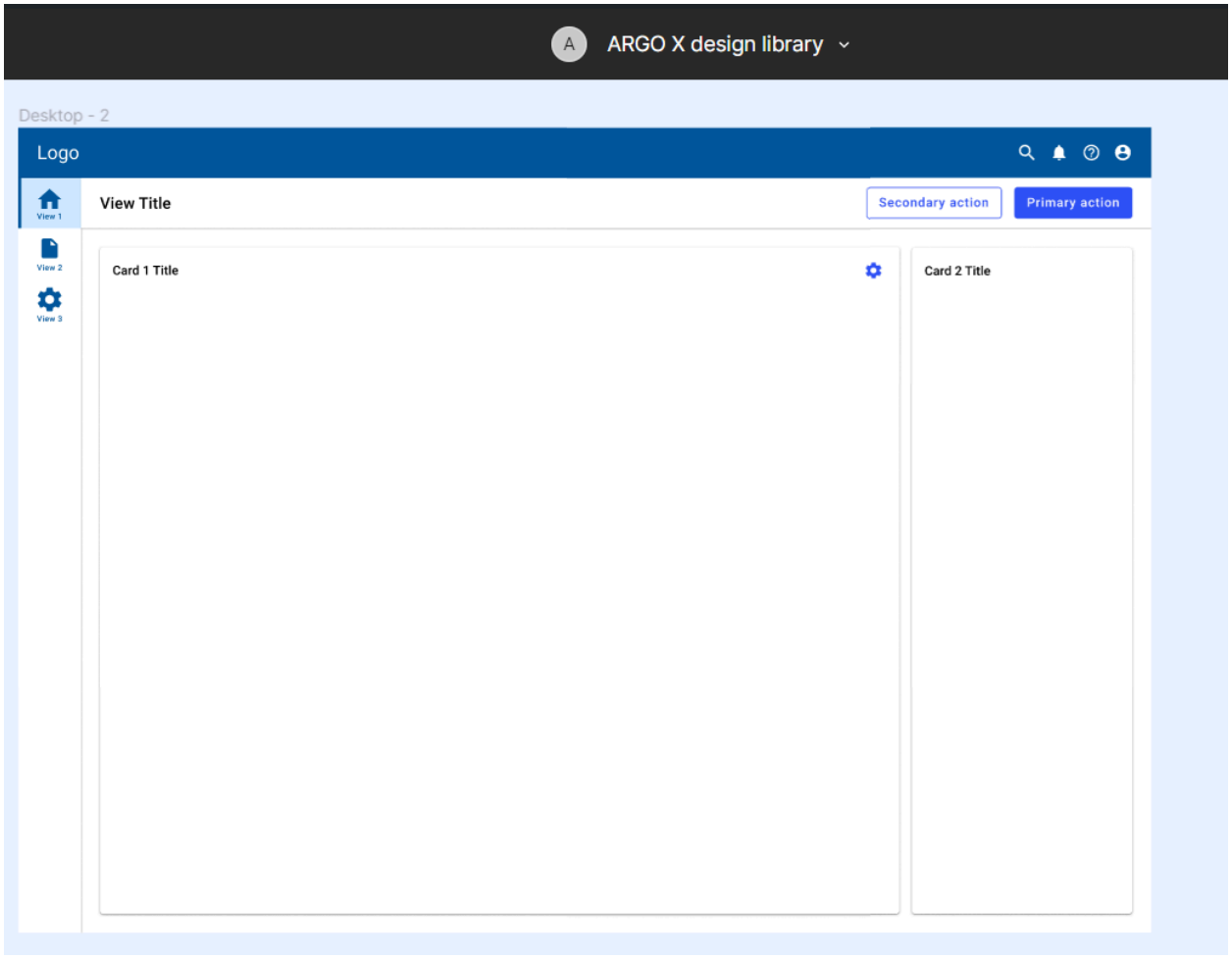


Figure 5.9: Figma Environment

The composite view is a collection of components that shows how they would interact with each other in a larger view. These designs illustrate how our components would work together in various environments and how larger user interfaces can be built from the component library.

5.2. Static model – class diagrams

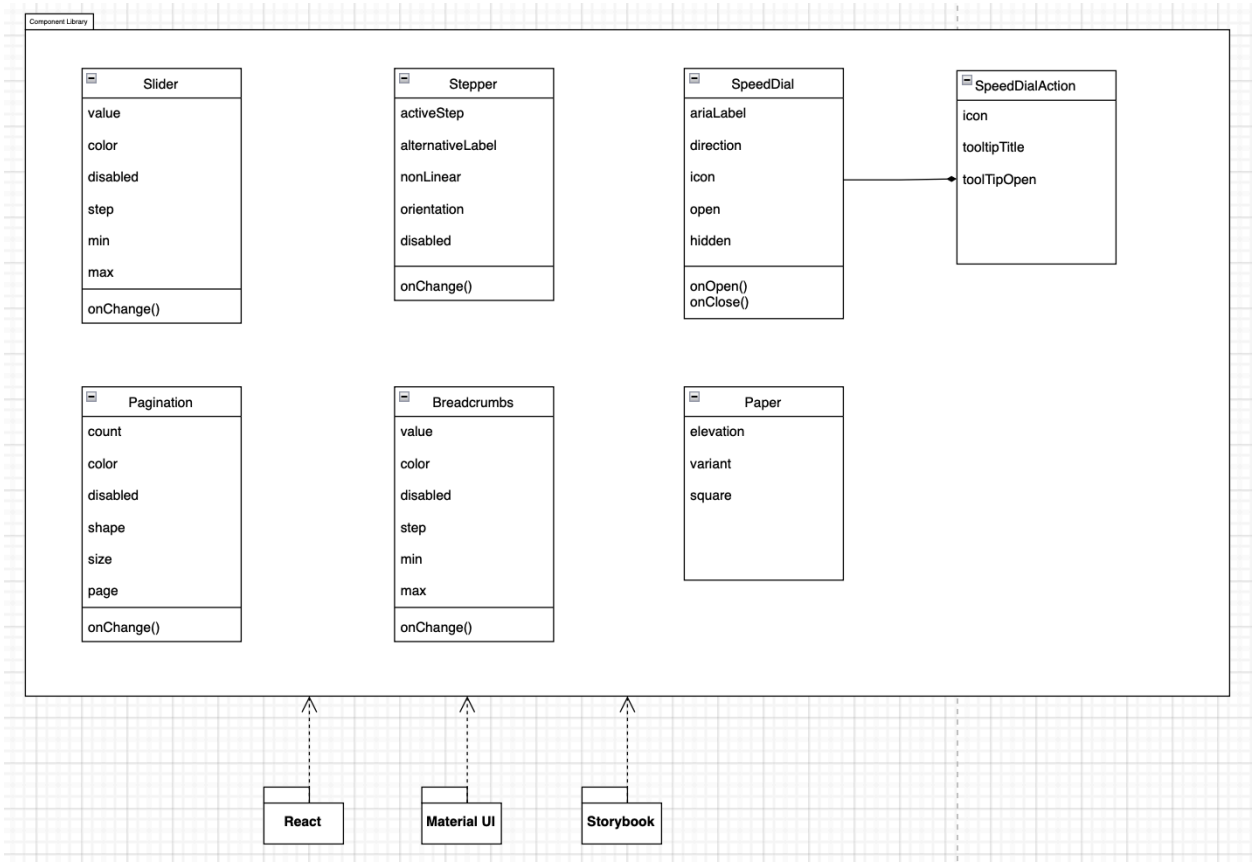


Figure 5.10: Static Model of the System

5.3. Dynamic model – sequence diagrams

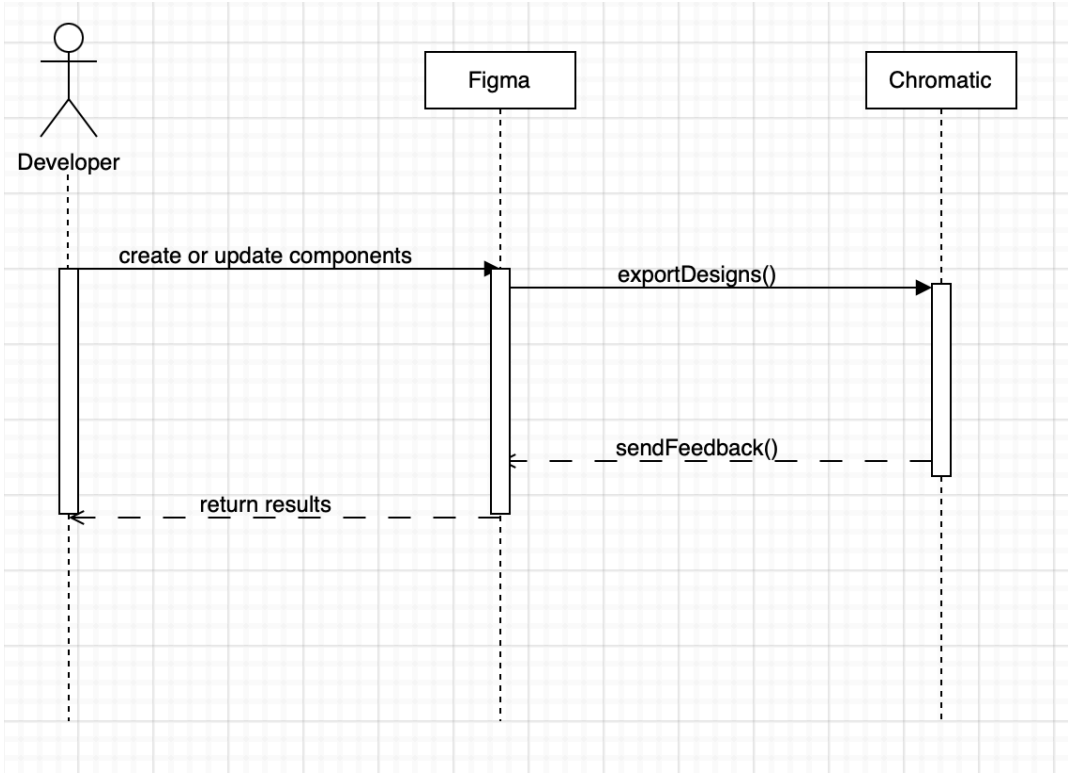


Figure 5.11: Dynamic Model with Figma, Chromatic

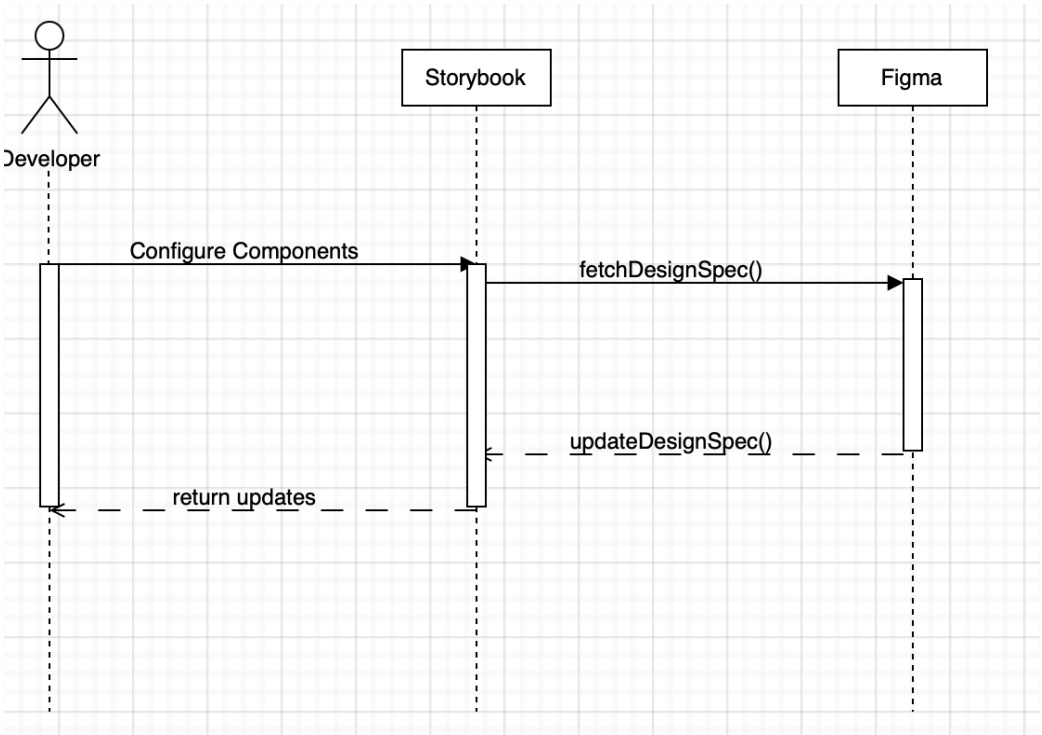


Figure 5.12: Dynamic Model with Storybook, Figma

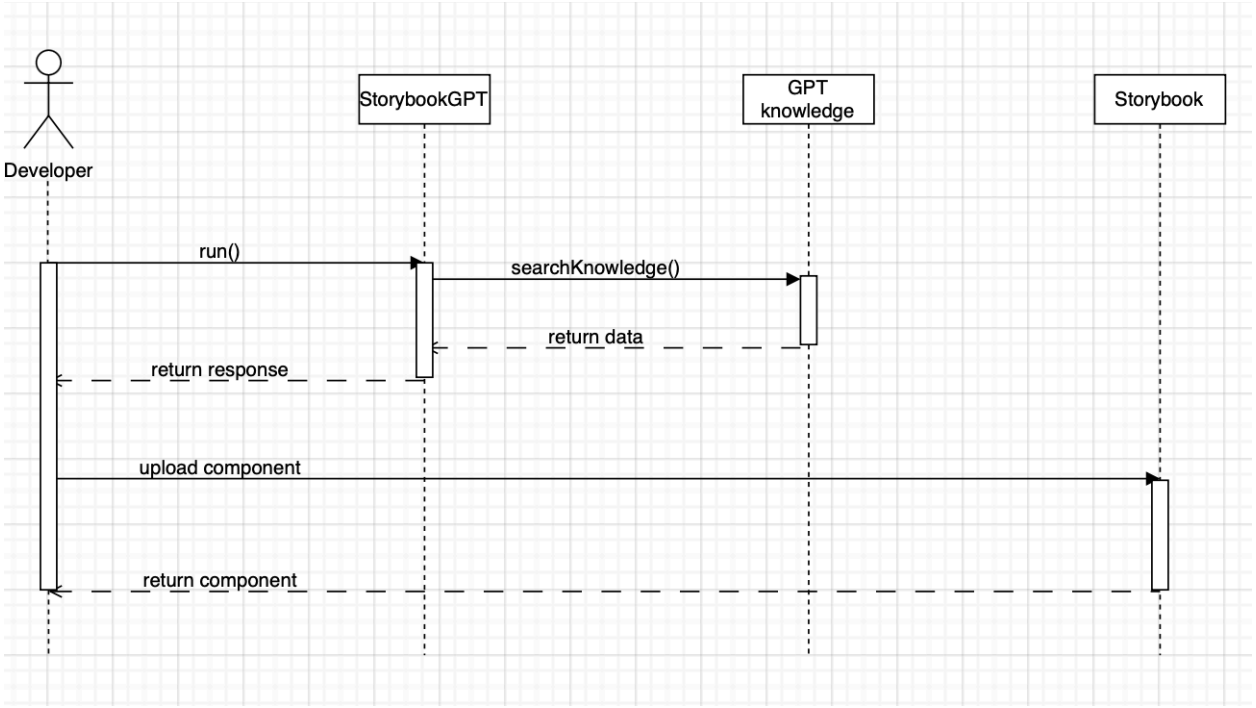


Figure 5.13: Dynamic Model with StorybookGPT, Knowledge, and Storybook

5.4. Rationale for your detailed design model

Graphic User Interface (GUI):

The component library requires the use of several systems, each with their own GUI's. In figures 1-8, we have outlined how the design of the GUI has impacted the development of components and other features through Storybook. Each of the GUI design's have been driven by the requirements of the features, enabling seamless component development from design to package release.

Static Model:

UML Class Diagrams provide a high level overview of the components that were built in this phase of the project. The UML Class diagram displays the component library and the objects that it compromises. Each component in the class diagram contains a set of attributes that are associated with the component, as well as a set of methods that apply to that component. UML class diagrams also provide a visual representation of our structure, helping to understand how the components are structured and relate to one another. For example, with a component like Button or Checkbox, the UML class diagram can show the properties and their relationship with other parts of the system such as Button being used within Form.

Dynamic Model:

A sequence diagram was used to demonstrate the dynamic model because it clarifies actions and processes that the class diagram might not have included. Using a sequence diagram can provide a comprehensive perspective among entities, giving a deeper understanding of the system. For our diagram, it illustrates the interactions among a developer, Storybook, Figma, Chromatic, and GPT-4, enhancing the understanding of the system's dynamics.

5.5. Traceability from requirements to detailed design model

Functional Requirements: based of use cases from requirements documentation

Functional Requirement	Design Element	Description
The system allows designers to review, customize, and finalize components, ensuring that all designed components are easy to use and align with user expectations and behaviors.	Figma, Chromatic, Storybook	Figma allows the developer to update the components design and export them to Chromatic and get feedback to the designer to make any adjustments. Storybook allows for the configuration of the components. fig. 11,12
The system enables frontend developers to access and review documentation created, ensuring adherence to necessary standards and guidelines, and approval upon review.	Storybook	Storybook allows frontend developers to access and review documentation created within Storybook. fig. 12
The system must support Component Library Developers in reviewing design specifications and writing code for components.	Storybook, Figma	Storybook allows the enhancement of by streamlining the configuration of components. Figma works to review the design of the components. fig.12
The system should allow input parameters for component stories and should generate stories accordingly, also allowing developers to review and approve the generated stories.	Storybook GPT	Storybook GPT allows developers to input parameters to customize components and generate stories. fig.13
The system must enable Component Library Developers to commit code changes to the Component Library repository.	GitHub, npm	The design and utilization of GitHub guarantees reliability and adhere to best practices in version control.
The system should allow Component Library Developers to create pull requests for component changes.	GitHub	The design of Github incorporates functionalities for initiating, reviewing, and merging pull requests, allowing for collaboration and code review across all components.

The system must successfully integrate components into the Component Library after implementation.	Component Libraries, npm, Chromatic	The setup employs npm and Chromatic to integrate and deploy components seamlessly into the Component Library.
The system should support iterative refinement of components until they meet specifications.	Figma, Chromatic, React	The collaborative design setup enables iterative improvements. Figma offers design utilities for enhancing components, Chromatic aids in visual testing and feedback cycles, while React permits adjustments in implementation according to specifications..

Table 5.1: Traceability from Requirements to Detailed Design of Functional Requirements

Non-Functional Requirements

Non-functional Requirement	Design Element	Description
Usability	Figma	Figma ensures usability by allowing developers to create mockups and visualize components. fig.3,7
Maintainability	GitHub	The design of GitHub allows the system to be easily maintained and allows for updates to easily be made.
Extensibility	GitHub, Storybook	Storybook allows for extensibility by allowing modifications and enhancement without significant changes.GitHub allows the system to be easily updated. fig.12
Scalability	React	The design of React's architecture allows it to handle increasing complexity for growing applications.

Table 5.2: Traceability from Requirements to Detailed Design of Non-Functional Requirements

6. Test Plan

6.1. Requirements/specifications-based system level test cases

UI Test Cases for Newly Introduced Components

ID	Component	Test Cases
CD1	Text Fields	a. Label displays label string. b. Context works as a text box by allowing input on click. c. Helper text displays helper string. d. Calendar icon displays when enabled.
CD2	Drop Down	a. Label moves on click. b. Drop down appears on click. c. Drop down has correct and entire contents. d. Color variants are accurate .
CD3	Paper	a. Paper displays components. b. Elevation Toggle works as values are changed. c. Sample paper displays according to controls.

CD4	Slider	a. Slider moves and actions are recorded. b. Disabled toggles correctly. c. Slider label displays a label string.
CD5	Pagination	a. Page changes on Click b. Changing boundary count reflects the correct boundary for variant page numbers. c. Changing count control reflects accurate page numbers. d. Default Page renders accurately on page refresh.
CD6	Pagination V2	a. Page changes on click b. Star disables when no longer applicable. c. Page numbers are sticky when clicked. d. Changing boundary count reflects the correct boundary for variant page numbers.
CD7	ARGO Button	a. Button works on click. b. Variant buttons work on click. c. Disable button does not work on click.

Table 6.1. UI Test Cases for Newly Introduced Components

System Test Cases

ID	Test Case
TC1	Figma designs appear under a component's design tab in Storybook
TC2	For each view change, ensure documentation is updated.
TC3	Newly created components appear in Storybook.
TC4	Prompt Storybook GPT for creation of new story based off parameters generates a new component and is reviewed to be successful by developer.
TC5	Make a new commit and confirm that version number is correct and the package is built on npm ARGO UX Master Component Library II.
TC6.1	Project is built and deployed to Chromatic after a pull request is merged.
TC6.2	Chromatic Regression Testing is approved by developer and appears in Library.
TC7	Component Library is successfully installed.
TC8	Figma design displayed in design tab matches component in Storybook

Table 6.2. System Test Cases

6.2. Techniques used for test generation

Three types of test generation techniques were employed: GUI Testing, Regression Testing, and Usability testing.

GUI testing is used on icons, buttons, menus, textboxes, and all other UI components to ensure they function correctly. This is a form of black-box testing.While mostly manual work in the beginning to test all implemented components thoroughly, GUI testing on a day to day basis will be done only per component. At deployment, all components are tested once with GUI manual testing. As components are added like in TC3, UC3, only one component is manually tested and is done in less than 10 minutes. Our GUI testing is used in TC1, TC3, TC4, and TC8. We will use a successful/unsuccessful measurement on the GUI testing and our goal is 95% successful.

Regression testing is the main benefit of our Chromatic integration into our system. This is a form of black box testing. The regression testing for Chromatic allows quick and easy comparison of changes that have been made to every build sent through the main branch into the Chromatic system. As a build is finished, a developer will use Chromatics built in regression changes to review the impact of the component differences and approve/deny as needed. Regression testing through Chromatic is used in TC2, TC4, TC6.1, and TC6.2. Regression testing is measured by an approval/denial system built into Chromatic.

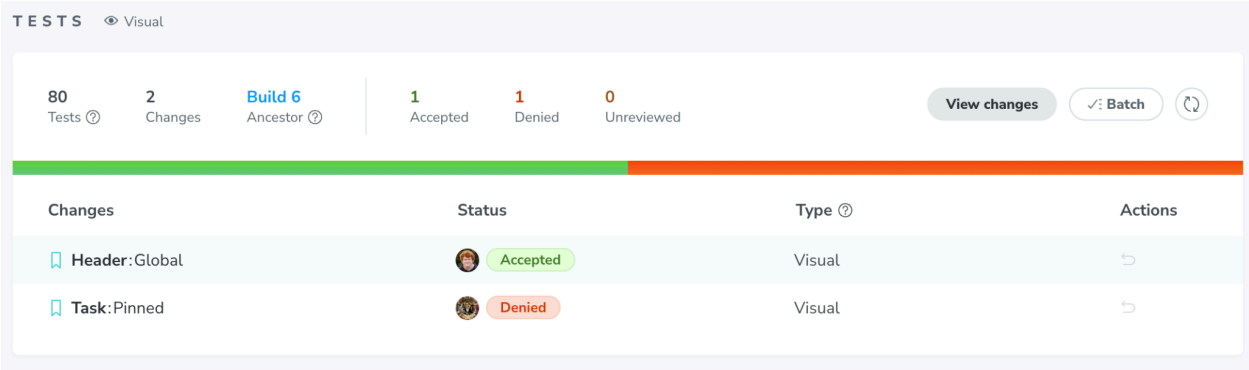


Figure 6.1. Chromatic Regression Testing.

Usability and acceptance testing were conducted by observing real users, like our developers, as they attempt to complete tasks on it. This version of white box testing ensures continuous improvement of code and development practices within the team, increasing opportunities for collaborative code. The test cases that use acceptance testing are TC5 and TC7.

6.3. Assessment of the goodness of your test suite

As this project was a proof of concept rather than a fully developed product, rigorous, thorough, and well-documented testing utilizing standard software testing techniques was not a focus. The goodness of our test suite was determined by the successful rate of UI test cases and the approval rate of our regression testing through Chromatic.

The metric used to evaluate the UI test cases was the success rate of the test cases. UI test cases were considered to have passed if the properties of the component in the Storybook Library behaved as described in the specific test case. This metric provides insights into the functionality and correctness of our UI components.

The metric used to evaluate regression testing was the approval rate of builds through Chromatic. This metric aids in assessing the impact of code modifications on the stability of our component library.

6.4. Traceability of test cases to use cases

Test Case	System Use Case
TC1	UC1: Designs Components
TC2	UC2: Views Documentation
TC3	UC3: Implement Components
TC4	UC4: Generate COmponent Stories
TC5	UC5: Publishing Components
TC6	UC6: Build and Deploy Storybook Documentation
TC7	UC7: Install Component Library
TC8	UC8: Test Components from Figma

Table 6.3. Traceability Of Test Cases To Use Cases

Each test case was created to meet the exit case specified in the corresponding Use Case. For example, Text Case 1 was built off the Use Case 1 exit condition “The components have been designed and finalized.”

7. Evidence the Document Has Been Placed under Configuration Management

Version In	Version Out	Changes	Reviewed By	Notion Task Numbers
n/a	0.0	Document Creation based on Template	Lillie McMaster	UG3-101
0.0	1.0	All missing fields.	All Group Members	UG3-102

8. Engineering Standards and Multiple Constraints

- Students should work with their project sponsor(s) to identify all the standards and constraints that should be applied for preparing this document
- Duplicate entries should be removed

- IEEE Std 1058-1998: Software Project Management Plans [pdf]
- PMBOK® Guide: Project Management Body of Knowledge [pdf]
- IEEE Std 12207: Software Life Cycle Processes [pdf]
- IEEE Std 15939: Measurement Process [pdf]
- ISO/IEC/IEEE Std 29148-2018: Systems and Software Engineering
    - Life Cycle Processes
- IEEE Std 830-1998: Software Requirements [pdf]
- IEEE Std 29148: Requirements Engineering [pdf]
- IEEE Std 1471-2000: Software Architecture [pdf]
- ISO/IEC/IEEE Std 42030:2019: Software, Systems and Enterprise
    - Architecture Evaluation Framework [pdf]
- IEEE Std 1016-1998-(Revision-2009): Software Design [pdf]
- IEEE Std 829-1983: Software Testing [pdf]
- ISO/IEC/IEEE Std 29119-1-(Revision-2022): Part 1 - Software Testing General Concepts [pdf]
- ISO/IEC/IEEE Std 29119-2-(Revision-2021): Part 2 - Test Process [pdf]
- ISO/IEC/IEEE Std 29119-3-(Revision-2021): Part 3 - Test Documentation [pdf]
- ISO/IEC/IEEE Std 29119-4-(Revision-2021): Part 4 - Test Techniques [pdf]

9. Additional References

- Larson, E. and Gray, C., 2014. *Project Management: The Managerial Process.* McGraw Hill
- Humphrey, W.S. and Thomas, W.R., 2010. *Reflections on Management: How to Manage Your Software Projects, Your Teams, Your Boss, and Yourself.* Pearson Education
- Northeastern University: 12 Steps to Develop a Project Management Plan by Shayna Joubert https://graduate.northeastern.edu/resources/developing-project-management-plan/
- Lamsweerde, A.V., 2009. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley
- Non-Functional Requirements in Software Engineering, L. Chung, B. Nixon, E. Yu and J. Mylopoulos, Kluwer Academic Publishing, 2000
- Scenarios, Stories, Use Cases Through the Systems Development Life-Cycle, I. Alexander and N. Maiden (eds.), John Wiley & Sons, 2004.
- Malan R, Bredemeyer D. Functional requirements and use cases. Bredemeyer Consulting. 2001 Mar.
- Lattanze, A.J., 2008. *Architecting Software Intensive Systems*: A Practitioner's Guide. CRC Press
- Bass, L., Clements, P. and Kazman, R., 2003. *Software Architecture in Practice*. Addison-Wesley
- Storybook Image from Medium. *Configuring Storybook: 6 Tips You Can't Miss*. Accessed March 11th 2024. https://medium.com/strands-tech-corner/storybook-configuration-in-react-project-ec59869f3e7d
- React Image from *GitHub React-Ui Topics*. Accessed March 11th 2024. https://github.com/topics/react-ui
- GitHub Image from Project Pythia. *What is GitHub?* Accessed March 11th 2024. https://foundations.projectpythia.org/foundations/github/what-is-github.html
- MUI Image from MUI. Accessed March 11th 2024. https://mui.com/
- Figma Image from Figma. Accessed March 11th 2024. https://www.facebook.com/figmadesign/
- ChatGPT from JacobsMedia. *ChatGPT Logo Square*. Accessed March 11th 2024. https://jacobsmedia.com/a-radio-conversation-with-chatgpt-part-1-sales/chatgpt-logo-square/
- Figma Learn: Storybook and Figma. Accessed April 1st 2024. https://help.figma.com/hc/en-us/articles/360045003494-Storybook-and-Figma
- Storybook Documentation: Building pages with Storybook. Accessed April 8th 2024. https://storybook.js.org/docs/writing-stories/build-pages-with-storybook
- Chromatic Docs: Document. Accessed April 8th 2024. https://www.chromatic.com/docs/storybook/document/
- A. Harbert, W. Lively and S. Sheppard, A graphical specification system for user-interface design, in IEEE Software, vol. 7, no. 4, pp. 12-20, July 1990. Accessed April 8th 2024.
- Larman, C., 2012. Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development. Pearson Education
- Hyman, B., 1998. Fundamentals of Engineering Design. New Jersey: Prentice Hall
- Simon, H.A., 2014. A Student's Introduction to Engineering Design: Pergamon Unified Engineering Series (Vol. 21). Elsevier
- GfG. "Software Testing Techniques." GeeksforGeeks, GeeksforGeeks, 6 Dec. 2023, www.geeksforgeeks.org/software-testing-techniques/.
- "Ui Testing: What It Is and How to Do It." Hotjar, www.hotjar.com/ui-design/testing/. Accessed 13 Apr. 2024.
- "Usability Testing: What It Is, Benefits, and What It Isn't." Hotjar, www.hotjar.com/usability-testing/. Accessed 13 Apr. 2024.
- Jorgensen, P.C., 2013. Software Testing: A Craftsman's Approach. Auerbach Publications
- Mathur, A.P., 2013. Foundations of Software Testing, 2/e. Pearson Education

Acknowledgment

The group 3 from UTD wants to extend their special thanks to each contributing member of the project, including:

- Ponchai Reainthong
- Kevin Roa
- Raisa Gonzalez
- Micheal Hube
- Eric Wong
- Zizhao Chen
- Hih-Wei Hsu

Without their support and guidance, the results of this project would not have been possible. Thank you!