FIGFLOW TW – A TOKEN-BASED DESIGN-TO-CODE FIGMA PLUGIN

Matthew Dent

N00220082

Supervisor: John Montayne

Second Reader: Catherine Noonan

Year 4 2025/2026

DL836 – BSc(Hons) Creative Computing

1. Abstract

This project displays the design and development paths taken to create a Figma plugin which is directed towards increasing the efficiency in the workflow of the design to development pipeline. The plugin aims to produce a user-friendly, efficient user interface which incorporates a token-based Tailwind CSS design system. These aims work cohesively to generate automated code outputs in language-style pairings including HTML + CSS, HTML + Tailwind CSS, and React + Tailwind CSS. The plugin makes use of the Figma API to extract node properties and convert them into Tailwind token groups, including colours, typography, spacing and layout properties.

A primary feature of the plugin is the token-based design system as it provides consistent design and code outputs, as the plugin provides a live preview of the generated code and real-time editing of the Figma design. Additionally, the plugin incorporates search and filter functionality for ease of use and efficient navigation of the user interface.

The project uses an iterative development process, as user testing is performed to gradually implement accuracy, user-accessibility and navigation quality. Through testing the plugin appears to improve efficiency when translating designs to a production-ready code output, which signifies the plugin is effective for usage in the design to development pipeline.

Research performed highlights the importance of automating the design to development process as it reduces the manual translation time and ensures accurate style and spacing translation is preserved. Future improvements for the project include expanding the available frameworks for code outputs, improving on the visual aesthetics of the user interface, and the integration of AI for increasing efficiency in the design process on Figma.

1.1. Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of a Bachelor Degree, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Recoverable Signature

X 

Student Signature

Signed by: 1f85baf3-ea0e-4da7-8f55-bb80d62f9480

1.2. Acknowledgements

I would like to express my gratitude to my supervisor, John Montayne, for his guidance, feedback and time spent throughout the planning, development and evaluation of this project. His experience and knowledge assisted in developing the technical and research aspects further. Especially in the early stages of the project as the initial idea and scope was narrow and unrefined, but his support guided the development which progressed immensely.

Additionally, I am thankful to all those who participated in the multiple testing stages of this project. With their feedback and perspectives, the project was refined and iterated to meet the aims and objective set.

Finally, I would like to thank my family and friends for their support and encouragement throughout the project's timeframe.

2. Contents

2.1. Figures list

3. Introduction and Project Context

3.1. Project Overview

This project focuses on developing a Figma plugin which allows for users to design and develop high-quality Figma designs and translate them accurately into generated code components. The plugin aims to increase the workflow efficiency between designers and developers through the use of automated code generation and Tailwind tokenization.

The functionality of the plugin consists of a token-based design system which incorporates Tailwind CSS into an interactive design panel. This design panel allows users to create Figma designs using tailwind colour, spacing, text and layout tokens. Tokens can be applied directly to a selected Figma frame and visually see the updated design in real time. Along with tailwind tokens, the design system contains an automated code generation utility. Multiple language-style pairings are available for generation, including HTML + CSS, HTML + Tailwind CSS, React + Tailwind CSS. The generated code is displayed as a preview whilst allowing the user to download or copy the code output. These systems combined provide an efficient workflow when designing to developing, as well as consistent styled code designs, which in turn provides improved usability of the design interface (Nielsen, 2020).

This project is directly impacting the problem that persists in the design to development workflow. Inconsistencies continuously occur in the hand-off stages development, as designers create high-fidelity designs and components that require as much attention when translating into developed code. This issue is displayed in Figure 1, which shows the original design to development workflow, including the requirement to manually correct inconsistencies. However, these inconsistencies interfere with the styling, typography and general spacing properties of the designs. This requires developers to manually translate designs which is time-consuming and inefficient for the workflow.

The problem affects modern workflows as iterative design is prominent in developing consistent and scalable designs. Through reducing these workflow deficiencies, the pipeline between designers and developers is able to produce high fidelity designs whilst maintaining the quality of the code generated and improving the overall consistency of the workflow (Piccolo et al., 2021). The contrast is demonstrated in Figure 2, as the workflow removes complications and decreases the manual involvement during the hand-off process.

Figure 1 - Original Workflow

Figure 2 - Plugin Workflow

3.2. Project Aim

The aim of the project is to design and develop a Figma plugin that allows users to create consistent high-fidelity designs and efficiently translate these designs into production-ready generated code outputs. This is pursued through the implementation of Tailwind design tokens, extraction of the Figma nodes to allow editing of node properties, and generation of clean code outputs in multiple languages.

Through research into competitors and available plugins a reoccurring issue is the lack of quality user interfaces, user accessibility and ease of use available in current plugins (Nielsen, 2020). There are several which force the use of external applications to perform basic interactions such as account creation, code generation and design previews. These issues contribute to the bottleneck that persists in the design to development pipeline, reducing the usability of the interface.

3.3. Project Objectives

The objectives of this project are to enhance the effectiveness of the technical and logical workflow of the design to development pipeline. Initially, the project focuses on understanding where issues are apparent, surveys are conducted on users in frontend development roles to understand the limitations around their workflow and development process.

A primary objective of the project is to create an optimised user interface which is displayed as a design panel in the Figma application. This interface must incorporate functionality that enhances the design to development workflow, including clear navigation, efficient design usage, and maintainable code outputs. Along with these necessities the plugin prioritises creating a design system which is token-based with

Tailwind CSS and consists of colour, typography, spacing and layout token groups (Design Tokens Format Module 2025.10, 2026).

Testing of the application is done through user testing which provides insight into flaws of the system and areas that require further development. These tests will guide the development of the project towards successfully achieve the project aims.

3.4. Success Criteria

The success of the project is achieved through the completion of numerous key features of the project's aims, these include areas of development, design and overall quality. The criteria is a combination of providing consistent and usable code outputs, and an efficient, user-friendly design system. This set of criteria yields the key features of the plugin which incorporates both functionality and user requirements.

The criteria set for the usability of the project focuses primarily on enabling the users to be capable of designing in the Figma application with the use of the plugin. This includes selecting design frames, applying token groups, and generating automated code. The user interface must provide clear navigation for users to effectively use the plugin's features without difficulties (Nielsen, 2020).

Along with the usability criteria, a set of quality criteria are implemented for ensuring the project is developed for efficiency and accuracy of the design production. This includes creating code generated outputs through precise translation of high-fidelity designs, whilst maintaining a token-based design system that provides accurate property stylings for designing and producing a seamless transition between the design to development process.

Overall, the project criteria are implied a success if the plugin presents reliable generated code outputs, an integrated user interface, and advancements on the design to development workflow to increase efficiency and reduce errors in the hand-off of the design.

3.5. Project Scope

The scope of this project is the definition of the functionality, effectiveness and accuracy of the plugin's features and outputs. The project focuses on enhancing the design to development workflow to create an efficient pipeline during the hand-off process. This includes converting high-fidelity Figma designs into production-ready generated code outputs.

The scope includes the implementation of code generation of a selected Figma frame, which allows users to create an automated code output of a desired frame or component. These outputs can be converted into multiple language-style pairing, including HTML + CSS, HTML + Tailwind CSS, and React + Tailwind CSS. In addition to generating code outputs, the scope incorporates supporting a token-based design system using the Tailwind token groups. These groups consist of colours, typography, spacing and layout properties, all of which are key to creating a high-fidelity design in the design stage.

Features comprised within the project scope are search and filter functionality which reflect the effectiveness and ease of use of the navigation in the plugin. The design of the user interface is prominent as the effectiveness of the workflow is dependent on the accuracy of the plugin interface and navigation.

The scope does not include certain features as the timeframe of the project ensures limited but realistic aims to be set upon completion. These features include cloud storage instead of client storage for custom tokens, AI integration for increasing the navigation and usage of the plugin, and support of additional frameworks and languages.

Through definition of the scope features, the project maintains an expected outcome which remains realistic of the timeframe and skillset available.

4. Research and Background

4.1. Outline

This report is designed to display the project's iterative cycle, the decisions and errors that were met throughout the development process. The continuous development of the design features, implementation of tailwind tokens, and generation of code outputs are explored in this report. The chapters provide insight into the outcomes previously made and the implementation path taken to incorporate them.

The introduction chapter describes the initial outline of the project, the aims and objectives set to complete, the success criteria to determine whether the aims and objectives were achieved, and the scope which decides the realistic outline of the possibilities the project can incorporate.

4.2. Introduction

Component-based development (CBD) has become a prominent approach to the software development process, specifically in the user interface development. Instead of constructing unified systems, CBD encourages the use of modular and reusable components that includes both structure and functionality into the development process. This improvement provides consistency across various systems (Sparling, 2000).

Design tools such as Figma and Adobe XD have advanced past a stagnant design page and have evolved into a collaborative system where users can create components, design systems and code generators. Design tools allow designers to create reusable components and apply styling through design tokens, leading to a consistent design layout throughout the structure and development process (Design Tokens Format Module 2025.10, 2026).

Even with these advancements, there remains a gap between the design and development pipeline. Designers create high-fidelity user interfaces through iterated prototypes (Coyette et al., 2007), however developers are left to manually interpret and translate these interfaces into secure code.

4.3. Literature Review

4.3.1. Foundations of Component-Based Development

Component-based development is the basis of creating systems through groups of reusable elements. These components combine the functionality and interface features into one to ensure consistent development throughout the system. The use of reusable components reduces duplication of code and improves system maintainability, providing a more manageable system overall (Sparling, 2000). The structure of a component system is displayed in Figure 3, which exhibits the architecture of an application built from multiple components.

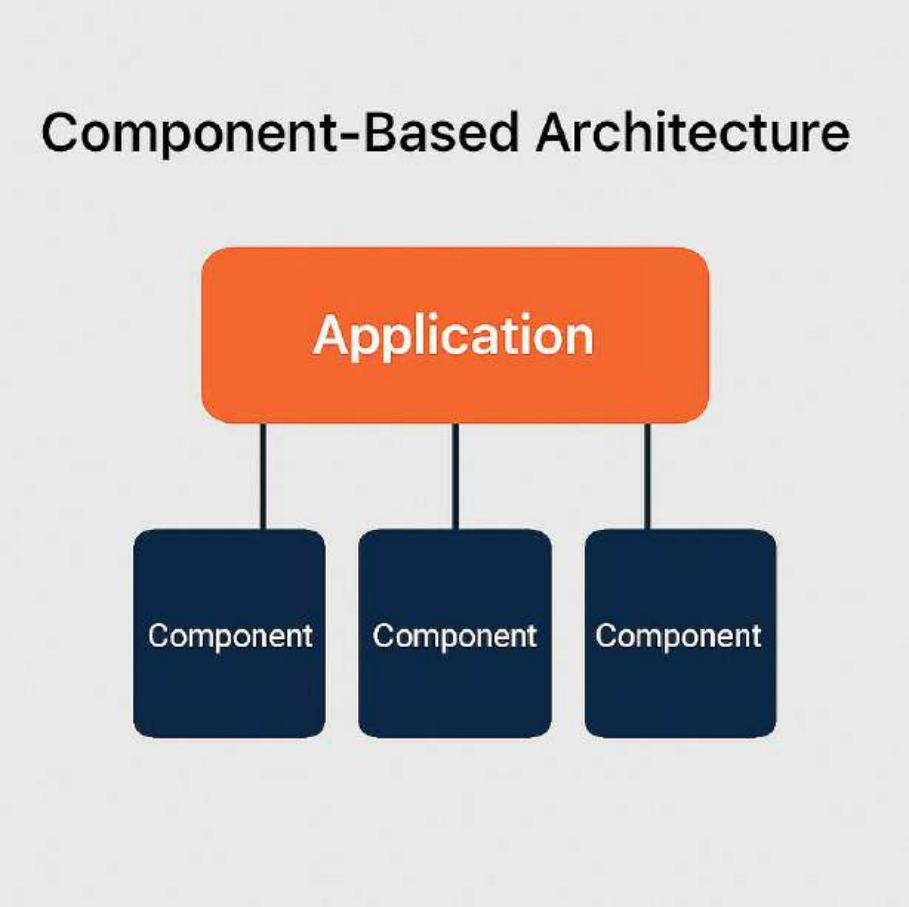


Figure 3 - Component-Based Development Architecture

Design systems improve the component-based development process by allowing elements to be implemented as components throughout the design environment. Through the usage of components, design and code outputs become more consistent providing accurate representation during the hand-off process between the design and development pipeline.

The use of CBD in design tools has increased the rapport between the design and development pipeline. Figma is an example of a tool that integrates the use of reusable components and elements, which demonstrates how the workflow allows for more accurate outputs across the development process (Huang, 2024).

4.3.2. Design Tool Extensibility and Plugins

Plugins increase the overall accessibility and features of design tools such as Figma, as they are enabled to extend functionality, increase production rate of designs, and use internally developed features to improve the efficiency of workflows. Through the integration of design tokens and AI tools, the development process is simplified and provides consistent outputs (WebThesis, n.d.).

Plugins can extract properties and information from designs and translate them into usable code. The use of design tools and components can reduce the manual hand-off process as designs are converted into code outputs, which increases overall consistency of code quality during the design to development workflow (Xiao et al., 2026). This factor is important as the project prioritises translating high-fidelity designs into automated code outputs through a Figma plugin.

4.3.3. Design-to-Code Translation

Design components can be converted into functional code which is important in the CBD workflow and design process. Style properties such as, spacing, colour, typography and layout are capable of being translated into generative code outputs. This process decreases the requirement for manual interference with the hand-off process (Chen & Chen, 2025). By increasing the quality of automated code outputs, fidelity of designs is less likely to be sacrificed, which ensures reusability of designed components (Ammattikorkeakoulu, 2018).

Design tools such as Figma provide a design environment suitable for the design-to-code pipelines where designs are converted from visual elements into components using frameworks, such as, React and Tailwind CSS. This insight provides an understanding of component structures and how generated outputs can support the maintainability of the design-to-code process.

4.3.4. Utility Frameworks and Token-Based Workflows

Design tokens are a system where properties such as colour, typography, and spacing are managed and accessible for design uses. Huang (2024) introduces the Figma-Enhanced App Design (FEAD) framework that implements the use of design tokens into applications to maintain a consistent design structure throughout.

Utility-first frameworks such as Tailwind CSS improve on this token system, as these frameworks provide reusable styling methods that cohesively work with the token-based systems (Saari, 2019). This workflow process is important for the project as mapping of Tailwind CSS into tokens is used to generate code outputs and maintain consistency of design to development integration.

4.3.5. Limitations of Automated Code Generation

The implementation of automated code generation is effective for the efficiency of design-to-code workflows. The usage of generated code increases the productivity of design to development pipelines as designs are translated into code during the hand-off process. However, this increase in speed can cause faults with consistency and readability of code. This flaw can increase the time spent manually debugging the poor code structure to produce usable components (Chen & Chen, 2025).

The development of component-based user interfaces enhances the design to code workflow, however it requires observation of developers to prevent automation from neglecting the quality of code when translating high-fidelity designs (Hapuli, 2025).

4.4. Technical Research

4.4.1. Figma Plugin API

The Figma Plugin API allows developers to communicate between the design elements and the design environment within the Figma application. It provides access to the Figma nodes where it views design properties of the node, extraction of these properties is available to further convert them into code outputs, these properties include spacing, colour, layout and typography (Kanapathipallai and Priyankara, 2026).

The communication between the user interface and the Figma environment is processed through messages. However, Figma provides strict communication rules where only the UI and the primary backend code file are capable of communicating with the Figma environment. This restriction guides developers to research communication methods between the platforms, this results in the use of bundling tools. Additionally, automated tools integrated in the Figma plugins are growing in popularity as they assist

in generating designs, code outputs and wireframes, which assists in supporting the design to development workflow (Ganesaraja, et al., 2025).

4.4.2. Frontend Architecture Decisions

Figma plugins are restricted to the UI interface code and the primary functionality code files. This constraint is filtering communication between the designer and the environment narrowly and causes limitations for user interactions with the plugin. The architectural structure contains limitations in comparison to modern frontend architectures that allow for a wide range of state management through the use of frameworks (Ganesaraja et al., 2025).

4.4.3. Code Generation Techniques

Code generation uses the methods of mapping properties into code outputs, specifically in this project mapping Figma node properties into automated code outputs, in multiple language-style pairings. Automated code generation increases the efficiency of design to development pipelines as there is a decrease in code redundancy, as well as improved accuracy in outputted code (Chen & Chen, 2025).

This system requires extraction of the design properties, including layout, styling and structure of the component. These extracted elements are then mapped to the Tailwind CSS token groups and further translated into generated code outputs.

4.4.4. Constraints of the Plugin Environment

Figma plugins perform within sandbox environments which contain limits on storage and performance. These constraints reduce the scalability, however this improves the stability when developing within the platform. Additionally, this flaw prohibits developers from communicating with the environment freely, as plugins must communicate through messages, therefore message handlers are crucial to plugin development (Gui et al., 2026).

4.5. Comparative Study

This section performs a comparative study of the proposed plugin against current Figma Plugins, specifically Builder.io and Figma to Code (HTML, Tailwind, Flutter, SwiftUI). This comparative study focuses on code generation, quality of code, accessibility of frameworks and usability of the interface, this will assist in recognising strengths and weaknesses of the current design to development pipeline.

Builder.io is an advanced AI-based design-to-code conversion tool, with capabilities to translate Figma designs into generated code through multiple framework outputs, including React, Next.js, Vue, Angular, Flutter and more. By examining the documentation, an understanding of the conversion process to frameworks is apparent. The tool uses an AI pipeline to structure code outputs to each framework, this enables automated generation of intricate user interfaces (Chen & Chen, 2025). This plugin provides code outputs that maintain structure and layout without affecting the fidelity of the design. The system also allows for integration with existing components and iterates these designs into the AI translation. An overview of the user interface is displayed in Figure 4. This image shows the navigation tabs for design exports, AI integration and code builders.

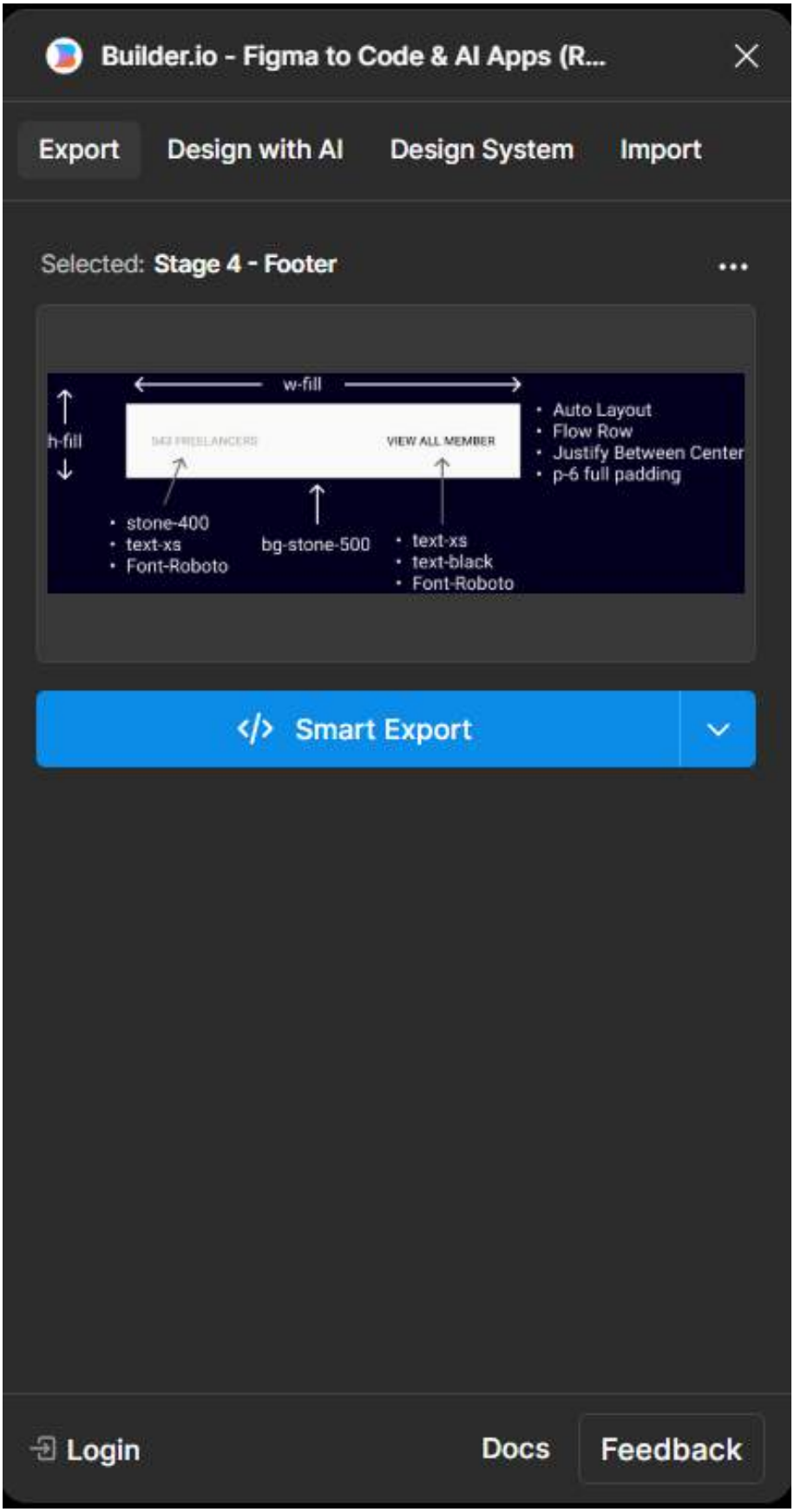


Figure 4 - Builder.io Plugin User Interface

Although the plugin contains a wide range of features, the use of AI presents limits. The plugin heavily relies on the use of AI to generate efficient and scalable code outputs which can cause issues with readability of the code. If the AI tool prioritises speed, the integrity of the code is lost (Chen & Chen, 2025). This will further affect the visual output of code as high-fidelity designs will become inaccurate. The plugin itself is reliant on user awareness and previous knowledge of use, this reduces the effectiveness of the workflow as new users tend to spend more time learning the functionality of the plugin. However, some features of the plugin are blocked by paywalls, which limits the scope of users, as displayed in Figure 5.

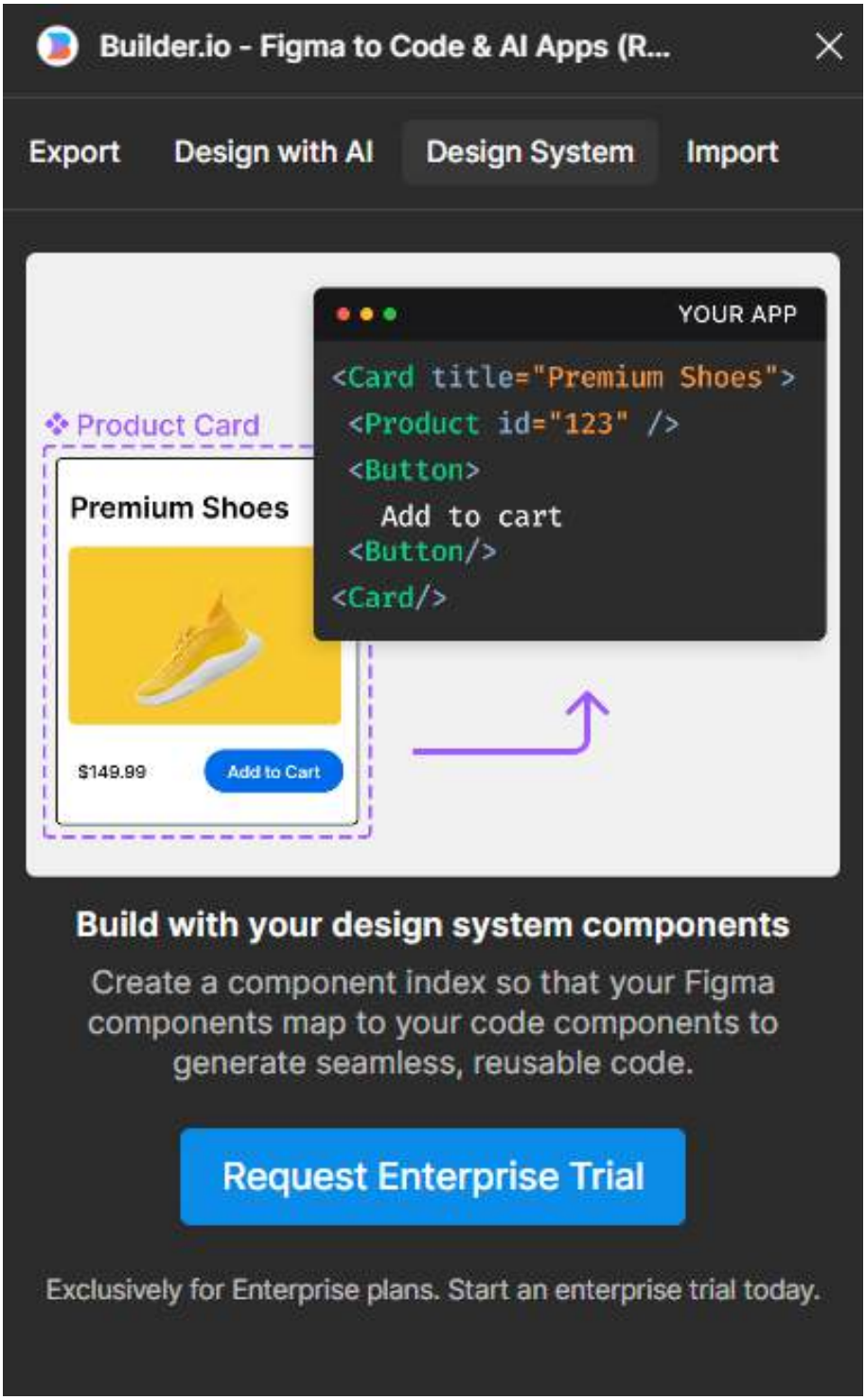


Figure 5 - Builder.io Paywall

On the other hand, the Figma to Code plugin enquires a direct approach to code generation. The plugin translates selected designs directly into both web application and mobile application formats. It supports numerous frameworks including, HTML, React (JSX) and Svelte for web applications, Flutter and SwiftUI for mobile applications, as shown in Figure 6. The plugin prioritises producing quality code outputs while layout

properties are protected, which preserves the fidelity of designs and reduces the requirement for bug fixing during the hand-off process (Xiao et al., 2026).

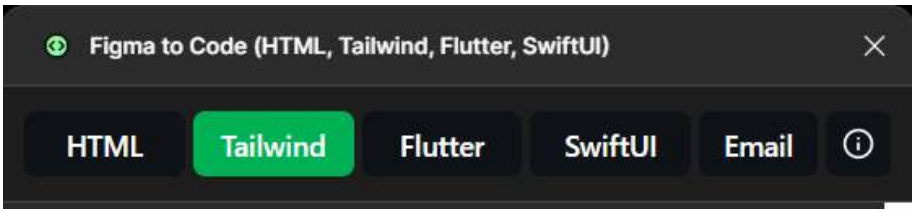


Figure 6 - Figma-to-Code Frameworks

This plugin is proficient in ease of use and user accessibility, however, it contains limits for the plugin scope. Figure 7 demonstrates how the generated code outputs contain complicated structuring which requires manual translation of components to become reusable. This process complicates the workflow and can issue further complications throughout development. This reflects the requirement for a balance between code quality and automation, as workflow efficiency is guaranteed with fidelity of designs being preserved during translation (Chen & Chen, 2025). Along with code output flaws is the user interface design, navigation of the tabs within the code output is time consuming, especially when deep scrolling horizontally and vertically is required to view the generated code outputs.

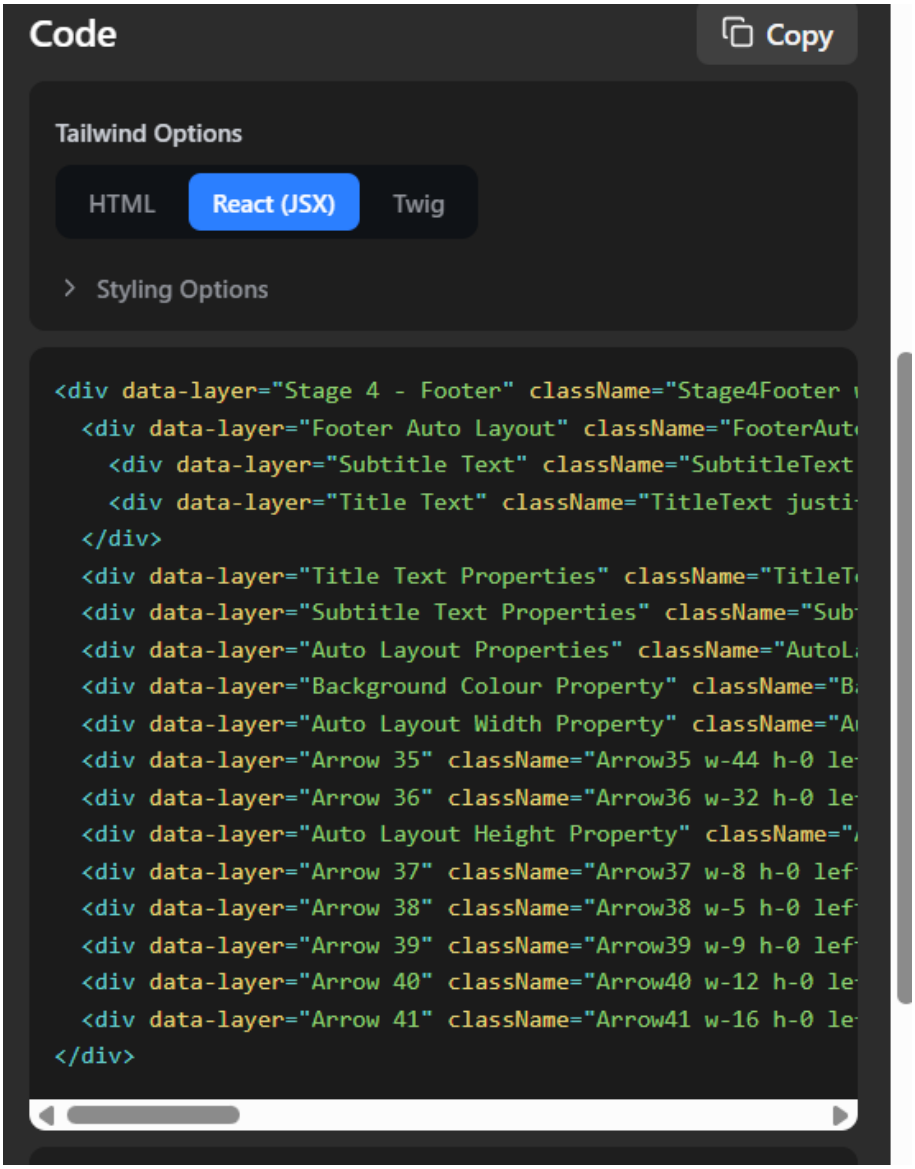


Figure 7 - Figma-to-Code Code Outputs

The proposed plugin differentiates itself from both compared plugins, as it focuses on balancing code generation with consistency and user control for design. The plugin avoids reliance on AI integration, by instead using a token-based system to map element properties into generated code outputs. This plugin improves on the basics of Figma to Code as it incorporates Tailwind CSS into the design system through tokenization. Through the integration of tokens into the design process the consistency of design and code outputs are improved (Ammattikorkeakoulu, 2018).

The plugin opposes Builder.io as it prioritises fast-paced exportation of code generations, by providing a copy to clipboard and download as txt file button. The proposed plugin aims to advance on user interfaces to create a user-friendly system for designing, whilst maintaining the structure and efficiency of the development process. The token and search functionality features simplify the overall navigation and

familiarity of the plugin interface and generated outputs. The user interface is presented in Figure 8, where clear navigation, filtering and searching options are available.

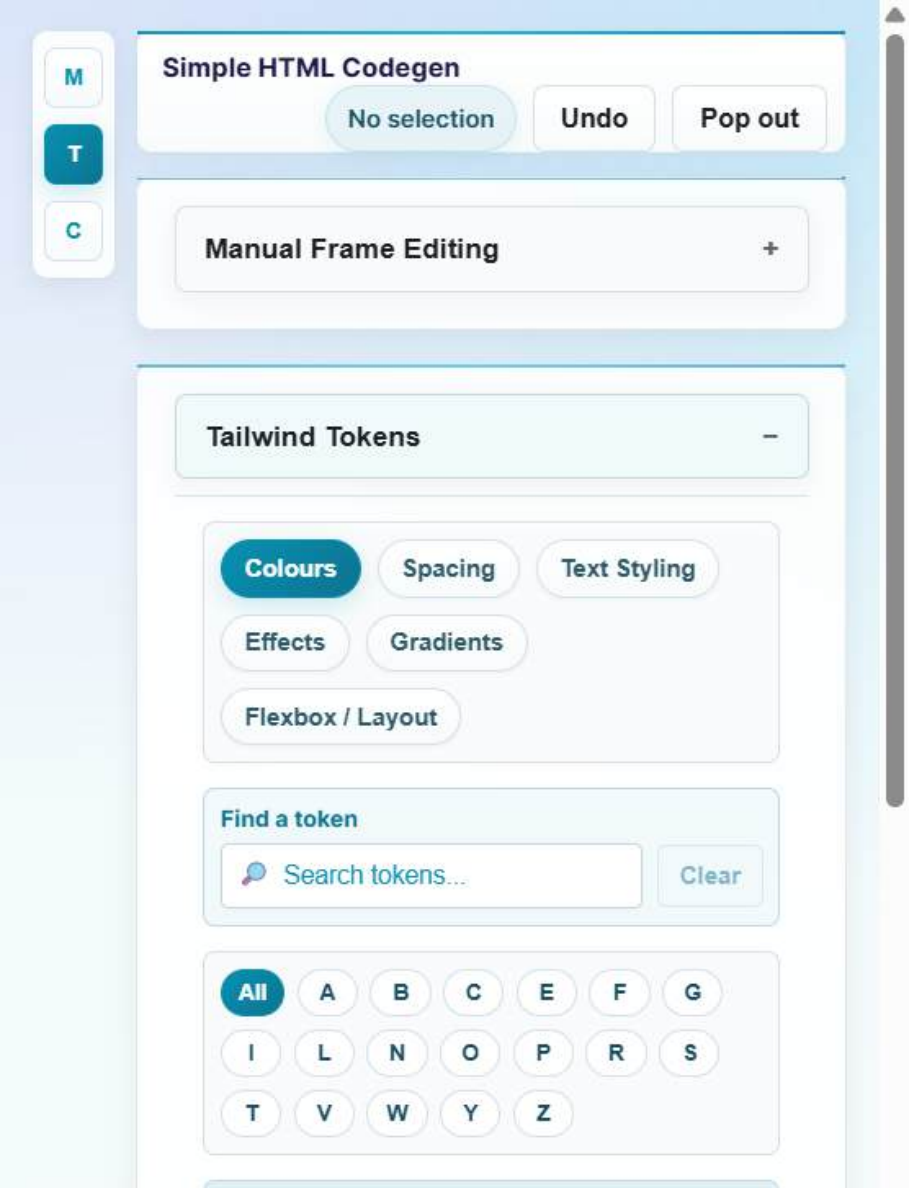


Figure 8 - Proposed Plugin User Interface

Overall, the comparative study describes the gap between current tools, how AI integration provides a wide range of code generation frameworks but lack speed. The proposed plugin focuses on consistency through a token-based system which increases the overall balance of the design-to-code workflow.

5. Requirements Analysis

5.1. Requirements Gathering

The requirements for the project were collected through user surveys, observational analysis and evaluating of current design-to-code tools available. The use of multiple sources is to ensure the system is fully explored through all stages of development while identifying issues throughout the process.

Initially data was collected through a survey created on Google Forms that was directed towards the workflows of frontend developers, usage of design tools and importance of design-to-code tools in current design to development pipelines. The results of this survey indicated that developers are biased towards tools that include the React and Tailwind CSS frameworks, however errors occur frequently during the translation of the designs into generated code outputs. This includes inconsistencies with spacing values, property mapping to code and generating redundant code. These concerns are displayed in Figure 9, where the users indicated that spacing, layout and responsiveness are decremental to workflow efficiency.

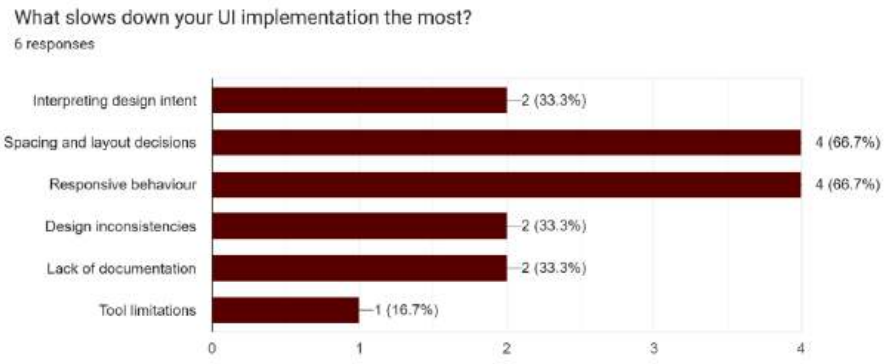


Figure 9 - Survey Feedback Highlighting Spacing, Layout, and Responsiveness Concerns

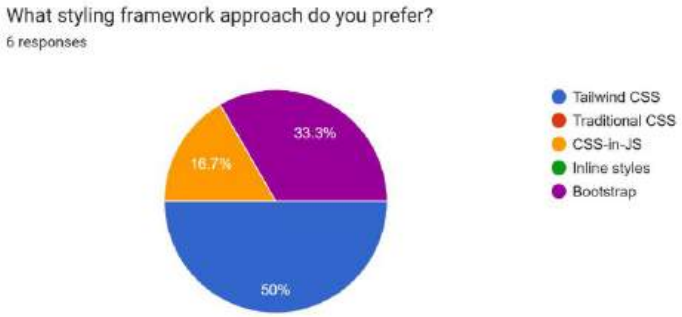


Figure 10 - Survey Feedback Highlighting Framework Preference

Additionally, Figure 10 showcases the preference towards available frameworks for designing, which leans intensely towards Tailwind CSS. However, results further implied the current issues and concerns with design to development workflows, as shown in Figure 11. Users state a need for efficient design and navigation of a system to reduce style issues, learning process and overall disorganisation.

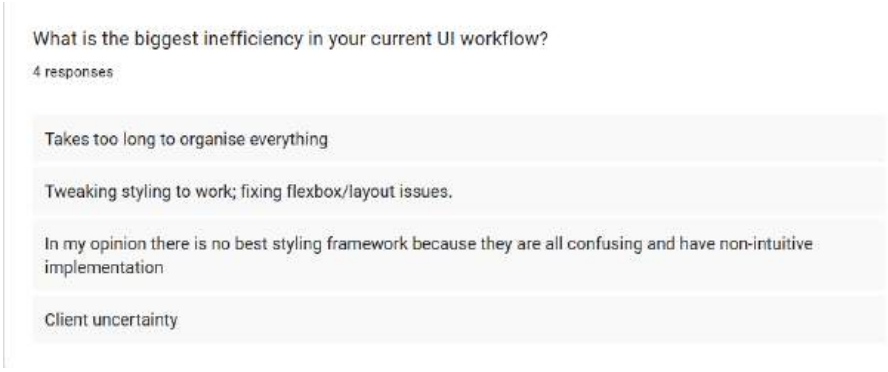


Figure 11 - Workflow Concerns

Once initial development of the complete plugin was created, another round of testing was performed through user testing and feedback. This process was to specify the errors that occurred when replicating a design with the use of the plugin. The feedback in Figure 12 highlights issues with navigation being unclear, tab dimensions were breaking during tasks, and limitations on design options available. These discoveries further influenced the development of the system and the functionality requirements set to complete.

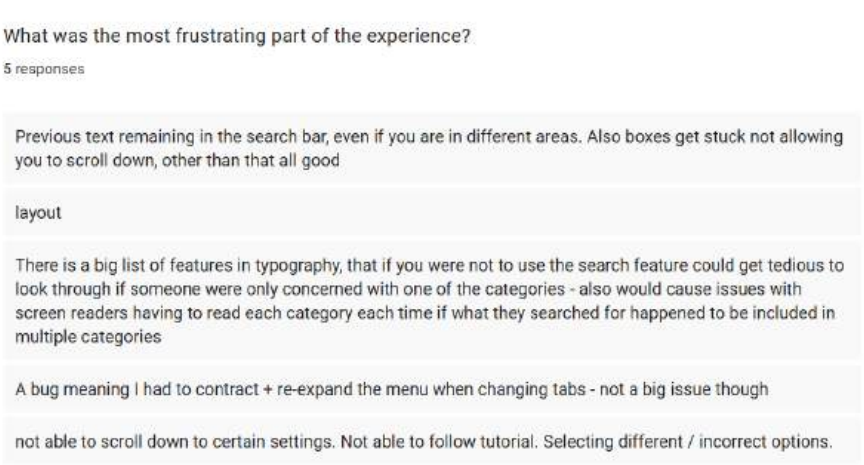


Figure 12 - User Testing Frustrations

Additionally, an observational study was conducted on existing Figma plugins, including Builder.io and Figma to Code. The comparative of these plugins resulted in an understanding of a divide in the systems where quality and consistency are sacrificed for a wide range of features and functionality. These plugins communicated with multiple frameworks to output generated code, however this cost the system speed and

structure in the outputted components. This provided advice to focus on balancing both efficiency with maintainability when designing code generators (Chen & Chen, 2025).

Overall, the requirements collected through this process corrected the aims and objectives that this system is required to achieve to produce an effective product. The use of multiple resources for information provided a stronger understanding of the requirements missing in the current state of design to development pipelines.

5.2. Requirements Modelling

5.2.1. User Groups and Personas

The plugin is designed to accommodate the design to development workflow, primarily focused on frontend developers, UX/UI developers and full-stack developers where users incorporate design tools to create components and elements to translate into code outputs for project usage. These groups were surveyed and tested to identify common issues and occurrences while completing tasks in the design environment.

To display these user groups, personas were created, a persona reflects a user, their background, goals, frustrations and requirements from the plugin. This assists in guiding the development of the plugin to discover the user needs and issues.

Figure 13 - Persona 1 Frontend Developer

Figure 13 showcases the first persona, which is a frontend developer who makes use of the Figma application and design tools to translate designs into React components using Tailwind CSS. This persona demonstrates the issues with manually translating designs into components, as they often encounter spacing and styling inconsistencies, this requires further manual hand-off work and repetitive development. These frustrations provide context of the necessity for extraction of designs and automated conversion into code outputs.

Persona 2 - Full Stack Developer
Name: Daniel Cliff
Age: 32
Role: Full Stack Developer
Experience: 8 years
Tech Stack: React, Next.js, Tailwind, Node.js

Background:
Builds complete applications from backend APIs to frontend UI. Often works independently or in small teams where efficiency is critical.

Goals:
Rapidly prototype interfaces from design files
Maintain scalable design systems
Keep frontend code consistent with Tailwind scaling

Frustrations:
Designs not structured properly for development
Time spent rebuilding layout structures from scratch
Lack of tools that connects design and production code

Needs:
Ability to select a Figma frame and extract structure
Token-based styling that aligns with Tailwind standards
React components ready for production workflows

How the Plugin Helps:
The plugin reads the selected frame, maps design properties to Tailwind tokens, and exports structured React components suitable for real-world applications.

Figure 14 - Persona 2 Full-Stack Developer

On the other hand, Figure 14 describes the second persona, a full-stack developer who requests efficient and scalable systems for designing applications. They state the importance of reusability in designing applications. The requirement for readable, consistent code is important for the development of a code generation feature.

Persona 3 - UI/UX Designer
Name: Sofia Sloan
Age: 29
Role: UI/UX Designer
Experience: 6 years
Tools: Figma, Prototyping Tools, Adobe XD

Background:
Designs user interfaces for web applications and collaborates closely with developers to ensure accurate implementation.

Goals:
Ensure design consistency across components
Reduce design-to-development miscommunication
Use design tokens to maintain visual consistency

Frustrations:
Developers misinterpreting spacing, typography, or colours
Design systems not translating cleanly into code
Repeated clarification of design specifications

Needs:
Clear mapping between design tokens and development styles
Tools that maintain consistency between Figma and code
Faster handoff from design to development

How the Plugin Helps:
The plugin allows designers to apply Tailwind-based tokens within Figma, ensuring the design values directly translate into developer-ready code.

Figure 15 - Persona 3 UX/UI Developer

Finally, Figure 15 details the third persona, a UX/UI designer who primarily creates user interfaces and requires accurate translation of stylings. This persona expresses the errors that occur from inconsistent design to development hand-offs, particularly the spacing, typography and colour properties. This information assists in the development of the token-based system within the plugin allowing for accurate and consistent design throughout the entire hand-off process.

Throughout the evaluation of each persona similarities are displayed, including manual translation during the hand-off process and request for consistent and accurate code generation. These goals contribute to the development plan for the project as it provides guidelines for the success criteria and the requirements. This proves the necessity of evaluating personas as they provide features for the implementation process.

5.3. Use Cases

The functionality of the plugin system is introduced through user interactions which defines the workflow of the plugin. This is represented through the use of a UML use case diagram. The diagram presents the interactions that occur during usage of the plugin system. The roles of the interactions are the User (Designer / Developer), and the External System (Figma Plugin API).

The User represents actions that occur when interacting with the frontend of the system, including selecting frames, previewing generated code, and applying style tokens. While the External System is required to provide a service to assist in the system's functionality, including providing Figma node data and communicating with the plugin to provide a connection to the Figma environment. The interactions described are exhibited in Figure 16, as it displays the relations between the User and External System with the plugin.

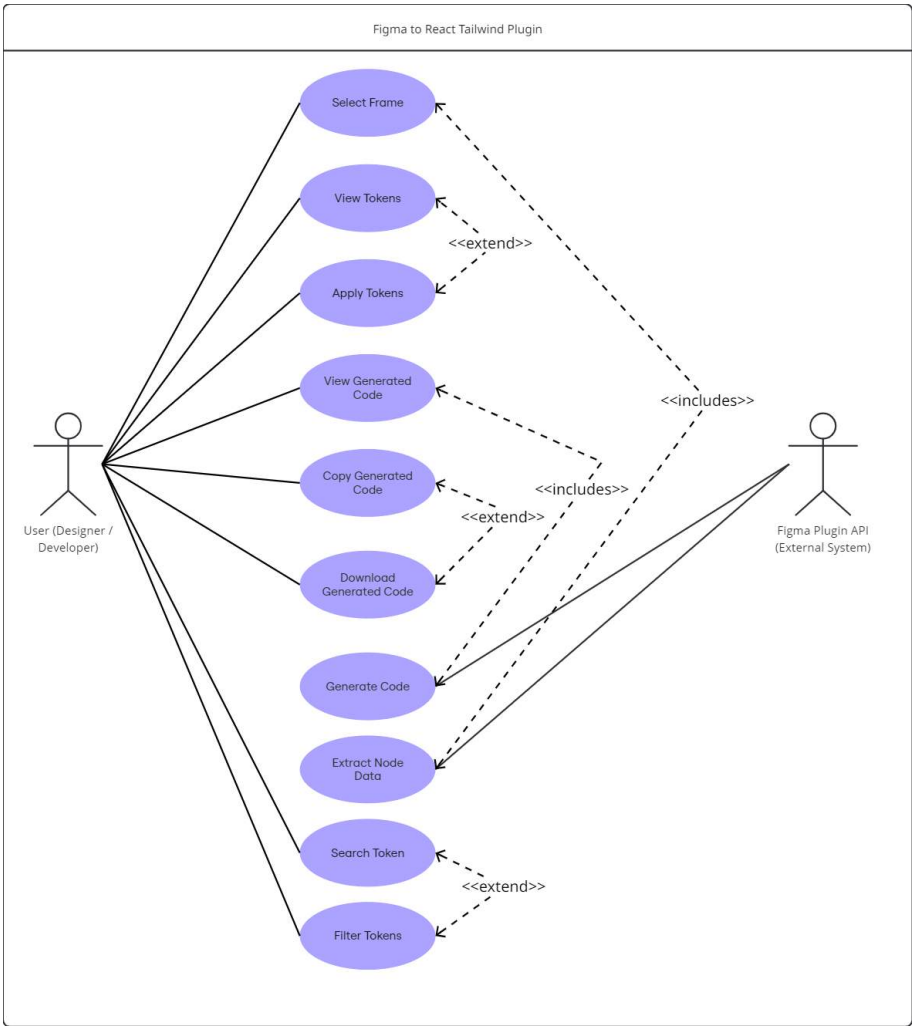


Figure 16 - Use Case Diagram

5.4. Functional Requirements

The functional requirements are the definition of the system and what the plugin's potential involves. These are created and altered based on both progression of the application, and feedback through user testing. The requirements provide a perspective on how the application should function as a system, this includes communication between users, the Figma API and the plugin system itself. These requirements range from simple actions such as selecting a Figma frame, and applying a token, to in-depth actions and algorithms such as mapping values to Tailwind tokens and translating visual designs into code.

The functional requirements are outlined in Figure 17, which describes features of the plugin, including the extraction of Figma nodes by communicating with the Figma Plugin API, as well as previewing the automated code outputs in the plugin user interface panel. The behaviours of the system are included in the requirements as features such as importing Tailwind tokens, generating React framework code output and providing an export function for code generation, these all clarify distinct areas of the workflow that builds the plugin for designers and developers to use it accurately.

1	Priority	Feature	Description
2	Functional	Frame Selection & Node Extraction	Allow user to select a Figma frame and extract node hierarchy, including nested layers, auto-layout structures, spacing, sizing, typography, and colour properties.
3	Functional	Tailwind Token System	Provide a structured library of Tailwind tokens (colour, typography, spacing, sizing, layout) that map extracted Figma properties to Tailwind utility classes.
4	Functional	Token Application Interface	Provide a plugin UI where extracted node values and available tokens are displayed, allowing users to apply Tailwind tokens directly to selected nodes.
5	Functional	React + Tailwind Code Generation & Export	Generate structured React components using Tailwind utility classes based on the processed layout and allow code to be copied or exported for development use.
6	Functional	Auto-Layout Interpretation	Detect Figma auto-layout structures and convert them into appropriate Tailwind flexbox layouts.
7	Functional	Component Hierarchy Preservation	Maintain nested layer hierarchy when generating React component structures.
8	Functional	Developer Code Preview	Display corresponding Tailwind classes and generated code within the plugin for developer reference.
9	Functional	Editable Node Values	Allow users to modify extracted node values or tokens before generating final code output.
10	Functional	Custom Token Creation	Allow users to create and manage their own token sets within the plugin.
11	Functional	Tailwin Theme Export	Export tokens as a Tailwind configuration theme file for integration into development projects.
12	Functional	Component Library Export	Export generated components as reusable React component files.

Figure 17 - Functional Requirements

5.5. Non-Functional Requirements

The non-functional requirements are examined through the quality of the plugin and the system, whether it can perform at a higher demand and if the plugin communicates accurately with the system and Figma Plugin API. These requirements are showcased in

Figure 18, providing an understanding of the deliverables that the system performs, including performance, node extraction, and code generation.

1	Priority	Feature	Description
2	Non-Functional	Usability	UI must be clear for designers/developers to apply tokens easily
3	Non-Functional	Performance	Node extraction and code generation should be fast
4	Non-Functional	Accuracy	Tailwind mapping must correctly reflect Figma properties
5	Non-Functional	Maintainability	Generated code should be clean and structured
6	Non-Functional	Compatability	Must work reliably with Figma Plugin API
7	Non-Functional	Scalability	Should be able to handle large, nested Figma frames

Figure 18 - Non-Functional Requirements

While the functional requirements focus on developing the system to work entirely, the non-functional requirements ensure that the plugin provides a higher quality of services. These services are consistent and accurate mapping of tokens and generated code outputs, and delivering code that is reusable, structured correctly and optimized for project usage.

5.6. Prioritisation

The system requirements follow the MoSCoW method, displayed in Figure 19, which demonstrates grouping the requirements into core groups of necessity (Stewart & Shi, 2026). Within the must-have group lies node extraction, code generation and the plugin interface. While the should-have requirements contain the auto-layout styling, a code output preview and editable nodes. Lastly the could-have incorporates the requirements that are least likely to be implemented due to time restrictions or lack of knowledge. These included custom tokens, component libraries, and Tailwind Themes.

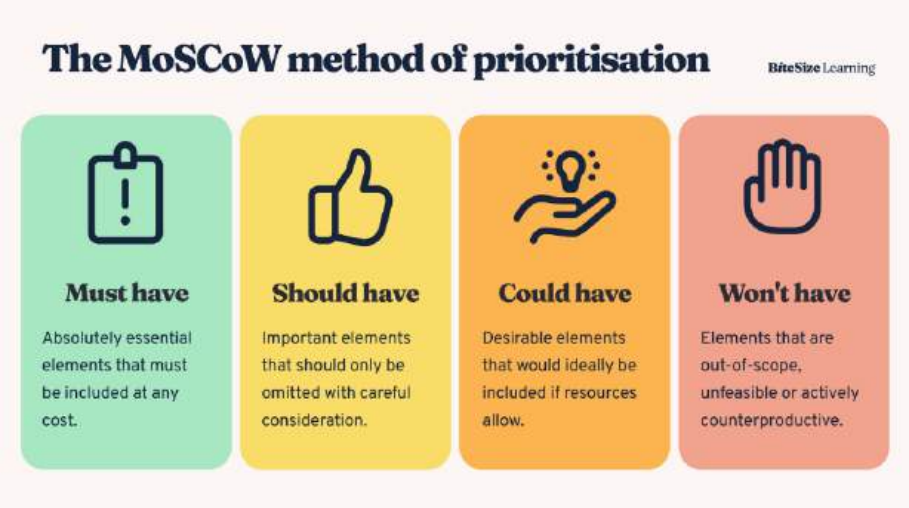


Figure 19 - MoSCoW Methodology

5.7. Requirement Evolution and Mapping

Throughout the development of the plugin system, the guidelines that are followed primarily rely on the requirements drafted. However, these requirements can evolve, as the project advances requirements can be accomplished sooner than expected, which indicates new requirements are necessary to continue progressing the project in a positive direction. The evolution of these requirements is presented in Figure 20 and Figure 21, which compares the first and second iterations of requirements.

Initially the project requirements scope was narrow as the understanding of the topic area was limited, this guided the requirements in a broad direction avoiding difficult challenges. The project expectations were focused on extracting Figma node properties and incorporating an editable frontend for users to interact with. These requirements were reflected on once the “must” and “should” points had been accomplished. The concept built for the plugin had evolved further to include difficult features, such as Tailwind tokens, custom tokens, and auto layout styles.

The alteration between the sets of requirements displays in what ways the project itself has progressed from mid-level expectations of node extraction, frame selection and code generation. Now the project is proposed to integrate a deeper understanding of plugin functionality and development, as well as the Figma Plugin API communication. These newly added requirements include multi-framework code generation and custom token generation of gradients and colours.

This refinement of requirements encourages that the plugin produced at the end of this project will demonstrate the technical learnings throughout the project timeframe.

1	Priority	Feature	Description
2	Must	Node Extraction	Extract layouts, auto-layouts, nested components, spacing, sizing, typography, and colours from Figma frames.
3	Must	Mapping Logic	Convert Figma node values into Tailwind utility classes using scales, handle arbitrary values.
4	Must	Code Generation	Generate HTML and React + Tailwind components.
5	Must	Plugin UI Panel	Display extracted values for developer preview and editing
6	Should	React Component Structuring	Apply component hierarchy in output for immediate integration
7	Should	Preview & Copy	Provide live preview of generated code, allow copy to clipboard
8	Could	Tailwind Theme Tokens	Allow users to export values using Tailwind custom themes.
9	Could	Code Export	Enable downloadable export of generated code.

Figure 20 - First Iteration of Requirements

1	Priority	Feature	Description
2	Must	Frame Selection & Node Extraction	Allow user to select a Figma frame and extract node hierarchy, including nested layers, auto-layout structures, spacing, sizing, typography, and colour properties.
3	Must	Tailwind Token System	Provide a structured library of Tailwind tokens (colour, typography, spacing, sizing, layout) that map extracted Figma properties to Tailwind utility classes.
4	Must	Token Application Interface	Provide a plugin UI where extracted node values and available tokens are displayed, allowing users to apply Tailwind tokens directly to selected nodes.
5	Must	React + Tailwind Code Generation & Export	Generate structured React components using Tailwind utility classes based on the processed layout and allow code to be copied or exported for development use.
6	Should	Auto-Layout Interpretation	Detect Figma auto-layout structures and convert them into appropriate Tailwind flexbox layouts.
7	Should	Component Hierarchy Preservation	Maintain nested layer hierarchy when generating React component structures.
8	Should	Developer Code Preview	Display corresponding Tailwind classes and generated code within the plugin for developer reference.
9	Should	Editable Node Values	Allow users to modify extracted node values or tokens before generating final code output.
10	Could	Custome Token Creation	Allow users to create and manage their own token sets within the plugin.
11	Could	Tailwin Theme Export	Export tokens as a Tailwind configuration theme file for integration into development projects.
12	Could	Component Library Export	Export generated components as reusable React component files.

Figure 21 - Second Iteration of Requirements

6. Design

6.1. System Architecture

Figure X displays the system architecture of the proposed plugin. The architecture demonstrates unidirectional flow (UDF), which means this application requires a specific action to be taken to extract the data and update the node information. The system is split into multiple components which showcase the divide between the user interface and the plugin system. Through the design of the system architecture, the process that is performed internally is more transparent as connections between the functionality and system itself are revealed.

This system follows a pipeline structure for processing user actions, as demonstrated in Figure 22. As users interact with the user interface the system extracts data from the Figma nodes. The data extracted includes properties such as colour, spacing, typography and layout. This data is directed into the token mapping system where it is translated into Tailwind CSS tokens, and the code generation component, where outputs for code frameworks are produced. This system is designed sequentially as each stage of execution is a unique process. This prevents disruption throughout the entire system.

Overall, the system is designed as a component-based architecture, as the processes integrates smoothly with the design environment of Figma, which allows for an organised pipeline system design (Sparling, 2000).

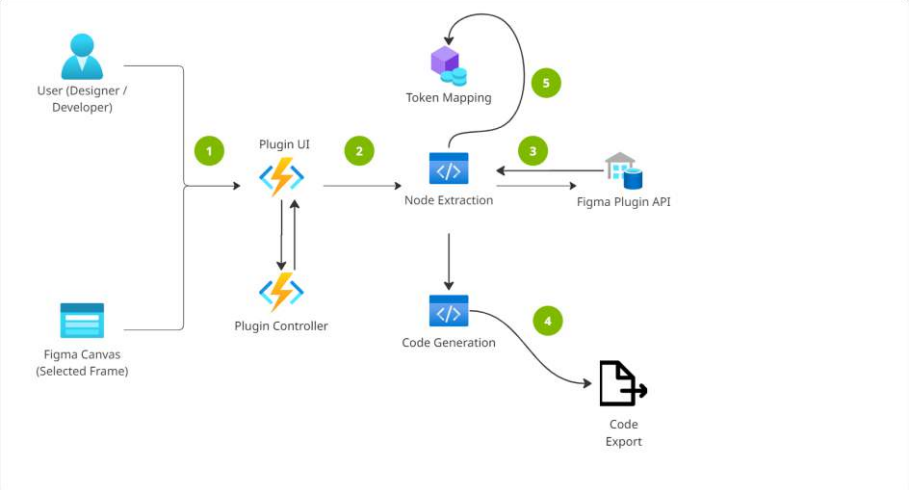


Figure 22 - System Architecture

6.2. Interface Design (UX/UI)

Figure 23 showcases the user flow of actions that occur when interacting with the plugin. These steps begin with starting the plugin application, and end when producing an output of either a design change or code format. Errors are continuously checked for as the key functionality of the plugin does not process if no frame has been selected, as this prohibits users from performing further actions on the canvas.

The system extracts node data from the selected frame which unlocks the ability to alter the design of the frame by overwriting the node property values or generate different framework formats of code outputs.

Applying a token value to a frame follows the path where modification of a node is mandatory, as the values are mapped through the Tailwind CSS token system and replace the node property.

Generating code outputs uses different methods than token application, as the user selects a framework format, this decides how to convert the node values into React with Tailwind CSS or HTML with Tailwind CSS formats, without altering the data of the node itself. These code outputs are available upon completion of the process where users can download or copy the produced code.

Figure 23 - User Flow Diagram

6.3. Wireframes

When developing a system, a series of designs for the user interface are required. These designs range from paper prototypes to high-fidelity Figma produced wireframes. Before creating the user interface for the plugin, paper prototypes were designed initially to propose a starting interface which contains the necessities for the application to function. These wireframes are iterated continuously throughout the design process.

The original designs were developed as low-fidelity paper prototypes, as these allow for swift iterations while adding or retracting features (Miller, 2021). Four versions of these prototypes were designed, each retaining more details and functionality.

The iterations of the paper prototypes are displayed in Figure 24, where the prototypes evolved from minimal layouts with two toggle switches and an output, into designs containing advanced features, buttons, and layout properties. These prototypes are

accurate illustrations of the progressive development of the project's functionality and requirements.

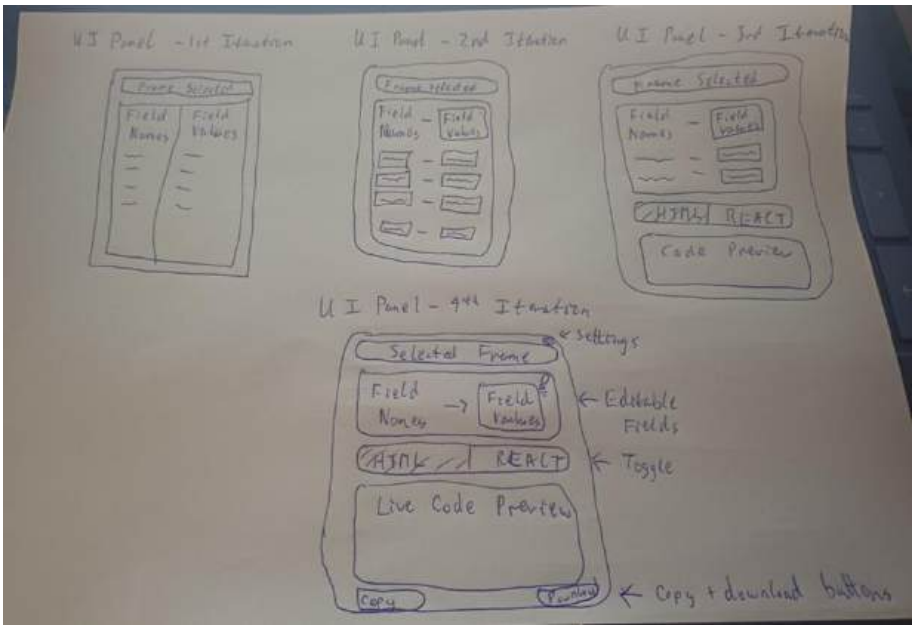


Figure 24 - Paper Prototypes

Following the creation of paper prototypes, designs began to be developed digitally in the Figma environment, as shown through Figure 25 to Figure 28. The virtual wireframes more accurately demonstrate the potential of the user interface, including the concept, layout, accessibility and functionality.

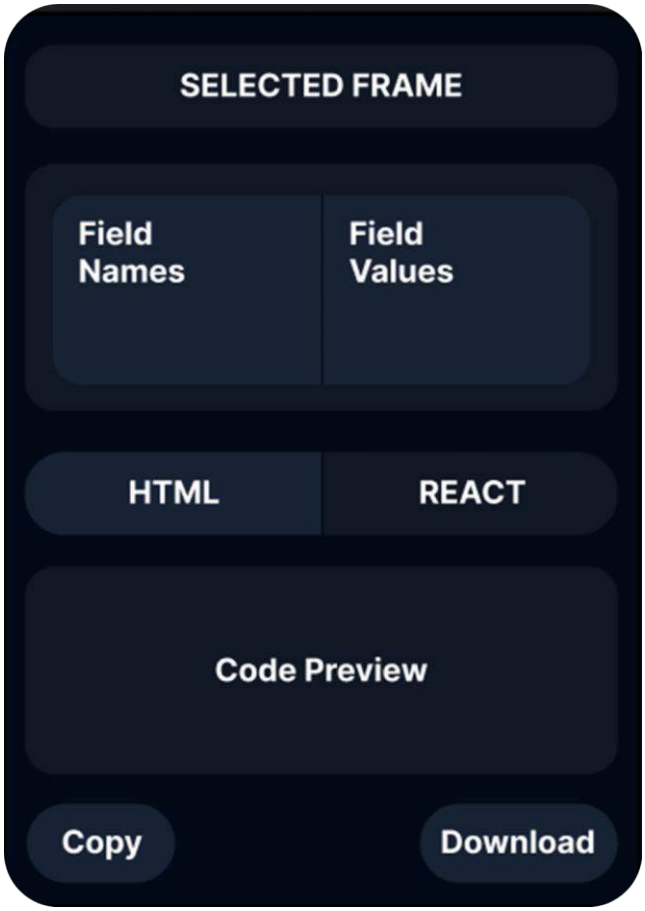


Figure 25 - Digital Wireframe V1

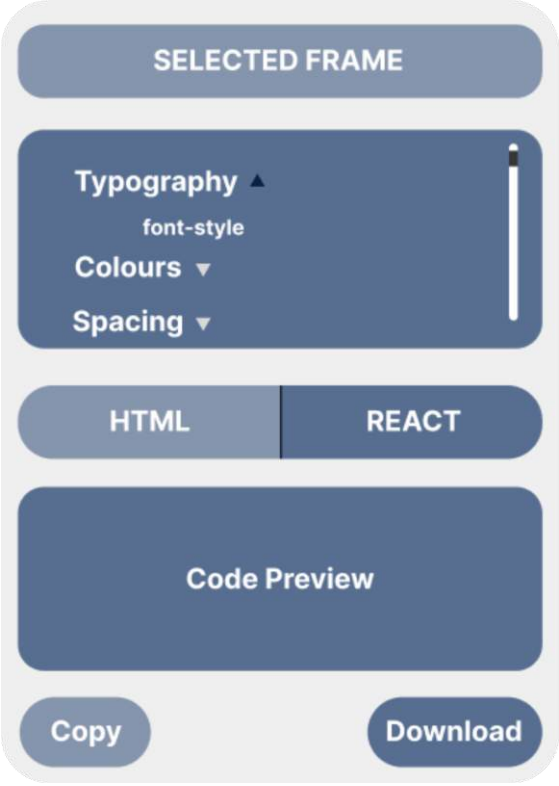


Figure 26 - Digital Wireframe V2

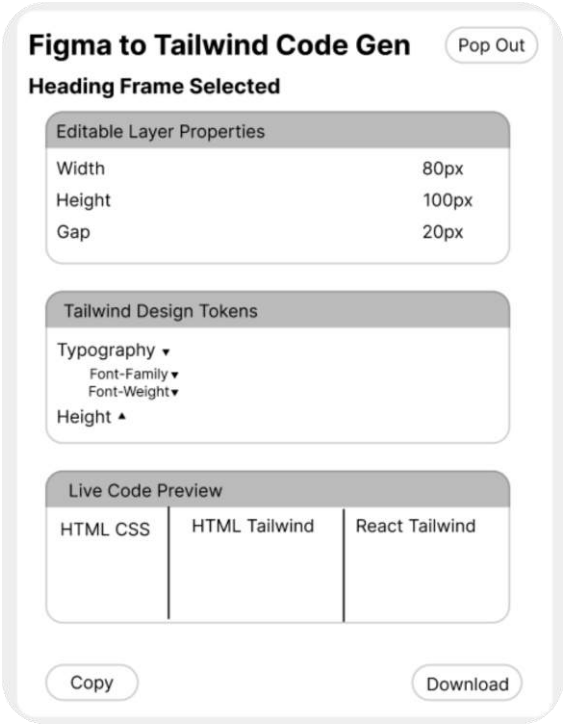


Figure 27 - Digital Wireframe V3

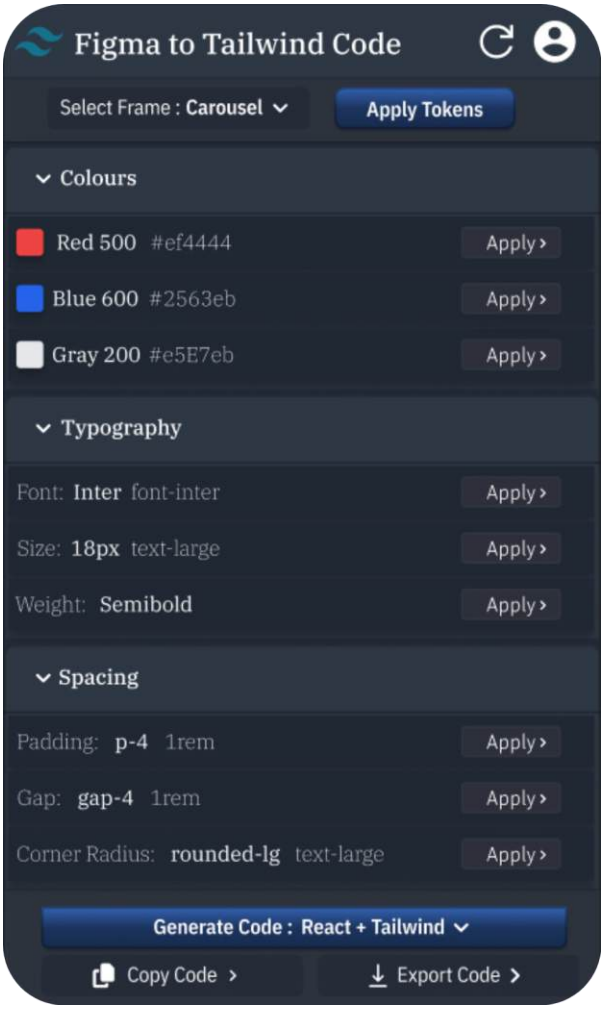


Figure 28 - Digital Wireframe V4

These designs progressed from a simple layout with minimal functionality and less concept, however, as they progressed the wireframes gained additional detail including, a code preview section, individual output tabs, and design themes to follow for layout, typography and colours.

Through the generation of the wireframes a scheme was constructed for the production of a user interface. The iterations performed on the designs provided insight on the issues the interface would encounter and the functionality the plugin system requires.

6.4. Major UI Decisions

The user interface for this project was designed through continuous development over the entirety of the project's timeframe. However, the design was based on efficiency for the design to development workflow, this implied reserving the familiarity of the Figma application design environment and positioning functionality where users expected.

Initially, the design idea was to develop a plugin that presented all utilities available on a single tab. This exhibited concerns early in the development stage as the interface became crowded and messy, leading to an inefficient design. This idea was instead translated into a single plugin panel that contained multiple tabs within for easy navigation and clear design qualities, as previewed in Figure 29.

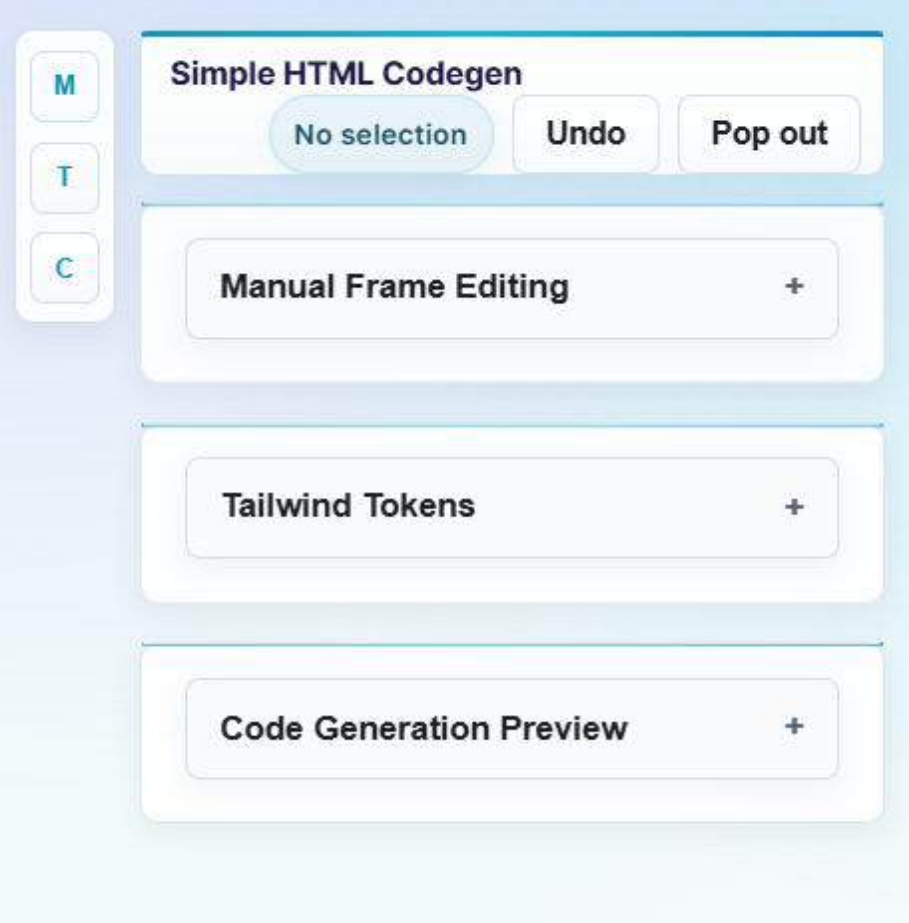


Figure 29 - Multi Tab Design

Furthermore, the implementation of tabs created more opportunities for additional functionality. Users were provided the options to manually alter node properties, select from Tailwind tokens, and preview generated code outputs. However, while developing these sections a hierarchy was required for ease of use, which was decided by testing the process taken to recreate a Figma design. Through evaluating which areas of functionality were primarily interacted with a decision was formed on the hierarchy of the design.

Finally, an important feature of the user interface was introduced after user and observational testing was performed. Evaluating the use of the plugin's user interface with a variety of testers provided insight on the overall design flaws. The feedback received from testing informed the decision to implement quick navigation buttons to increase efficiency, this avoids the repetitive action of navigating between collapsible tabs. Along with navigation, feedback was provided on the dropdown menus for filtering Tailwind tokens, as it appeared overwhelming to search for a particular token solely scrolling through the entirety of the token database. Dropdowns were created to filter specific sections to decrease the clutter and improve efficiency of the plugin's user interface. These improvements to the user interface are showcased in Figure 30.

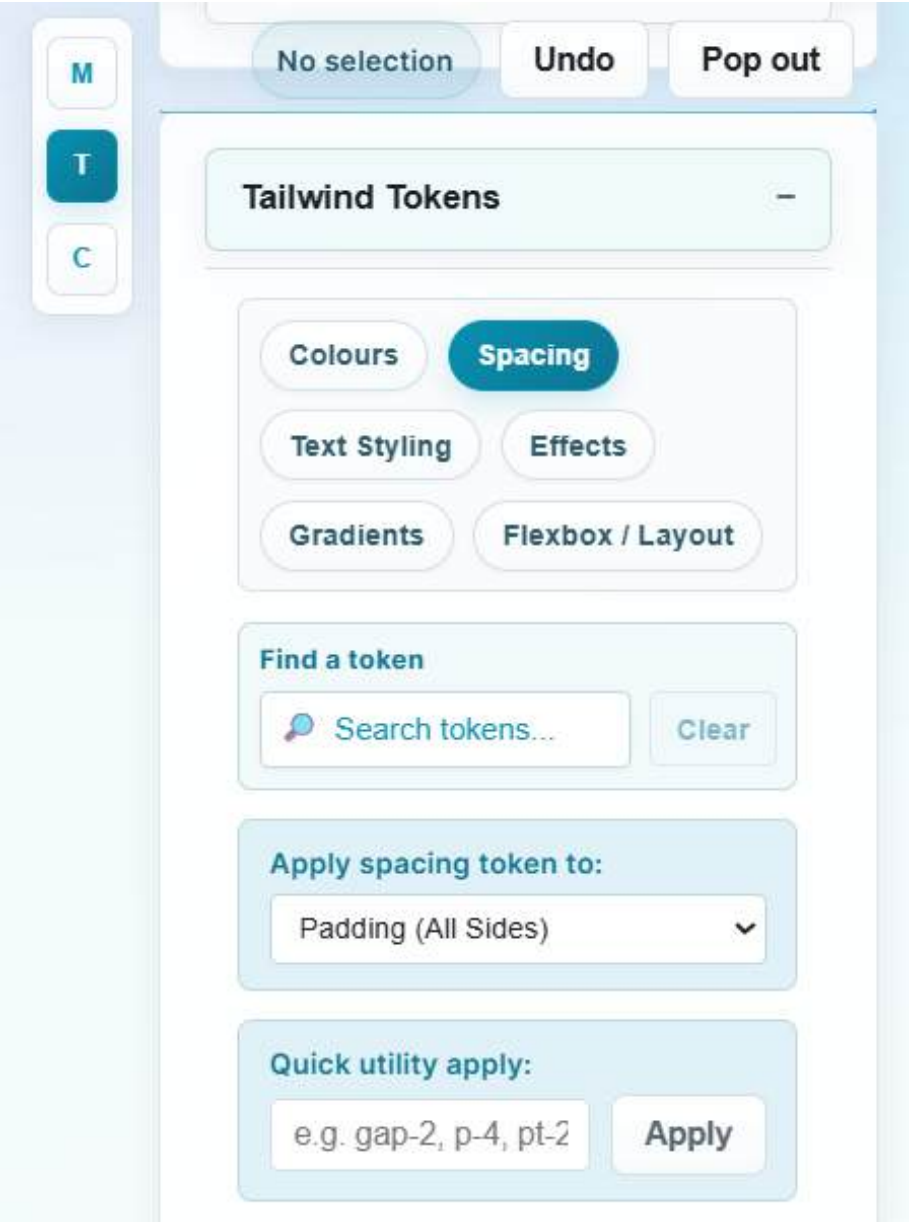


Figure 30 - Design Feedback Implementations

6.5. Style Guide and Tokens

6.5.1. Colour Palette

The design of the user interface was developed with a collection of blue-toned colours. These provide readability for the content within the user interface while maintaining a consistent and aesthetic visual design. The palette displayed in Figure 31 incorporates variations of cyan, slate, and teal at different hue intensities for background colours, tab sections, and search boxes.

The background of the interface remains a light grey with a blue shadowing as the focus remains on the content within the interface rather than the panel itself. This draws attention to the functionality that users may require to complete tasks. The use of variants of blue tones is to differentiate the functionality available, as general tabs remain slate, the options for filtering are distinguished by the purple and blue hues.

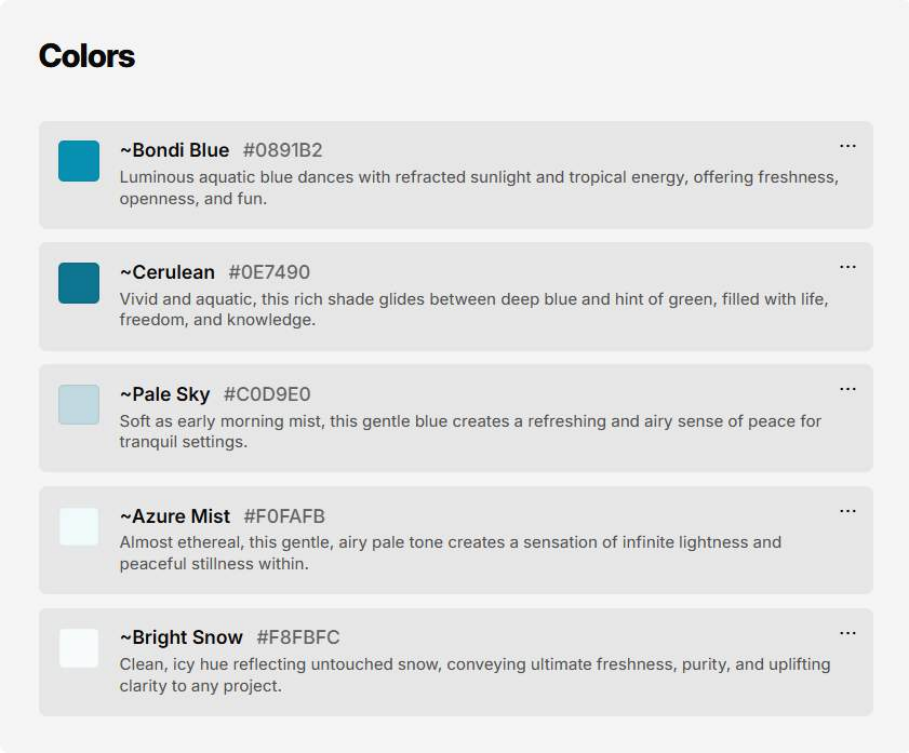


Figure 31 - Colour Palette

6.5.2. Design Tokens

The development of the plugin originally stemmed from creating an advanced Figma design panel where users can edit manually the Figma nodes. This idea evolved into advancing on the original design panel by implementing design tokens. Design tokens are reusable variants that are represented through different design properties, including colour, spacing, typography and layout (Design Tokens Format Module 2025.10, 2026). This implementation provided a consistent workflow from the design process to the exportation of generated code outputs.

Colour tokens were the primary target for implementation as they guided the process of expanding the functionality of the plugin. These colour tokens are designed as a palette which contains varieties of colours and hues, these hues range from values of 50 to 950 in intensity, as shown in Figure 32. Through the application of a colour token to a selected frame in the plugin, the system communicates with the Figma Plugin API to locate the colour value in the node. This colour token is originally a hex value, therefore conversions are required to apply the token as Figma colours follow the RGB value system. The process assures accurate translation of colour tokens to the Figma variant.

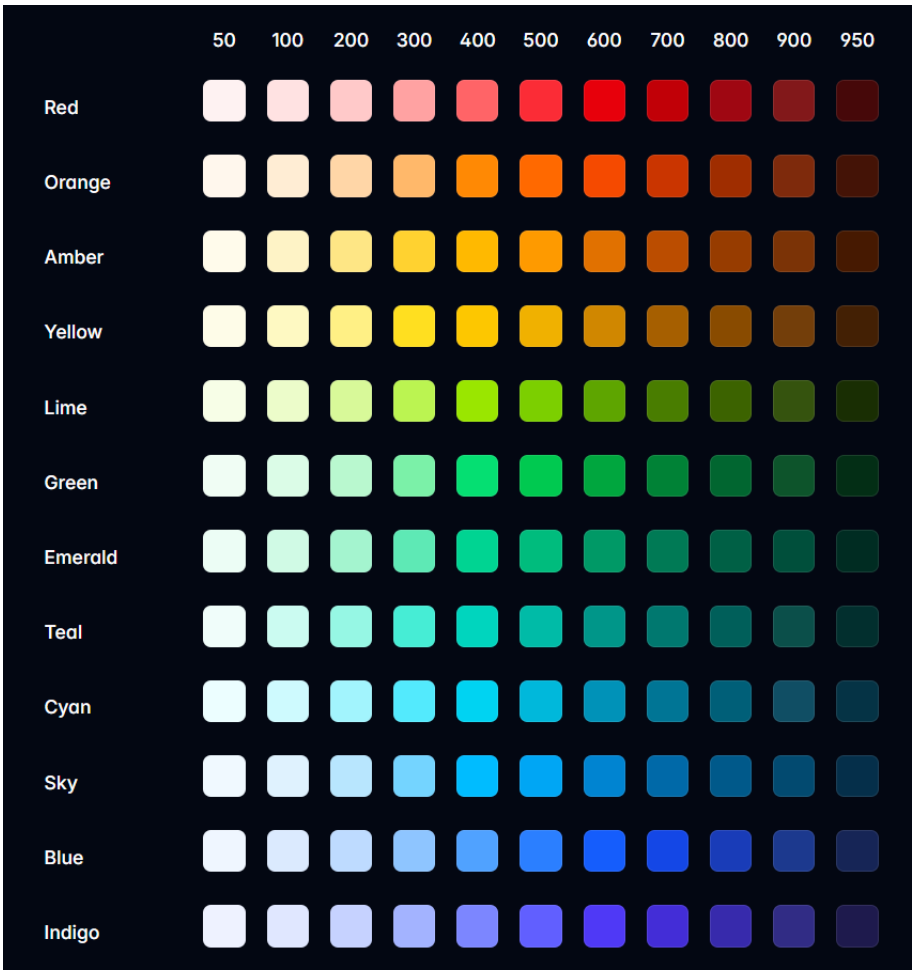


Figure 32 - Tailwind Colour Hues

Apart from the colour tokens remains the spacing, typography and layout properties. These tokens follow similar precautions when translating the values to ensure accurate visual representation in the Figma design. These tokens consist of standard Figma properties, such as padding, gap, width, font size, font families and more. Figure 33 displays, these tokens are paired with values, meaning the tokens are a set of predefined style options. The tokens are integrated into the plugin as buttons, when users apply the tokens the Figma Plugin API communicates with the plugin system to apply these values.

The integration of Tailwind CSS tokens provides support for the design and development process, the designs created through the use of the tokens are produced as accurate duplicates in the generated code outputs for Tailwind CSS. This reduces the risk of inconsistencies and the requirement of manual editing in the hand-off process.

Figure 33 - Tailwind Design Token Format

6.6. Process Design

6.6.1. Frameworks and Libraries

The plugin is designed through the use of the Figma Plugin API which acts as an interface for communication between the plugin and the Figma environment. The plugin system is developed with TypeScript which interacts safely with the Figma Plugin API, allowing the plugin to extract node data and map tokens to the Figma environment.

The user interface is programmed through HTML, CSS and JavaScript, by using standard programming languages the plugin avoids the restraints of the Figma environment and maintains a higher performance output. This technical stack is described in Figure 34, showcasing each significant technology in use for the development of the project. Support for this optimisation is provided through the esbuild bundler which enables plugin files to interact internally without lowering the quality of the file structure. The usage of ESLint, TypeScript ESLint, and the Figma ESLint plugin assist in debugging style, code and Figma Plugin API errors.

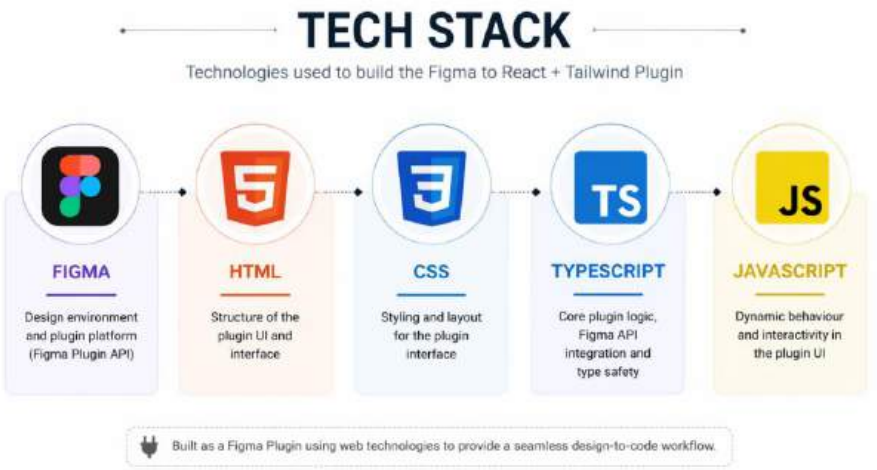


Figure 34 - Technical Stack

Overall, the combination of frameworks and libraries are efficient for the development of a design tool. The code remains efficient and structured while the communications to the Figma API are accurate and functional.

6.6.2. Core algorithms

Algorithms are prominent in the development of this plugin, especially token mapping algorithms. These structure the basis of the functionality that makes this project efficient for the design to development workflow. The process of mapping tokens extends from users requiring accurate translations of variables to visual changes.

The users interact with the interface to apply a selected token which sends the identifier to the plugin. This identifier consists of a token-value pairing where the label of the token is colour, and the value is the shade 500, as presented in Figure 35. This process searches for the corresponding token-value pair in the token registry, which is a collection of the grouped tokens. Once the token-value pair is located, the hexadecimal value of the colour is translated into an RGB value that Figma can understand. The new RGB value is further applied to the node replacing the previous property.

```
/**
 * TokenRegistry manages access to design tokens
 */
export class TokenRegistry {
  private tokenData: typeof TAILWIND_TOKENS;

  constructor(tokens: typeof TAILWIND_TOKENS) {
    this.tokenData = tokens;
  }

  private extractTokenValue(token: any): string | null {
    if (typeof token === "string") {
      return token;
    }
    if (token && typeof token === "object" && "$value" in token) {
      const tokenValue = (token as any).$value;
      return typeof tokenValue === "string" ? tokenValue : String(tokenValue);
    }
    return null;
  }
```

Figure 35 - Token Registry

The mapping of tokens is a similar process across the token registries, including spacing, typography, effects and layout. The system performs the conversion to Figma appropriate values and overwrites the node properties. Token mapping ensures that the visual output will match the acquire Tailwind CSS token to prevent inconsistencies in designs and code outputs which increases efficiency in the design to development workflow.

6.6.3. Event handling and user feedback

The plugin is designed through the communication with the Figma Plugin API, this system provides messages between the two platforms when users interact with the user interface. It demonstrates a divide between the UI layer and the Figma Environment.

The user interface consists of tokens, gradients, layouts, typography, spacing and code generation tabs, each of which contain an event listener that communicates with the user interface and the plugin system through postMessage functionality.

The plugin system receives the messages and processes them in event handlers to understand the request from the user interface. Once the message is extracted the plugin executes the request, such as applying a token or overwriting a node property. Following completion, the plugin sends a success notification to the user interface as user feedback. It proceeds to extract the newly updated node properties and refreshes the user interface content.

The user feedback is determined whether the action was successfully completed or not. This will return a success or failure notification to the user interface. They are structured as messages which perform visual feedback on the user interface.

6.6.4. Asynchronous handling

Asynchronous handling is important for the production of a plugin as it assists in the communication between the user interface and the plugin system. It handles the transferring of messages between the platforms. The plugin receives the messages from the user interface though asynchronous routing, the message is processed through asynchronous functions including, fetching the node information and the available font families, and receiving the client Storage tokens. The use of asynchronous handlers provides clear pathways for execution of the messages.

6.7. Database Design

6.7.1. Storage Model (Client Storage)

The plugin uses a client-storage system through the Figma Client Storage API which is useful as it avoids cloud-storage which provides stability of the plugin system. Client-storage allows users to store data internally on their local machine which prevents other plugins from accessing stored data.

Specifically, within the plugin gradient tokens are stored in client-storage as users are capable of creating custom gradient tokens. Client-storage consists of caches which are accessible through Figma messages, on startup of the plugin it retrieves the tokens within the cache and post them to the user interface. The process to accessing these caches is demonstrated in Figure 36, where the plugin communicates with the Figma client storage to call the token key.

```
// This function loads custom token data from persistent plugin storage.
const loadCustomTokensFromStorage = async () => {
  const colors = await figma.clientStorage.getAsync(
    CUSTOM_COLORS_STORAGE_KEY,
  );
  const gradients = await figma.clientStorage.getAsync(
    CUSTOM_GRADIENTS_STORAGE_KEY,
  );
```

Figure 36 - Colour Token Client Storage

7. Implementation

7.1. Development Methodology

The development plan for the project was based on sprints and meeting reviews. The iterative scrum cycle displayed in Figure 37 allows for consistent alterations of the plugin's features, as the milestones were broken down into smaller sections, the productivity improved (Jurado-Navas and Munoz-Luna (n.d.). Each stage of sprints focused on a particular component of the plugin's development, this supported the early-stage creation of the project.

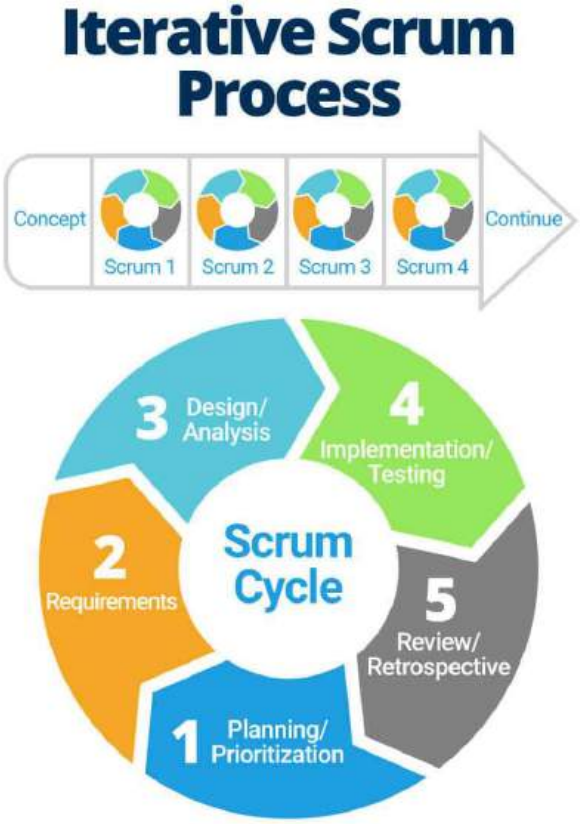


Figure 37 - Scrum Iterative Methodology

A sprint contained a focus that aligned with the primary feature missing from the plugin, such as for navigation there is the search bar or filter options. These tasks were manageable, however new directions or errors are encountered during development. This is incorporated in the finding's sections of the sprints, where developments are noted and evaluated to consider the next action to achieve completion of the focus point. Sections for decisions and actions were included to determine the most effective route to succeed in developing the component.

The generation of sprints were derived from the weekly recaps of meetings. This benefitted the development of the project as tasks were examined and dissected to assure completion is possible. Flexibility of plans were important for creating the plugin, tasks can be unachievable, however, through the failures new opportunities appeared. Such as, attempting to implement a margin algorithm was unsuccessful when focusing on manual editing of node properties, this idea evolved into the integration of Tailwind CSS tokens for spacing and layout properties.

Overall, the development process remained progressive and efficient as planning was a key feature of completing the aims and objectives of the project.

7.2. Development Environment and Tools

The plugin was developed through the use of the Figma Plugin API which connected the Figma environment with the plugin system. TypeScript was primarily used to code the plugin system as it communicates with the Figma Plugin API with ease, this allowed access to node extraction and mapping of Tailwind CSS tokens.

Managing the development of the project was important for planning future features, understanding bug fixes and tracking progression of the components. This was performed through Git Version Controlling which allowed for consistent development of the plugin and reviewing previously performed actions and commits (Perez-Riverol et al., 2016). Figure 38 demonstrates a simple view of branching and commit connections. Along with version controlling is building the application, this action was handled with esbuild, a bundler that compiled multiple files to communicate directly with the user interface and Figma Plugin API.

Figure 38 - Git Version Control

The quality of code was secured through the ESLint tool which worked with the TypeScript ESLint and Figma ESLint plugin tools. Together these tools ensured consistent and secure code when communicating between the design environment and the plugin.

The use of AI tools was used to provide support in generating ideas, drafting alternative phrases for writing, understanding unclear code, debugging errors and exploring large

open-source Git repositories. The outputs generated by the AI tools were used as guidance while the decisions were independently decided, no AI material was integrated without evaluation and modification to complete understanding.

7.1. Error Handling and Debugging

Developing an application requires testing of features and correcting errors that occur. This method of correction is called debugging, as the developer performs tests to isolate the issue before mending it. Throughout the development of this project, the debugging process incorporated examining the system during runtime, reviewing the console in the Figma environment for error codes, and using tools in the code editor.

Once the application is ready for deployment, the ESLint bundler formats the files to communicate with the Figma Plugin API, to debug the application while live a “npm watch” command is performed to analyse alterations made to the code in real-time, as displayed in Figure 39. Upon finding a bug, further investigation is carried out to determine the location and action performed that produced the error. Fixes are applied and the build tests are performed again.

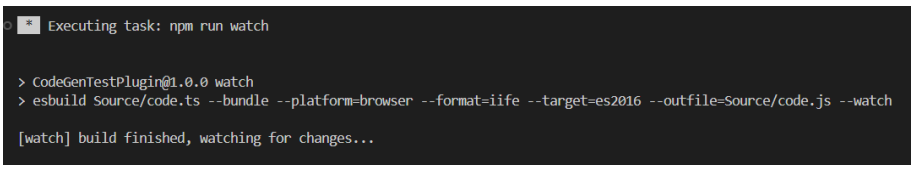


Figure 39 - ESLint Bundler Watch Debug

Tools used to handle errors consists of ESLint, TypeScript ESLint and Figma ESLint plugin. These allow the plugin to successfully run and communicate with the Figma Plugin API and the Figma environment. Logging the console was performed in early stages of development to test the results of extracting nodes to determine if properties were successfully extracted and overwritten.

Upon completion of the project bugs remain in the application, primarily performance and user interface issues that appear insignificant to the functionality and effectiveness of the workflow.

7.2. Implementation by Feature Area

The plugin consists of numerous components, each of which were developed and tested as feature areas. These implementation areas include Tailwind CSS tokens, search and filter functionality, and code generation.

The Tailwind CSS tokens were initially considered a “could” requirement type, as the understanding and knowledge of the feature area was lacking. However, upon further investigation this functionality became the primary feature of the application. The process of the tokens includes node extraction, token pairing, registry searching and overwriting property values. These tokens include colour, spacing, effects and typography as examples of these tokens are displayed in Figure 40.

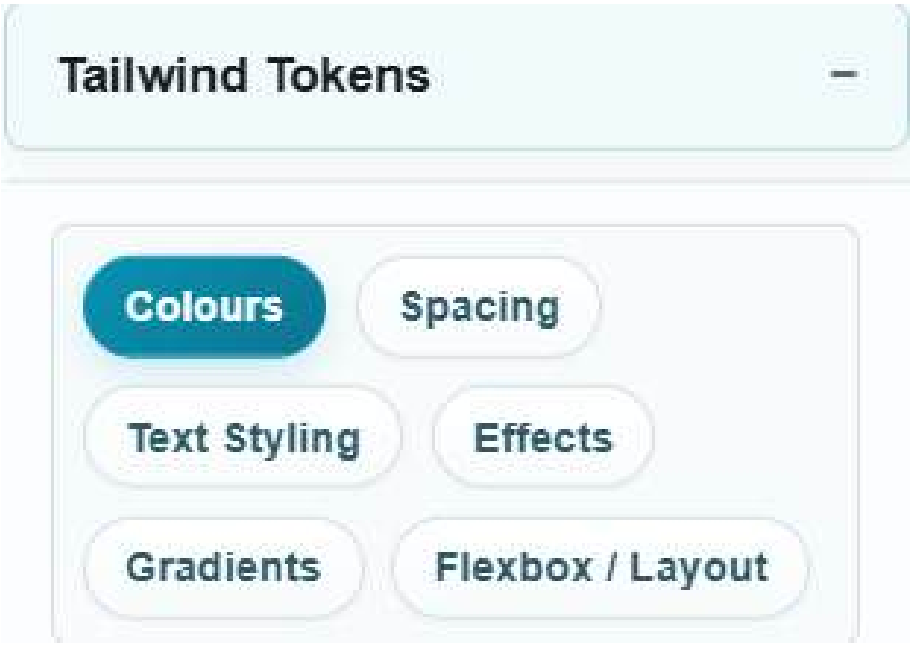


Figure 40 - Available Tailwind Token Properties

Navigation is an important feature of efficient workflows, which provides numerous options for filtering and searching functions. The plugin incorporates a use of dropdown, pill, and collapsible tabs for filtering, while the search bar remains a standard feature. Through user testing, feedback requested an increase in filter options which was later implemented as typography dropdown menu that filters the available content to match the selected filter. These filter implementations are displayed in Figure 41.

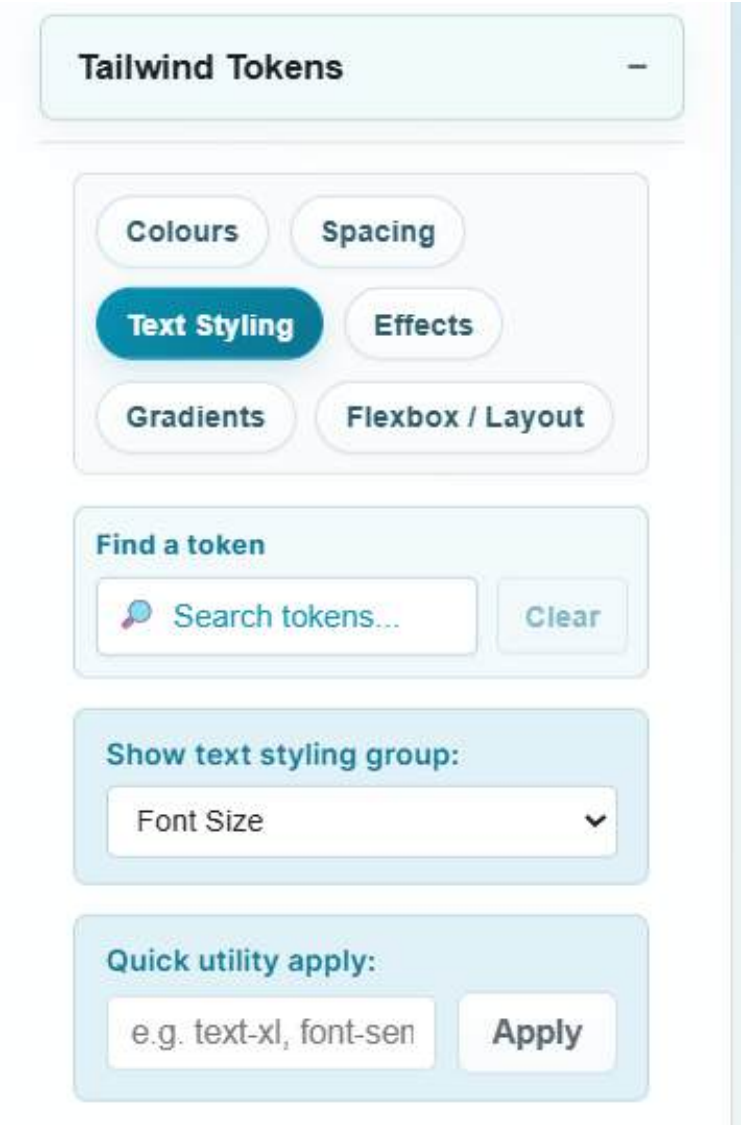


Figure 41 - Typography Filtering

Along with the development of key features such as, Tailwind tokens and navigation filters, a core feature developed from early stages of the development process that created the original scope is node extraction. Through communicating with the Figma Plugin API via asynchronous functions the plugin was capable of extracting node properties to further allow messaging to be returned to the Figma environment to edit the extracted property values. This functionality was developed into a manual editing function where users could change values displayed easily as displayed in Figure 42.

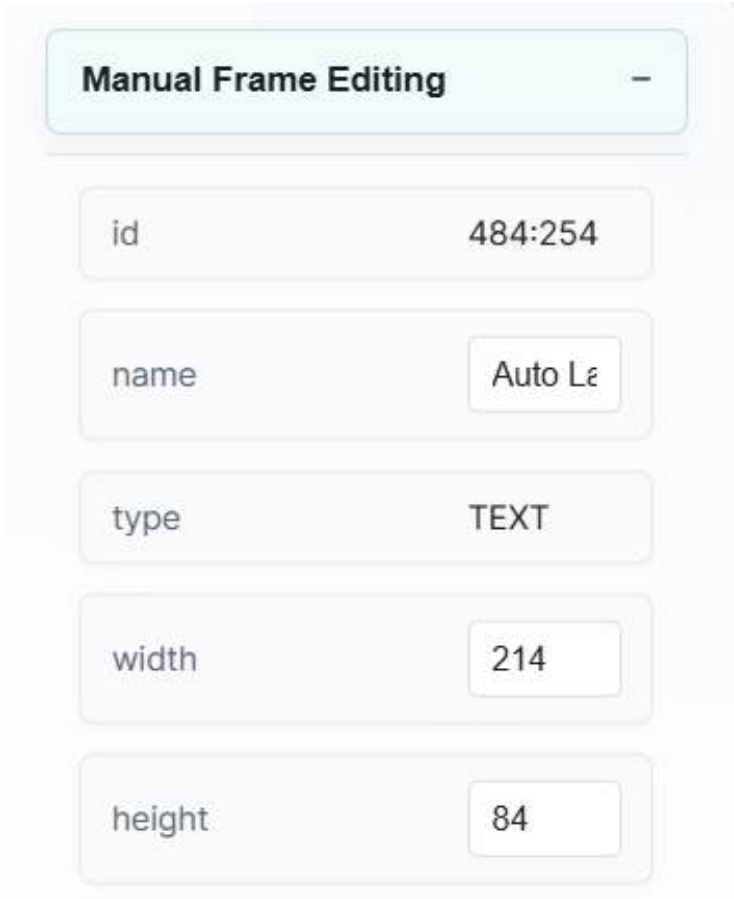


Figure 42 - Manual Editing via Node Extraction

Finally, code generation was an original requirement added to the objectives for this project. To increase the efficiency of design to development workflows an accurate code generator was required. Figure 43 represents the multiple format outputs for various language-style pairings, including HTML + CSS, HTML + Tailwind CSS, and React + Tailwind CSS. The focus on this feature is to provide identical code to the visual design created in the Figma environment. Doing so, this reduces the manual hand-off requirements and risks of inconsistencies between designs and outputted code.

Figure 43 - Code Generation Outputs

7.3. Major Technical Challenges

A primary example of a major technical challenge throughout the development process is controlling the communication between the user interface and the plugin system. Discovering the split design environments required research into the Figma Plugin API documentation to understand the methods of communication between the platforms. Figure 44 demonstrates the formatting of creating the messages sent between the plugin and the user interface displayed on the Figma environment.

```
    figma.ui.postMessage({
      type: "plugin-notification",
      level,
      message: noticeMessage,
    });
};
```

Figure 44 - Example Message Format

Each interaction performed by a user requires a message, this is segmented into message handlers in the user interface and plugin system, as both ends are required to receive and send messages with tasks to be executed. This was the first iteration of communication in the system, individual message handlers. Later these handlers evolved into routing systems, where messages contained a message type, and content. This provided distinct routing of messages to the designated channel allowing communication between the user interface and plugin system to remain consistent and fast.

Communication was not the only challenge that occurred. In early stages of development, the mapping of values appeared to output inconsistent values. To evaluate this token registries were implemented which consisted of token-value pairings to ensure accurate outputs were produced. This fix allowed designs to produce accurate code outputs as the applied token value was directly translated into the generated code output. Figure 45 displays the process of calling the token-value pair for a colour token, as it requests the colour token group and shade values.

```
getToken(category: string, colorName: string, shade: string): string | null {
  // Navigate to the color group: tokens[category][colorName]
  const categoryData = (this.tokenData as any)[category];
  if (!categoryData) {
    return null;
  }
```

Figure 45 - Example Token-Value Pairing

Originally, an objective set was the creation of a margin algorithm, the initial process was to create a custom token for margin that users can apply when designing. Through testing and evaluation of the Figma environment it was deemed out of scope for the current timeframe provided. Many factors contributed to this decision as the Figma design system does not include margin as an option available, therefore, extraction of the node property was not recognised. Figure 46 displays the code temporarily implemented whilst developing the margin algorithm, it seemed potentially viable apart from the styling of Figma frames, as they use absolute positioning.

```
+ function computeMargins(node: SceneNode, parent: SceneNode) {
+   // For auto-layout parents, spacing is modeled via padding and gap.
+   if ("layoutMode" in parent && (parent as FrameNode).layoutMode !== "NONE") {
+     return { top: 0, left: 0, right: 0, bottom: 0 };
+   }
+   if (parent.type === "FRAME") {
+     return computeMarginsWithFrame(node, parent);
+   } else {
+     return computeMarginsNotFrame(node, parent);
+   }
+ }
+
+ function computeMarginsWithFrame(node: SceneNode, parent: SceneNode) {
+   const top = (node as any).y ?? 0;
+   const left = (node as any).x ?? 0;
+   const right = (parent as any).width - (node as any).width - left;
+   const bottom = (parent as any).height - (node as any).height - top;
+   return { top, left, right, bottom };
+ }
+
+ function computeMarginsNotFrame(node: SceneNode, parent: SceneNode) {
+   const top = ((node as any).y ?? 0) - ((parent as any).y ?? 0);
+   const left = ((node as any).x ?? 0) - ((parent as any).x ?? 0);
+   const right = (parent as any).width - (left + (node as any).width);
+   const bottom = (parent as any).height - (top + (node as any).height);
+   return { top, left, right, bottom };
+ }
```

Figure 46 - Margin Uncompleted Code

Finally, an error occurred with accessing the font families that were supposedly available. Through examining the Figma environment design panel, hundreds of font families appeared to be applicable. However, in the plugin only primary font families such as sans, and inter were available. This was resolved by requesting access to the Figma environment font families rather than manually integrating each set, as shown in Figure 47.

```
const getAvailableFontsCached = async (): Promise<Font[]> => {
  if (!availableFontsCache) {
    availableFontsCache = await figma.listAvailableFontsAsync();
  }
  return availableFontsCache;
};
```

Figure 47 - Accessing Figma Fonts

7.4. Code quality

The quality of the code was important for producing an effective workflow for a project focused on design to development pipelines (Ammattikorkeakoulu, 2016). The formatting of the code is split into files for utilities, tokens, formatters, converters and code generation, this divide provides a clear navigation of the code for management and editing, as well as understanding the connection between functions and helpers.

The naming convention exhibited in Figure 48 is camelCase, this was primarily followed for the creation of files in the project. The method describes assigning the first character of the file lowercase and the following character of each new word uppercase. The benefits of camelCase are easier navigation of files and differentiating words within the file names.

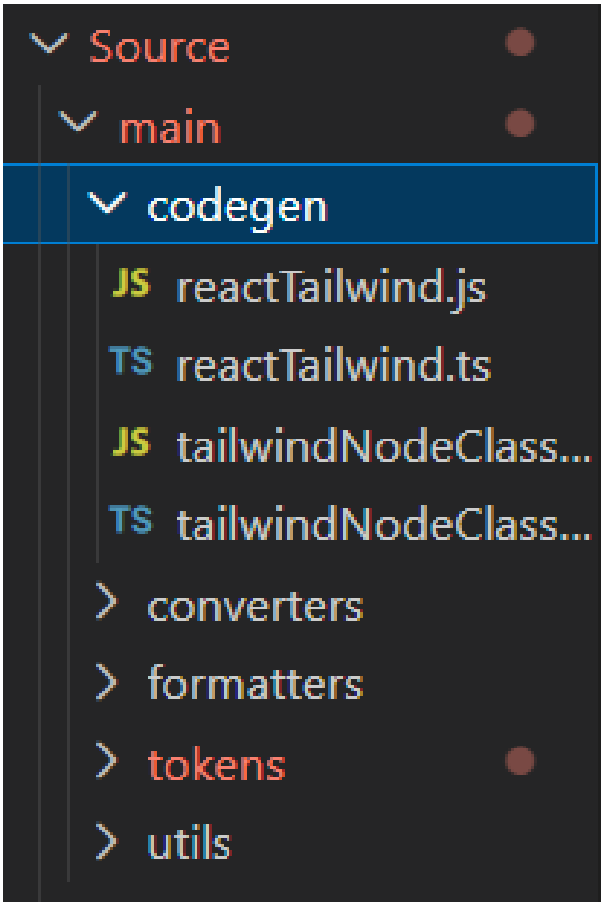


Figure 48 - File Formatting

Developing a project requires multiple files, functions, and algorithms, which are difficult to understand at a glance. Figure 49 demonstrates the use of commenting in the development process, including how commit messages are formatted to easily understand the functionality integrated, these messages are stored in a commit history where the code stored can be reviewed by team members.

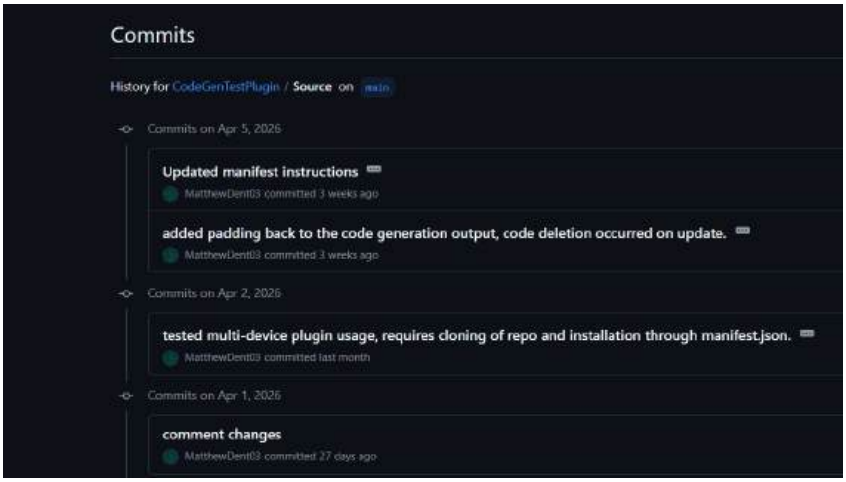


Figure 49 - Commit History

8. Testing and Evaluation

8.1. Test Plan

Testing is a defining stage of the project's performance as the functionality, user interface and system are evaluated thoroughly by multiple users. The selected strategy for testing largely included user testing, where a use case task is provided and criteria is assigned for success and failure. The user testing conducted primarily focused on qualitative results than quantitative, this direction was chosen as understanding the prominent errors through a smaller testing group provided a deeper insight into the feedback returned. Quantitative data may provide a less thorough review of smaller focus points that are relevant to this development process. However, quantitative data remains necessary for collecting a general perspective of the application and the workflow it supports.

Apart from user testing, an observational test was conducted, which focuses on the general usage of a user whilst working in the environment without contribution by the examiner. The combination of these testing strategies provides qualitative and quantitative data relating to the overall features of the application (Nielsen, 2020).

A task provided for testing must have criteria in place for success and failure, these act as guidelines to ensure accurate responses are collected from testing. The task assigned consisted of a testing environment, in which users were informed to replicate the design outputs through utilising the plugin's functionality. Criteria were disclosed to include successful outcomes and failures.

The success criteria consisted of the application of tokens correctly, selection of a Figma frame, accuracy when recreating the visual design, and ease of use when navigating the user interface. However, criteria for failure were revealed to include the inability to complete tasks without assistance, issues when navigating the user interface, incorrect application of tokens, and distant visual replication of the design.

As testing provides feedback on areas of improvement, user groups are essential for gathering directed critiques. Initial survey tests were conducted to create a basis for the direction of the project's requirements, these users comprised of frontend developers, backend developers, full-stack developers, students and UX/UI designers. This set of testers provided a variety of inputs that were incorporated into the requirements to generate a plugin that addressed all areas of feedback. These participants were later included in the user testing period.

8.2. Types of Testing

Testing was deployed as user testing and observational testing. User testing delivers a range of results as testers of different experience levels return feedback on the application (Husseini, 2023). Users were provided with a use case task, criteria for success and failure, a testing environment and a tutorial guide. Along with user testing came observational testing, a method that consists of examining the user as they navigate the system and perceive errors and flaws of the application and methods taken to execute the task.

Apart from testing the usage of the application, the functionality was examined continuously throughout the development process. This ensured that features such as navigation, search and filtering, token application and code generation were performing correctly and yielding accurate results.

Finally, regression testing was performed during the implementation of important features, in addition to applying alterations to the system based on user feedback from testing (Li, 2019). Regression testing ensures that components of the plugin that existed previously continue to function as intended after new changes are completed.

8.3. User Feedback Results

Through evaluating the results and feedback of the testing and survey, insight was gained into the areas of the application that were successful, and areas that failed. The results shown in Figure 50 and Figure 51 describe how users had reported that 80% partially completed the task due to time constraints, while 20% fully completed the task. The estimated time users took performing the design recreation was within 10 to 20 minutes.

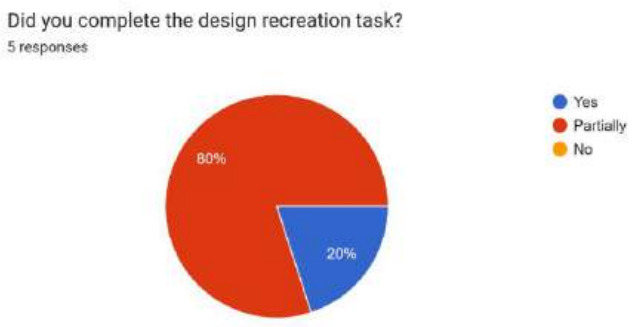


Figure 50 - User Testing Survey Feedback Completion Rate

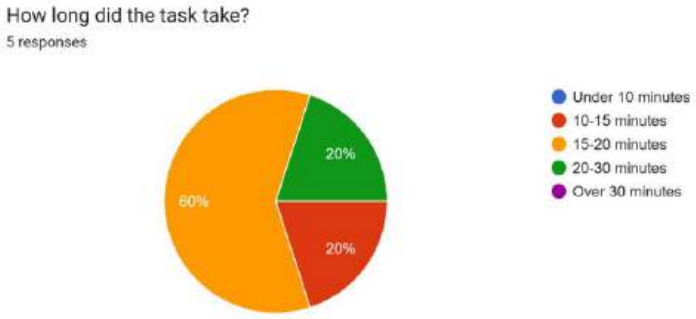


Figure 51 - User Testing Survey Feedback Task Time

Through observing users perform the tasks there were visual identifiers of functionality and user interface failings. These consisted of unfamiliarity with the user interface, as testers continued to find difficulties in navigating to the correct spacing and layout tokens, as reviewed in Figure 52.

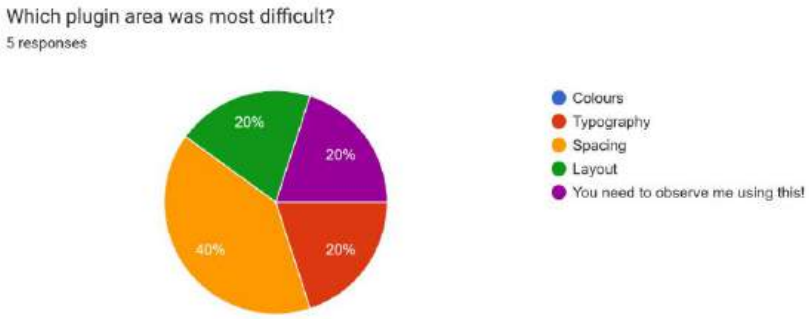


Figure 52 - User Testing Survey Feedback Difficulties

Additionally, the user review shown in Figure 53 describes that there were issues with the content in search boxes persisting after application of tokens and navigating to alternative tabs. This bug caused confusion with users as they attempted to proceed with the next action, except the previous search was preventing tokens from displaying.

Figure 53 - User Testing Survey Feedback Frustrations

Evidence of these findings are displayed through the collected results of a google forms survey, observational notes and user feedback inputs.

8.4. User Feedback

From results in the survey users described the usability of navigation and layout to be largely successful, however, errors were encountered with the size of the tabs, as it prevented users from accessing the full functionality of the plugin at times. User satisfaction and further usage of the plugin was received positively overall, as displayed in Figures 54 and 55.

Along with navigation, users responded on the area of difficulty during testing, results concluded spacing was producing consistent complications as the width and height tokens were separated between the layout and spacing tabs.

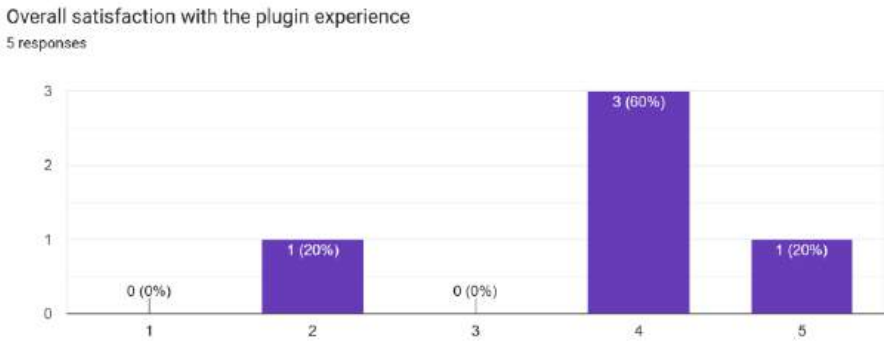


Figure 54 - User Testing Survey Feedback Satisfaction

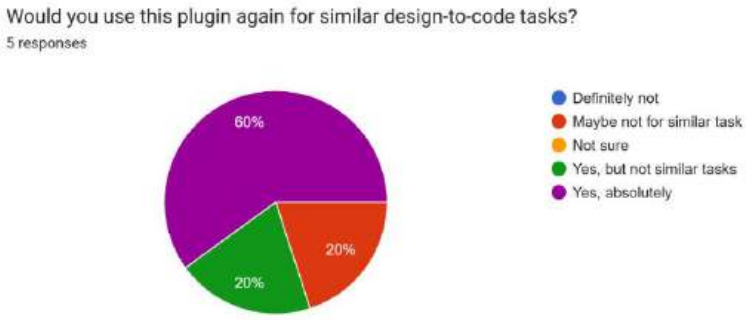


Figure 55 - User Testing Survey Feedback Usage of Plugin

Finally, users responded to survey questions asking for areas of improvement and requested features, as shown in Figure 56 and Figure 57. The findings show a need for visual notifications when successfully applying tokens to a frame. Users mentioned the inefficiencies of the typography tokens tab, as it remained unfilterable unlike each other section. And upon improvements recommended by users, experienced developers desired a fast method to apply tokens directly to the frame through manual inputs.

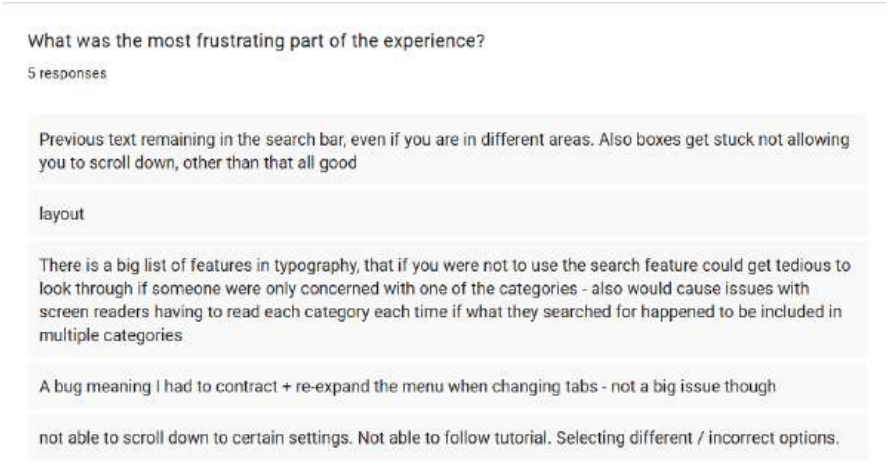


Figure 56 - User Testing Feedback Frustrations

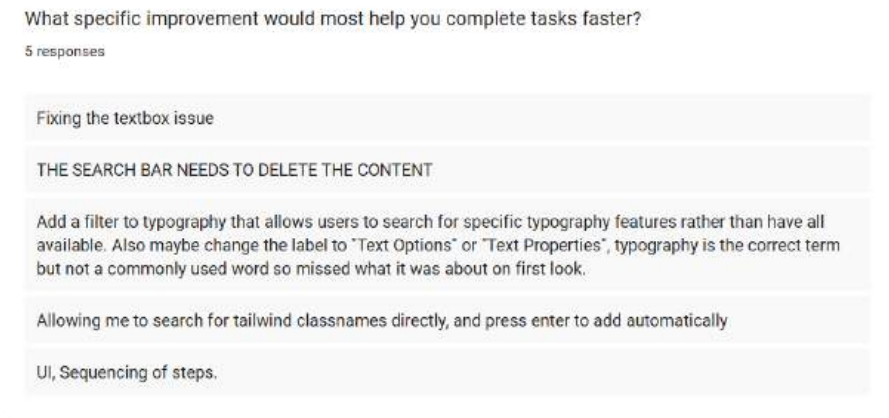


Figure 57 - User Testing Feedback Improvements

Following the compiling and evaluation of feedback results, edits began to be implemented into the project to adhere to these findings. In doing so, a button to clear the search bar was created, as well as automatic removal of search items when navigating to other tabs.

Along with search fixes, the user interface refresh script was updated to expand the tab to the full content size. Additionally, the typography tokens were supported with filter options to display the content of the selected token group. Finally, visual notifications were implemented as a popup in the user interface when applying a token successfully.

8.5. Testing Feedback Summary

Following the completion of the testing period, results and feedback conclude that the plugin functions successfully as it meets the success criteria originally established. Users successfully completed the majority of the use case task and provided positive feedback on the functionality and effectiveness of the user interface (Nielsen, 2020). The system proved to perform at an efficient standard, as seamlessly selected and edited frames with the use of tokens and produced similar design replicas.

Furthermore, user testing and survey feedback revealed difficulties with the user interface and similarity with pre-existing design environments. Errors were discovered with tab sizing and search functionality, as well as positioning of specific tokens were unfamiliar to alternative design panels, including the Figma environment.

Upon comparison of the criteria constructed to guide the development of the plugin, success has been proven in the production of a Tailwind CSS token system, accurate design to code generation, and improving the workflow efficiency in the design to development pipeline.

9. Project Management

9.1. Methodology and Project Sprints

The project management were conducted as sprints, this provided an iterative process where development was consistent and implementations were evaluated frequently (Jurado-Navas and Munoz-Luna, n.d.). This method of management allows components to be focused on per sprint while errors are resolved regularly. Sprints contained multiple sections of information, such as focus, key findings, decisions and actions. These headings defined the point of development, the relevant information for proceeding, errors or concerns discovered through development and the actions performed. This system allowed for progressive design to occur as components were deployed and issues were fixed.

Following the Scrum methodology of sprints, the planning & prioritisation sprints included sprints 1 and 2, which primarily focused on discovering the limitations of the Figma Plugin API and the Figma environment, as well as the process to extract nodes. These sprints are shown in Figure 58.

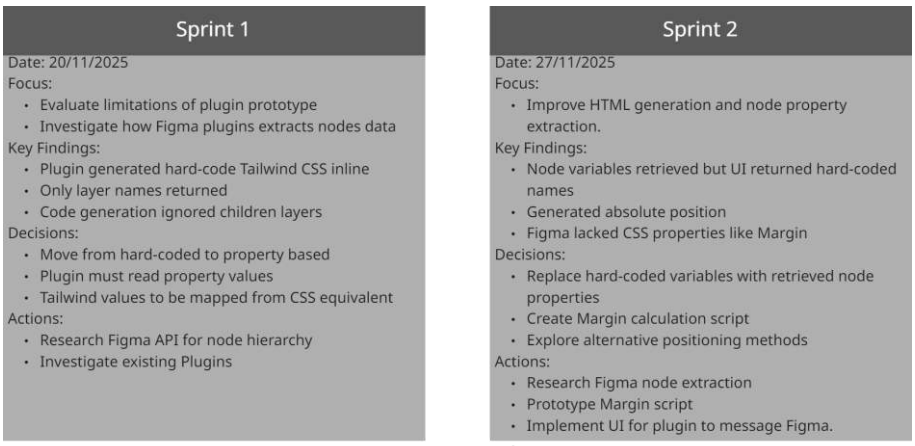


Figure 58 - Planning Sprints 1&2

The next stage of Scrum involves the Requirements & Exploration which includes sprints 3 and 4 as shown in Figure 59. This stage of development explores the technical requirements of the plugin including code generation and mapping CSS to Tailwind CSS.

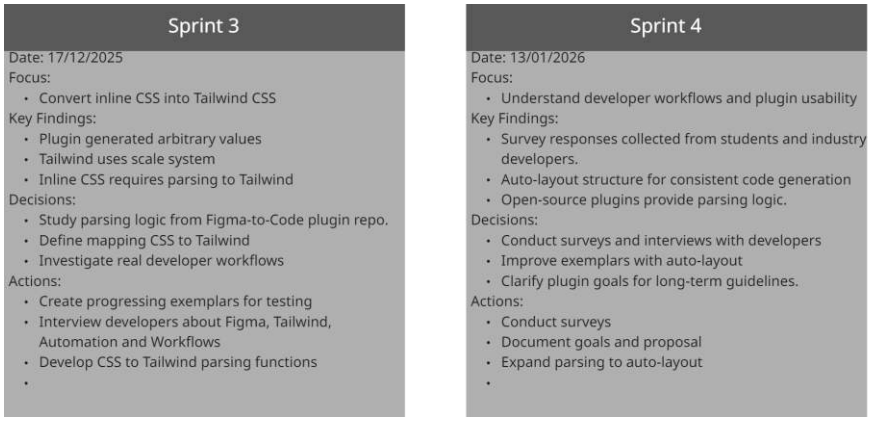


Figure 59 - Requirements Sprints 3&4

The Design phase is prominent in this development process as the incorporation of Tailwind tokens is introduced, Figure 60 displays sprints 5 through 7 where the plugin interface is redesigned to include the use of Tailwind tokens. Research of the W3C token format standards, the token code was reiterated to provide appropriate key-value pairing.

Figure 60 - Design Sprints 5&6&7

Along with the design process is the implementation period, where the functionality of the plugin is further expanded, redeveloped and enhanced. Figure 61 demonstrates how sprints 8 and 9 detail the implementations proceeded. Including, search and filter functionality, as well as improving code generation output accuracy.

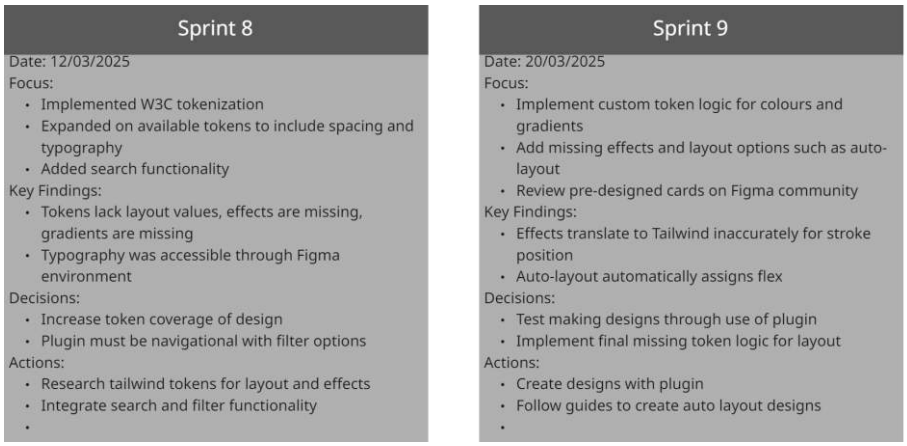


Figure 61 - Implementation Sprints 8&9

Finally, Figure 62 showcases the Testing and Iteration stages of the Scrum methodology. This is the process of testing the application through user testing, observational testing and then implementing the feedback into the iteration process. This stage improves the overall functionality and efficiency of the plugin.

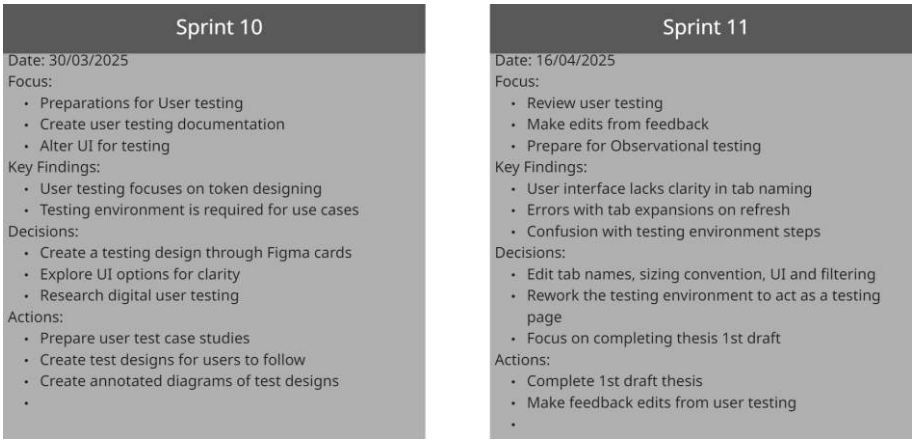


Figure 62 - Testing and Iteration Sprints 10&11

In addition to sprints, weekly meetings were carried out to discuss the newly implemented features of the plugin and future improvements. Through the assessment of the plugin state, insight was gained into the potential enhancements available to incorporate into the design plan.

9.2. Planning and Tracking

The planning and tracking of tasks for the project's progression was done through Git commits and notarising instances of improvements or errors. Sprints were used for general milestones and guidelines for what steps are needed for implementing components. The Git commits displayed in Figure 63 are examples of how commits provided updated information on the state of the plugin code and previously integrated features and bug fixes.

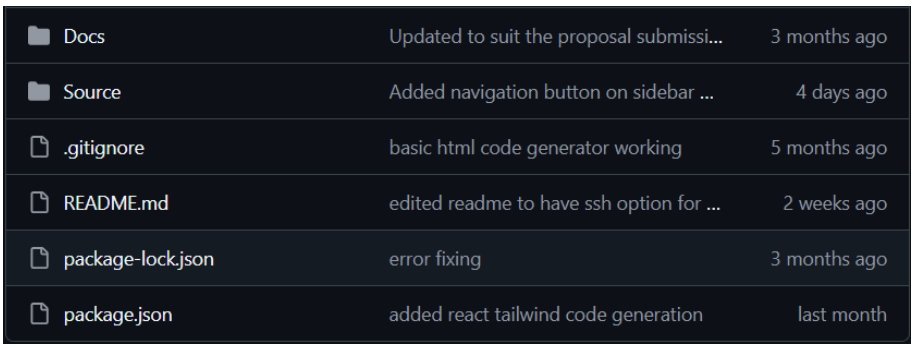


Figure 63 - Git Commits Comments

Apart from sprints and Git commits were meetings that occurred where functionality was reviewed and future improvements were coordinated. Finally, personal status notes were created in a simple notepad, this allowed for easy tracking of whether current objectives were completed or in progress.

9.3. Risk Mitigation

Over the course of developing the project risks were encountered, these included knowledge constraints on project requirements, timeframe management and limitations of the Figma Plugin API. However, a primary risk factor was the Figma environment as it restricted the use of frameworks, coding languages and communication between the user interface and the plugin system.

Through extensive research of open-source Git repositories of Figma plugins, bundlers were discovered mid-way through the development of the project. This produced information on both the Turbo and ESLint bundlers which allow plugins to communicate with the user interface in the Figma environment through multiple code files. The choice to use ESLint over Turbo was decided by their benefits, as Turbo focuses primarily on speed for building, ESLint prioritises quality of code which is essential for enhancing the design to development workflow.

Finally, time management played an enormous role in risk mitigation, during the creation of requirements for the project, what was expected as an outcome appeared unrealistic due to time constraints. However, gradually these requirements were revisited as the project began to grow in possibilities and understanding of the system in use. This guided to development of a second iteration of requirements.

9.4. Reflection on process effectiveness

The development process was significantly effective as the project progressed consistently throughout the timeframe. Although errors and setbacks were encountered, it provided insight into the benefits of an iterative approach. As requirements grew during the expansion of the project, the scope was adjusted to match these alterations. A feature which worked well during the development process was segmenting the functionality into manageable sprints. This allowed for gradual iteration of key features that provided a basis for the plugin's functionality.

However, there were features that did not work as smoothly. Although the sprints provided guidance for the tasks, they did not reveal the difficulties of achieving their completion. Another feature that required alteration was the implementation of tokenization. There were numerous errors and concerns, specifically with lack of

knowledge and timeframe management. This phase of design required immense research and development to introduce a token-based system into the plugin. However, through trial and error the system provides the basis of the aim and objective to reducing the inefficiencies of design to development workflows.

10. Conclusion and Future Improvements

10.1. Summary of Project Outcomes

Upon completion of this project the design and development process is believed to be a success, as the system produced influences the design to development workflow in a beneficial manner. This was achieved through the integration of a Tailwind CSS tokenization system, as well as a multi-format automated code generator. Together these primary features create a system that permits users to build visual designs through the use of Tailwind CSS tokens, as well as altering previously designed components and generating a variety of code exports in various formats.

The completed plugin has several points of functionality that work cohesively together to provide a service which generates almost identical code outputs of visual designs, while maintaining the code quality and fidelity of the design. This is preserved through the token-based system that ensures applied values are translated into code outputs accurately, which reduces the risk of inconsistencies between visual designs and code generation, in addition to the decrease in manual requirements throughout the hand-off process.

10.2. Did the project meet the Aims & Objectives?

The project achieved completion of the aims and objectives, these aims included improving the efficiency and consistency of workflows in the design to development pipeline. This task contained multiple objectives, including the development of an automated code generator, a token-based design system, and an efficient, user-friendly interface which must be capable of communicating with the Figma Plugin API.

These objectives were successfully fulfilled throughout the design process. Although original requirements were reconsidered and altered, the finalised product exceeds the expectations of those requirements.

10.3. Critical Reflection

10.3.1. Methodology

Through the use of sprint methodology an understanding was developed that it provides guidelines for future tasks and implementation, keeping track of issues and bugs. However, the tasks appear more manageable when described in a sprint than when undertaking the task itself. This caused concern with time management of the project resulting in setbacks and forced more time to be allocated to completing the task.

10.3.2. Personal & Technical

During the development process of the project technical issues became apparent, specifically the implementation of tokenization. Tailwind CSS tokens provided an enormous opportunity to reduce the inconsistencies with design to development workflows, however, the process of integrating tokens was challenging. The task involved creating a token registry that paired the token with a value and provided a search function for locating the matching tokens.

10.3.3. Professional learning

This project challenged the technical and methodological aspects of developing a plugin. Additionally, professional learning was tested as well. The development opposed difficulties on planning and organisation, including documentation, Git commits and time management. These obstacles were set to improve the understanding of professional workflows and how to navigate them.

10.4. Limitations

Although the project itself is a success, there remains limitations in a technical aspect, this includes the system code and the UX/UI. Currently the code generator produces multiple formatted outputs, however, these are limited to HTML and React for frameworks.

As well as framework shortage, the generated outputs are readable and functioning, except the code appears repetitive and the lack of AI integration prevents element recognition to produce specified component outputs.

Finally, the user interface established does not follow the design layout of the Figma environment. Through observational and user testing this became evident as users failed to navigate tokens in an efficient time. The feedback provided described the familiarity with design panels and where the plugin was expected to contain functionality.

10.5. Future Improvements

10.5.1. Short-term improvements

In short-term the project intends to implement improvements based on user feedback through user and observational testing. These include the enhancement and organisation of the testing environment as steps were found to be unclear. This provides new users an example design to practice the plugin usage.

Additionally, advancements on accuracy of the user interface panel are required to increase workflow efficiency. Feedback suggested that the layout of specific tokens was preferred in alternative tabs and various visual layouts.

Finally, the implementation of refined search functionality. Currently the project provides filter dropdowns and per page searching, however, if a user requests to search across the tabs there is no resource available. This feature can increase productivity while using the plugin.

10.5.2. Long-term development

For long-term the project aims to expand available code formats for automated generation are required to improve usability for developers and increase the user base. These formats currently include, HTML, CSS, React, and Tailwind CSS, however, there is a large collection of frameworks that are commonly used for development that are not integrated in this project.

To continue to develop modern applications the integration of AI tools is important, as AI is becoming a key feature of modern technology. The use of AI in the plugin would assist in creating base designs, apply stylings and improve code generation. This implementation would require close observation to ensure quality of code is not being sacrificed at the cost of AI integration.

Finally, the use of client-storage is only beneficial on small scale projects, this introduces the necessity for cloud-storage as users become capable of collaborating on the same designs and sharing tokens. In doing so, productivity within teams will improve and workflows become more efficient.

10.6. Final concluding statement

This project showcases the integration of significant components that create this plugin. From extracting node properties, to implementing tokenization, to generating code outputs. The plugin consists of various features that cohesively improve the efficiency of design to development workflows by translating high-fidelity designs into development ready generated code. However, flaws are evident through testing of the plugin. Although the user interface is functional, it lacks visual aesthetics, extended framework accessibility and tool implementation. In addition to these cons, the user testing was conducted in a small testing group which prohibited further understanding of current errors and future improvements. Future improvements will focus on developing a targeted UX/UI to improve the visual design and expand on the functionality provided by the plugin through further observational and user testing. As the previous testing prioritised qualitative over quantitative testing. These alterations will support the goal to reduce inefficiencies in the design to development pipeline.

11. References

A code generator for Component Oriented Programming framework. (2011, September 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/6079314

A comparative review of AI techniques for automated code generation in software development: advancements, challenges, and future directions. (2024). Questa Soft. https://www.ceeol.com/search/article-detail?id=1223185

Ammattikorkeakoulu, M. (2016). *Optimizing web development workflow*. Theseus. https://www.theseus.fi/handle/10024/118611

Ammattikorkeakoulu, M. (2018). *Tools for code quality in front-end software development*. Theseus. https://www.theseus.fi/handle/10024/143272

Automatic generation of User Interface(UI) with the help of AI technologies - WebThesis. (n.d.). https://webthesis.biblio.polito.it/33238/?template=default

Chen, Y., & Chen, L. (2025, November 6). *PSD2Code: Automated Front-End Code Generation from Design Files via Multimodal Large Language Models*. arXiv.org. https://arxiv.org/abs/2511.04012

Coyette, A., Kieffer, S., & Vanderdonckt, J. (2007). Multi-fidelity prototyping of user interfaces. In *Lecture notes in computer science* (pp. 150–164). https://doi.org/10.1007/978-3-540-74796-3_16

Design Tokens Format Module 2025.10. (2026, April 10). https://www.designtokens.org/tr/drafts/format/

Designing and Prototyping Interfaces with Figma. (n.d.). Google Books. https://books.google.ie/books?hl=en&lr=&id=7zvrEAAAQBAJ&oi=fnd&pg=PP1&dq=figma+plugin+constraints&ots=Opov7hB2FT&sig=9n-3LVwrt3UumjCM8QPeVZxB1wM&redir_esc=y#v=onepage&q=figma%20plugin%20constraints&f=false

ECAI 2002. (n.d.). Google Books. https://books.google.ie/books?hl=en&lr=&id=5ZuuF0ogxU4C&oi=fnd&pg=PA417&dq=ai+in+software+debugging&ots=e2nocnW0KM&sig=xR7lwscJTJZnUaVoskaaIcxxkl8&redir_esc=y#v=onepage&q=ai%20in%20software%20debugging&f=false

Eren, P. E. (2026, January 20). *Assessment of AI-generated front-end code quality: a comparative study*. https://open.metu.edu.tr/handle/11511/118453

Figma Make: Create with AI-Powered Design Tools. (n.d.). Figma. https://www.figma.com/make/?gclsrc=aw.ds&utm_source=google&utm_medium=cpc&utm_campaign=22570738035&utm_term=figma&utm_content=799714320462&utm_

adgroup=182267199800&gad_source=1&gad_campaignid=22570738035&gbraid=0AAAAACTf0kPxTdoAqXN3eA1QDDsfyBf9-&gclid=CjwKCAjw46HPBhAMEiwASZpLRHZmpZFY2m-qdVso_QY_NZa2bkXtnT2p_P3pkRE9crEbupiQDi9ujBoC7ZQQAvD_BwE

Ganesaraja, R., N, S., Ap, S., Rathinasamy, K., Amancharla, C., Das, R., Panse, S. D., Batwe, A., Vijayan, D., Ashok, V., P, T. A., Rao, K. J., Olivero, A., Roshan, Manthena, R. R., A, A. Y. S., Tripathi, H., Selvaraj, S., Chin, V., . . . Vijayakumar, S. (2025, December 5). *Beyond Prototyping: Autonomous, Enterprise-Grade Frontend Development from Pixel to Production via a Specialized Multi-Agent Framework*. arXiv.org. https://arxiv.org/abs/2512.06046

Gui, Y., Zhang, J., Wang, Y., Ma, T., Wan, Y., He, S., Chen, D., Zhao, Z., Jiang, W., Shi, X., Jin, H., & Yu, P. S. (2026, April 15). *FIGMA2Code: Automating Multimodal design to code in the Wild*. arXiv.org. https://arxiv.org/abs/2604.13648

Hapuli, J. (2025). *Figman low-code-liitännäiset*. Theseus. https://www.theseus.fi/handle/10024/902384

Huang, T. (2024, November 22). *FEAD: FIGMA-Enhanced App Design Framework for improving UI/UX in Educational app development*. arXiv.org. https://arxiv.org/abs/2412.06793

Husseini, K. (2023). *Testing front-end architecture*. Theseus. https://www.theseus.fi/handle/10024/790644

Introduction | Developer Docs. (n.d.). https://developers.figma.com/docs/plugins/

Israelian, O. (2025). *Multiplatform UX/UI Design Portfolio: analysis and evaluation*. Theseus. https://www.theseus.fi/handle/10024/904616

Jaspan, C. N. C. (n.d.). *Proper Plugin Protocols - ProQuest*. https://www.proquest.com/openview/e3528c44297183d4c3f1a2ee9d7b687d/1?pq-origsite=gscholar&cbl=18750

Jurado-Navas, A., & Munoz-Luna, R. (n.d.). *ScRum Methodology in Higher Education: Innovation in teaching, learning and Assessment*. https://eric.ed.gov/?id=EJ1160130

Kanapathipillai, I., & Priyankara, O. (2026, January 15). *CoGen: Creation of reusable UI components in Figma via textual commands*. arXiv.org. https://arxiv.org/abs/2601.10536

Li, Y. (2019). *Front-end testing : an important part of quality assurance in Front-end development*. Theseus. https://www.theseus.fi/handle/10024/265272

Making sure you're not a bot! (n.d.). https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1680045&dswid=8710

Miller, D. (2021). The best practice of teach computer science students to use paper prototyping. *International Journal of Technology Innovation and Management (IJTIM)*, *1*(2), 42–63. https://doi.org/10.54489/ijtim.v1i2.17

Nielsen, J. (2020, July 23). *Assessing the usability of a user interface standard*. Nielsen Norman Group. https://www.nngroup.com/articles/assessing-usability-user-interface-standard/

Odeh, A. (2024). Exploring AI innovations in automated Software source code generation: progress, hurdles, and future paths. *Informatica*, *48*(8). https://doi.org/10.31449/inf.v48i8.5291

Pan, S., Wang, L., Zhang, T., Xing, Z., Zhao, Y., Lu, Q., & Sun, X. (2024, September 20). *"I don't use AI for everything": Exploring utility, attitude, and responsibility of AI-empowered tools in software development*. arXiv.org. https://arxiv.org/abs/2409.13343

Perez-Riverol, Y., Gatto, L., Wang, R., Sachsenberg, T., Uszkoreit, J., Da Veiga Leprevost, F., Fufezan, C., Ternent, T., Eglen, S. J., Katz, D. S., Pollard, T. J., Konovalov, A., Flight, R. M., Blin, K., & Vizcaíno, J. A. (2016). Ten simple rules for taking advantage of Git and GitHub. *PLoS Computational Biology*, *12*(7), e1004947. https://doi.org/10.1371/journal.pcbi.1004947

Piccolo, S. R., Ence, Z. E., Anderson, E. C., Chang, J. T., & Bild, A. H. (2021). Simplifying the development of portable, scalable, and reproducible workflows. *eLife*, *10*. https://doi.org/10.7554/elife.71069

Plugins for Code Generation | Developer Docs. (n.d.). https://developers.figma.com/docs/plugins/codegen-plugins/#getting-started

Putra, Z. F. F., Ajie, H., & Safitri, I. A. (2021). Designing A User Interface and User Experience from Piring Makanku Application by Using Figma Application for Teens. *Putra | IJISTECH (International Journal of Information System and Technology)*. https://doi.org/10.30645/ijistech.v5i3.145

Saari, T. (2019). *Creating a design token library for ABB's CommonUX design system*. Theseus. https://www.theseus.fi/handle/10024/227894

Sparling, M. (2000). Lessons learned through six years of component-based development. *Communications of the ACM*, *43*(10), 47–53. https://doi.org/10.1145/352183.352202

Stewart, I., & Shi, Y. (2026a). *MOSCOW analysis in user engagement in a local government bus station project*. https://doi.org/10.4135/9798348853921

Stewart, I., & Shi, Y. (2026b). *MOSCOW analysis in user engagement in a local government bus station project*. https://doi.org/10.4135/9798348853921

Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. (n.d.). Tailwind CSS. https://tailwindcss.com/

The Designer's guide to Figma. (n.d.). Google Books. https://books.google.ie/books?hl=en&lr=&id=v8zDEAAAQBAJ&oi=fnd&pg=PA1&dq=figma+documentation&ots=KM9D8bPIUn&sig=c5TWHWMmtWNQXcQMy-SokYhhec0&redir_esc=y#v=onepage&q=figma%20documentation&f=false

Xiao, J., Qin, J., Li, S., Lam, M. H., Wan, Y., Huang, J., Huo, Y., & Lyu, M. R. (2026, February 22). *ComUICoder: Component-based reusable UI code generation for complex websites via semantic segmentation and element-wise feedback*. arXiv.org. https://arxiv.org/abs/2602.19276

Zhu, L., Feng, Y., Zhu, H., Wang, S., Zhu, M., Yu, C., Zhang, Y., Xu, D., Zhao, D., Feng, Y., & Chen, W. (2025). FIGMA2Code: Automatic Code generation method for FIGMA design drafts. *Journal of Computer-Aided Design & Computer Graphics*, *37*(2), 321–329. https://doi.org/10.3724/sp.j.1089.2023-00336

12. Appendices

12.1. Appendix A

12.1.1. Detailed Test Cases

A.1.1 User Testing Overview

The testing conducted included user testing and observational testing. These methods ensure feedback for both the user interface and the plugin system. Users were tasked with creating designs based on annotated diagrams through the use of the plugin design environment. They were provided a consent form, a guided tutorial and a use case task document.

A.1.2 Signed User Consent Forms

A.1.2.1 Lili Consent Form

[redacted]

iadt
DUN LAOGHAIRE

Figure 64 - A.1.2.1 - Lili Consent Form

A.1.2.2 Ethan Consent Form

Project Title: Figma-to-Code Plugin

Principal Investigators: Matthew Dent

You are invited to participate in a user test for a Figma-to-Code plugin, developed to assist users in converting design components into functional code more efficiently. The purpose of this study is to evaluate the plugin's usability, performance, and overall user experience.

Your participation involves completing a set of design tasks using the plugin and providing feedback on your experience using the plugin during the completion of the task.

Participant Information:

- **Participant Name: Ethan Holland**

- **Date of Birth** *(DD:MM:YYYY)*:**26/01/2001**

- **Gender** (mark 'X'):
  - ○ **Female**
  - **X** **Male**
  - ○ **Non-Binary**
  - ○ **Prefer Not to Say**

- **Contact Information** *(optional)*:_____

Consent to Participate:

I, Ethan Holland, voluntarily agree to participate in this user test. I understand that my participation is entirely voluntary and that I may withdraw at any time without penalty.

Purpose and Procedure:

I understand that the purpose of this user test is to evaluate the usability and effectiveness of the Figma-to-Code plugin.

The procedure will involve:

- Using the plugin within Figma to complete specific design-to-code tasks
- Interacting with plugin features such as generation, editing and navigation
- Providing verbal or written feedback on usability and performance
- Allowing the session to be recorded for analysis purposes

Figure 65 - A.1.2.2 - Ethan Consent Form Page 1

Confidentiality and Data Usage:

I understand that all information collected during this study, including video recordings, will be kept confidential. My personal data will not be shared with third parties and will only be used for academic research purposes and evaluation of the plugin.

Rights of Participants:

I understand that I have the right to:

- Refuse to answer any questions or perform tasks I am uncomfortable with
- Withdraw from the study at any time without penalty
- Request deletion of any personal data collected during the test

Use of Recorded Material:

I understand that recordings of my session may be used for analysis, academic reporting, or presentation purposes related to this project. My identity will remain confidential unless I provide explicit consent otherwise.

Contact Information:

If I have any questions or concerns about this user test or my participation in it, I can contact at *N00220082@iadt.ie*.

By signing below, I acknowledge that I have read and understood the terms and conditions outlined in this consent form, and I agree to participate in the user test under these conditions.

Participant's Signature: 

Date *(DD:MM:YYYY)*: **11/04/2026**

Print Name: (e.g "JOHN DOE"): **ETHAN HOLLAND**

Figure 66 - A.1.2.2 - Ethan Consent Form Page 2

A.1.2.3 Emma Consent Form

Project Title: Figma-to-Code Plugin

Principal Investigators: Matthew Dent

You are invited to participate in a user test for a Figma-to-Code plugin, developed to assist users in converting design components into functional code more efficiently. The purpose of this study is to evaluate the plugin's usability, performance, and overall user experience.

Your participation involves completing a set of design tasks using the plugin and providing feedback on your experience using the plugin during the completion of the task.

Participant Information:

- **Participant Name: Emma Hynes**

- **Date of Birth** *(DD:MM:YYYY)*:**25/01/2003**

- **Gender** (mark 'X'):
  - **Female** **X**
  - **Male**
  - **Non-Binary**
  - **Prefer Not to Say**

- **Contact Information** *(optional)*: **ehynes2003@gmail.com**

Consent to Participate:

I, **Emma Hynes** voluntarily agree to participate in this user test. I understand that my participation is entirely voluntary and that I may withdraw at any time without penalty.

Purpose and Procedure:

I understand that the purpose of this user test is to evaluate the usability and effectiveness of the Figma-to-Code plugin.

The procedure will involve:

- Using the plugin within Figma to complete specific design-to-code tasks
- Interacting with plugin features such as generation, editing and navigation
- Providing verbal or written feedback on usability and performance
- Allowing the session to be recorded for analysis purposes

Figure 67 - A.1.2.3 - Emma Consent Form Page1

Confidentiality and Data Usage:

I understand that all information collected during this study, including video recordings, will be kept confidential. My personal data will not be shared with third parties and will only be used for academic research purposes and evaluation of the plugin.

Rights of Participants:

I understand that I have the right to:

- Refuse to answer any questions or perform tasks I am uncomfortable with
- Withdraw from the study at any time without penalty
- Request deletion of any personal data collected during the test

Use of Recorded Material:

I understand that recordings of my session may be used for analysis, academic reporting, or presentation purposes related to this project. My identity will remain confidential unless I provide explicit consent otherwise.

Contact Information:

If I have any questions or concerns about this user test or my participation in it, I can contact at *N00220082@iadt.ie*.

By signing below, I acknowledge that I have read and understood the terms and conditions outlined in this consent form, and I agree to participate in the user test under these conditions.

Participant's Signature: Emma Hynes

Date *(DD:MM:YYYY)*: **18/04/2026**

Print Name: (e.g "JOHN DOE"): **EMMA HYNES**

Figure 68 - A.1.2.3 - Emma Consent Form Page 2

A.1.2.4 Adam Consent Form

Project Title: Figma-to-Code Plugin

Principal Investigators: Matthew Dent

You are invited to participate in a user test for a Figma-to-Code plugin, developed to assist users in converting design components into functional code more efficiently. The purpose of this study is to evaluate the plugin's usability, performance, and overall user experience.

Your participation involves completing a set of design tasks using the plugin and providing feedback on your experience using the plugin during the completion of the task.

Participant Information:

- **Participant Name:** Adam Smith
- **Date of Birth** *(DD:MM:YYYY)*: 25 / 05 / 2001
- **Gender** (mark 'X'):
  - o **Female**
  - X **Male**
  - o **Non-Binary**
  - o **Prefer Not to Say**
- **Contact Information** *(optional)*:

Consent to Participate:

I, [Participant's Full Name], voluntarily agree to participate in this user test. I understand that my participation is entirely voluntary and that I may withdraw at any time without penalty.

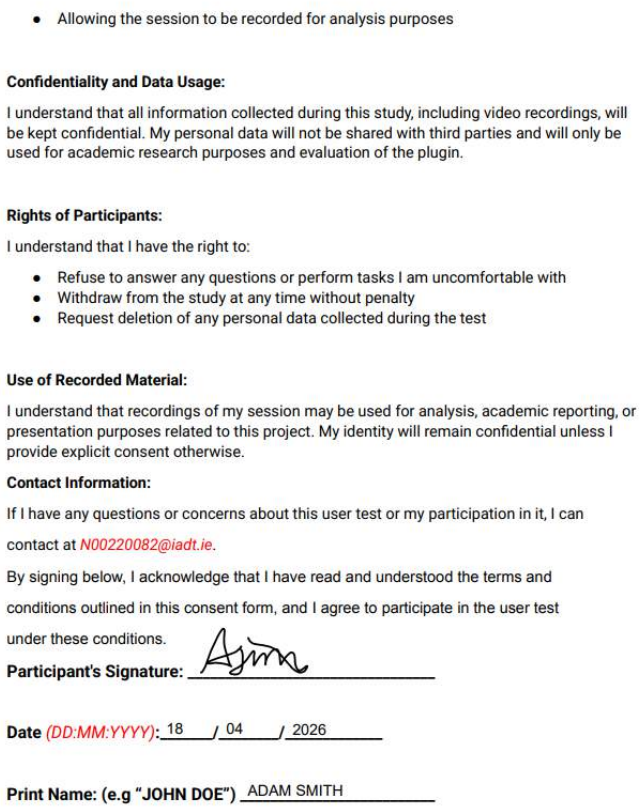
Purpose and Procedure:

I understand that the purpose of this user test is to evaluate the usability and effectiveness of the Figma-to-Code plugin.

The procedure will involve:

- Using the plugin within Figma to complete specific design-to-code tasks
- Interacting with plugin features such as generation, editing and navigation
- Providing verbal or written feedback on usability and performance

Figure 69 - A.1.2.4 - Adam Consent Form Page 1



- Allowing the session to be recorded for analysis purposes

Confidentiality and Data Usage:

I understand that all information collected during this study, including video recordings, will be kept confidential. My personal data will not be shared with third parties and will only be used for academic research purposes and evaluation of the plugin.

Rights of Participants:

I understand that I have the right to:

- Refuse to answer any questions or perform tasks I am uncomfortable with
- Withdraw from the study at any time without penalty
- Request deletion of any personal data collected during the test

Use of Recorded Material:

I understand that recordings of my session may be used for analysis, academic reporting, or presentation purposes related to this project. My identity will remain confidential unless I provide explicit consent otherwise.

Contact Information:

If I have any questions or concerns about this user test or my participation in it, I can contact at N00220082@iadt.ie.

By signing below, I acknowledge that I have read and understood the terms and conditions outlined in this consent form, and I agree to participate in the user test under these conditions.

Participant's Signature: _____

Date *(DD:MM:YYYY)*: 18 / 04 / 2026

Print Name: (e.g "JOHN DOE") ADAM SMITH

Figure 70 - A.1.2.4 - Adam Consent Form Page 2

A.1.3 Support Documentation

Supported documentation for the use case task, guided tutorial and unsigned consent form are displayed in Appendix F.

12.2. Appendix B

12.2.1. Survey Test Cases and Raw Results

This appendix displays the surveys used to gather information and feedback required for guiding the project direction. The Frontend Developer Workflow survey prioritised understanding the workflows of frontend designers and developers. These questions included research into preference of design tools, frameworks, as well as positives and negatives of design to development workflows. Additionally, the Figma-to-Code survey collected feedback from users after user testing was completed. The questions directed focus on understanding the functionality of the application from a user's perspective, including visual accessibility, quality of the functionality and ease of use. The surveys were conducted on the Google Forms application and results were converted into an exported CSV file.

12.2.2. Appendix B.1 Frontend Developer Workflows, Design Tools, and Design-to-Code Automation Survey

B.1.1 Survey Questions

Q1. What best describes your primary role?

Q2. Years of professional development experience?

Q3. Industry you primarily work in (e.g Web development, startups, agency, e-commerce)

Q4. Team size you usually work in?

Q5. Level of involvement in UI decisions

Q6. How frequently do you build UI components from scratch?

Q7. Do you follow an established system at work? (shared components, spacing scales, and design rules)

Q8. At what stage do you usually start writing code?

Q9. How often do designs change after development has started?

Q10. What slows down your UI implementation the most?

Q11. How do you validate UI accuracy against designs?

Q12. Which design tools do you interact with?

Q13. What do you mainly use designs for?

Q14. How often do you use prototypes or wireframes?

Q15. Do prototypes accurately reflect real code implementations?

Q16. What is commonly missing from wireframes or prototypes?

Q17. How frequently do you use Figma?

Q18. Which Figma features do you rely on most?

Q19. What Figma features cause the most confusion during implementation?

Q20. How consistent is Figma's inspect panel values when implementing designs in code?

Q21. What styling framework approach do you prefer?

Q22. Why do you use or avoid Tailwind CSS?

Q23. How do you handle non-standard spacing values from designs?

Q24. What are the biggest drawbacks of utility-first CSS? In your opinion.

Q25. What benefits does Tailwind provide in your workflow?

Q26. What framework do you mainly build components in?

Q27. How do you structure UI components?

Q28. How often do you refactor components due to design changes?

Q29. What component aspects take the most time to implement?

Q30. Have you used design-to-code tools?

Q31. What challenges or limitations have you experienced from using code generation tools?

Q32. What features would make automated code generation usable in your workflow?

Q33. Do you use Figma Plugins in your workflow?

Q34. What tasks do plugins help you with?

Q35. What limitations have you encountered with plugins?

Q36. What features would an ideal design-to-code plugin have?

Q37. How often do you manually convert design or inline CSS values to Tailwind classes?

Q38. Would an automated conversion tool improve your workflow?

Q39. What problems would such a tool solve for you?

Q40. At which stage of your workflow would this tool be most useful?

Q41. Do you believe this tool would benefit teams at scale? Why?

Q42. What is the biggest inefficiency in your current UI workflow?

Q43. What tooling improvements would you like to see in the industry?

Q44. Any additional insights related to frontend design, tooling or automated code generation?

B.1.2 Raw Results

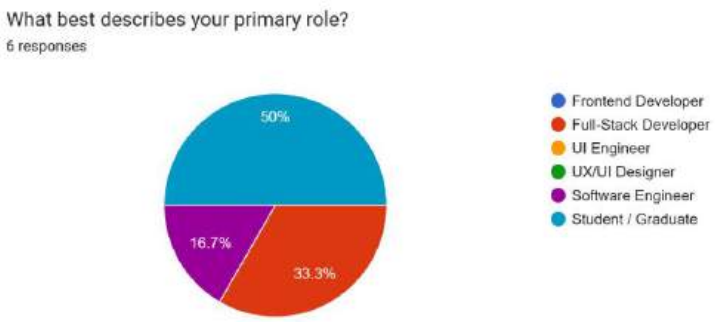


Figure 71 - B.1.2 Q1

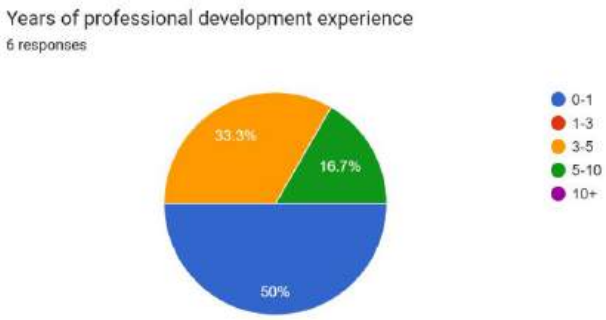


Figure 72 - B.1.2 Q2

Figure 73 - B.1.2 Q3

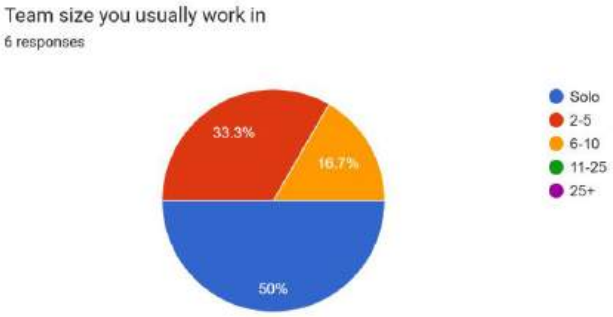


Figure 74 - B.1.2 Q4

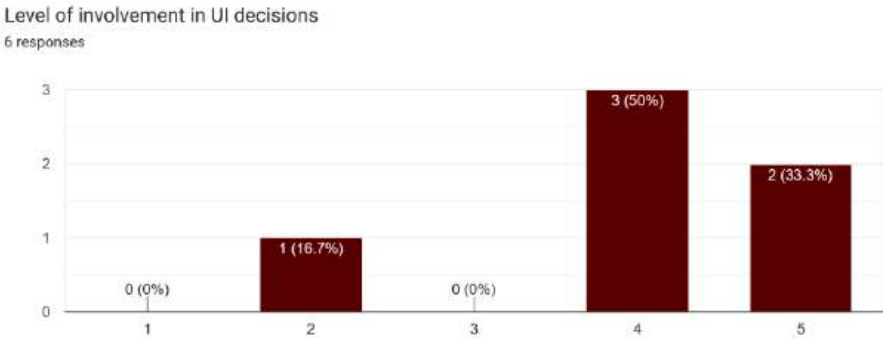


Figure 75 - B.1.2 Q 5

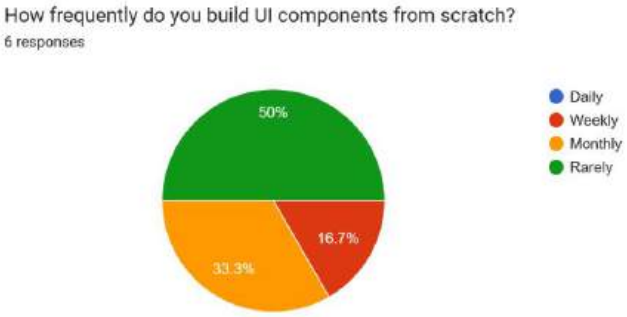


Figure 76 - B.1.2 Q 6

Do you follow an established design system at work? (shared components, spacing scales, and design rules)

6 responses

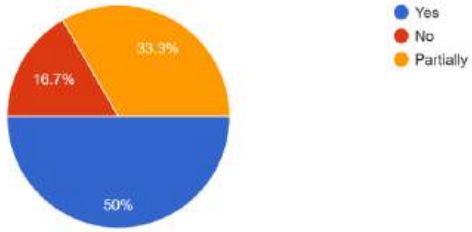


Figure 77 - B.1.2 Q 7

Describe your typical workflow from initial design to deployed UI

5 responses

Wireframes, Figma, Creating Website, Testing, Deployment

low fidelity wireframes > low fidelity coded project > high fidelity design > high fidelity project > deploy

For small features a general concept is envisioned by text description, and is then implemented, matching existing styling. A stakeholder is then consulted for input on the result before tweaks and finalisation.

For larger features, we generally have the UI prototyped via a UX designer in Adobe XD, before being split up into components and the implementation done. Each component of the design is generally added into Storybook, which helps consistency, and allows us to show stakeholders prior to the functionality being added into the main application.

Try to follow predefined ui, such as teachers' code

complete feature analysis, determine what data is being populating, and what procedure populates them -> develop wireframes that utilise the benefits of the updated technology -> get wireframes approved from client -> create components and unit tests within IDE and push to GitLab repository -> GitLab deploys UI once merged and criteria is met -> review sessions are complete

Figure 78 - B.1.2 Q8

At what stage do you usually start writing code?

6 responses

Figure 79 - B.1.2 Q 9

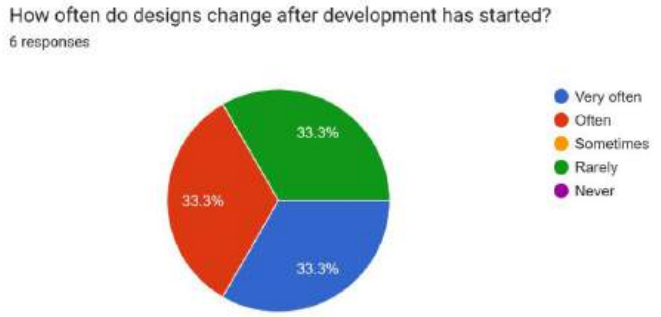


Figure 80 - B.1.2 Q 10

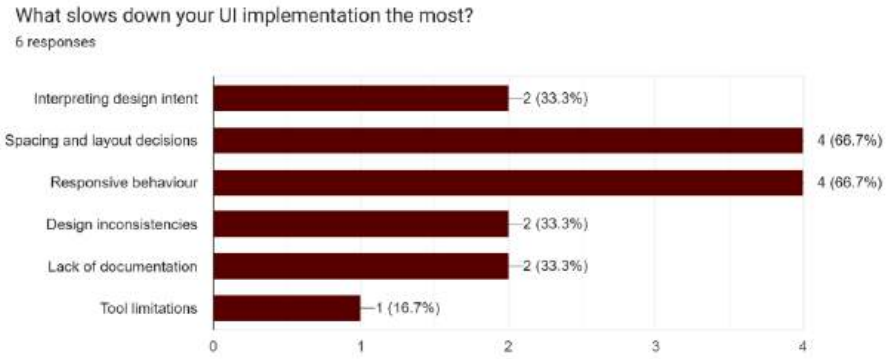


Figure 81 - B.1.2 Q 11

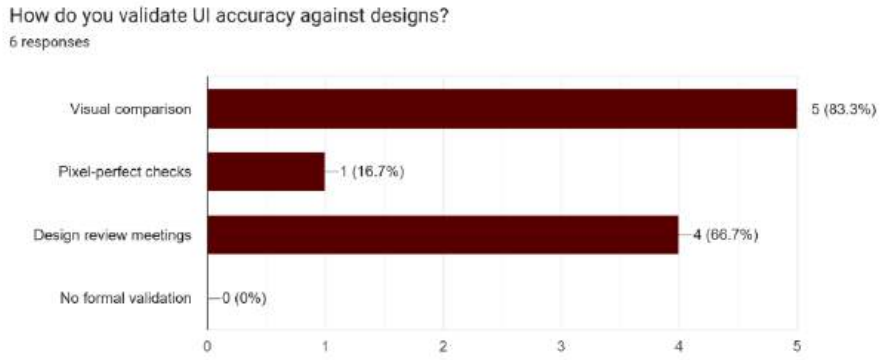


Figure 82 - B.1.2 Q 12

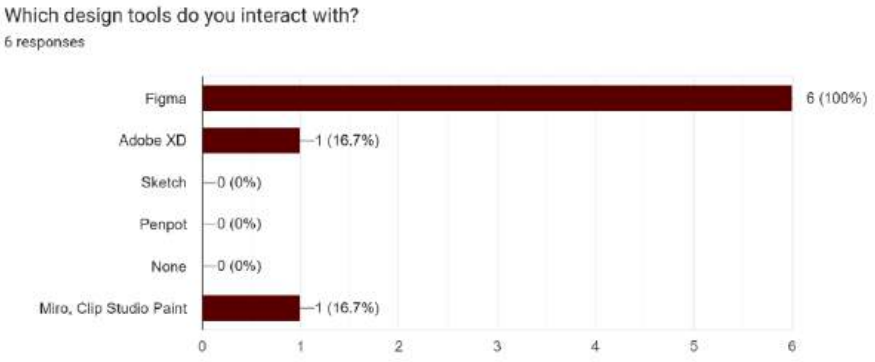


Figure 83 - B.1.2 Q 13

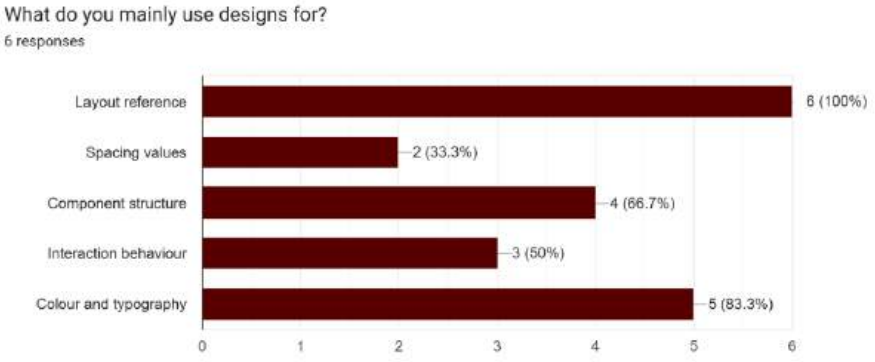


Figure 84 - B.1.2 Q 14

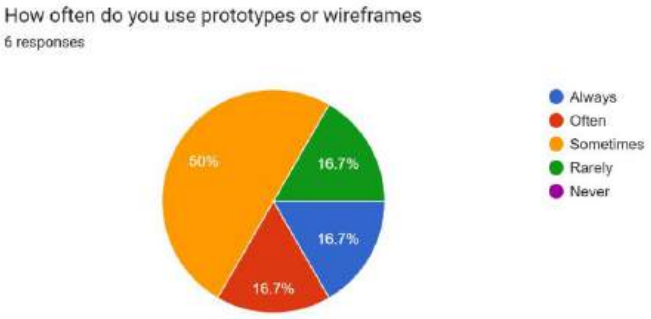


Figure 85 - B.1.2 Q 15

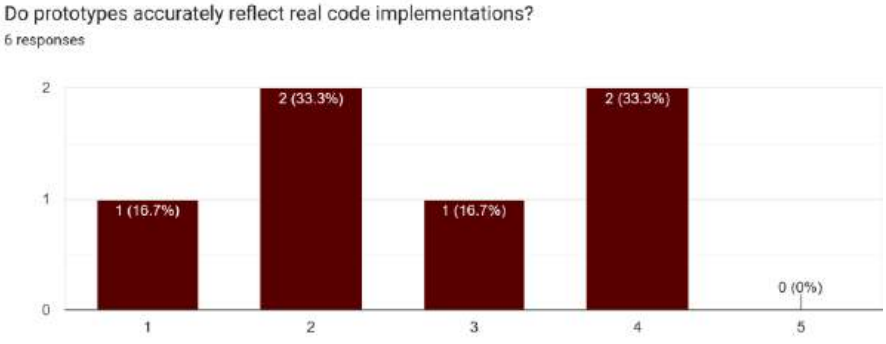


Figure 86 - B.1.2 Q 16

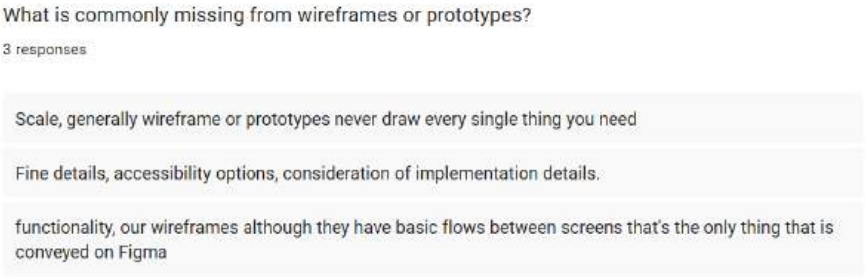


Figure 87 - B.1.2 Q 17

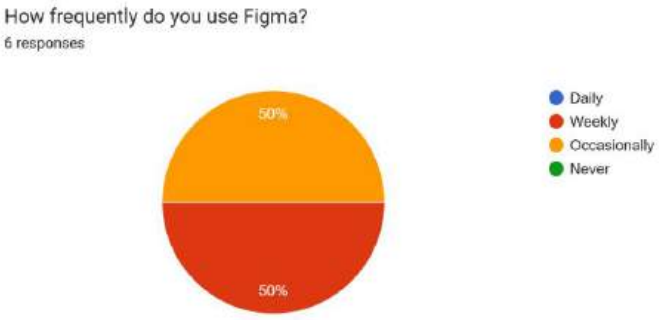


Figure 88 - B.1.2 Q 18

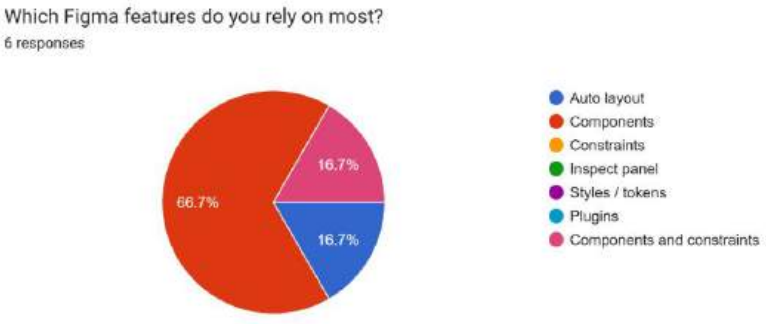


Figure 89 - B.1.2 Q 19

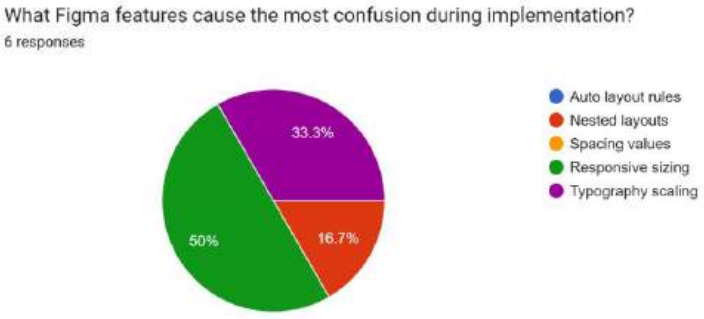


Figure 90 - B.1.2 Q 20

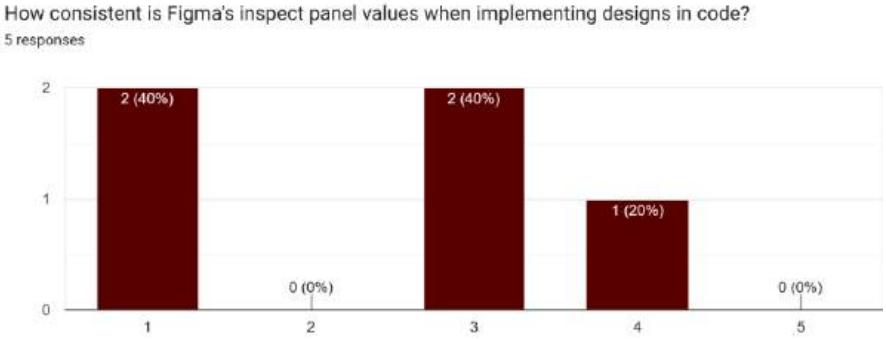


Figure 91 - B.1.2 Q 21

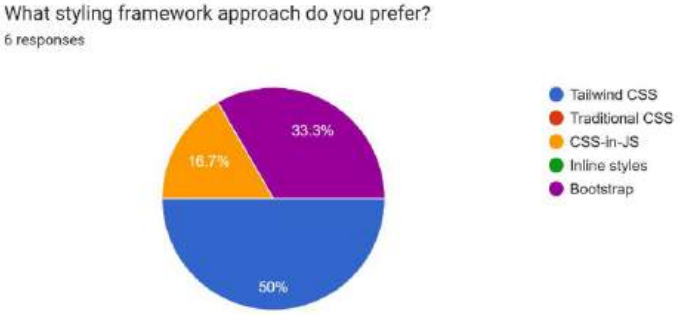


Figure 92 - B.1.2 Q 22

Why do you use or avoid Tailwind CSS

5 responses

I find it can become messy or overcomplicated

Tailwind is great for getting something working quickly, and exploring design with code

I use tailwind.
Pros:
- See immediately how a component is styled
- Very easy to see what a classname does, and very easy to add/change styles
- Compiles into small packages

Confusing

Not our company standard

Figure 93 - B.1.2 Q 23

How do you handle non-standard spacing values from designs

6 responses

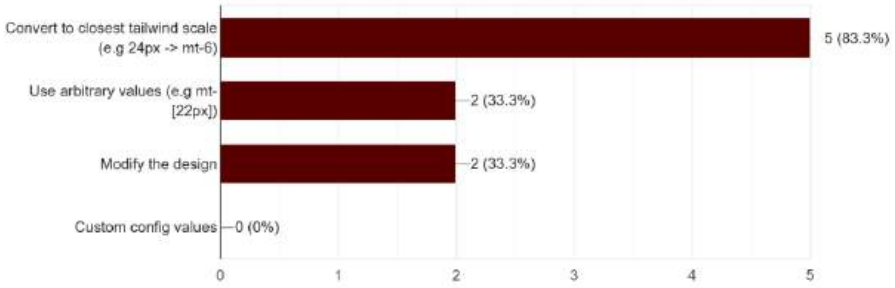


Figure 94 - B.1.2 Q 24

What are the biggest drawbacks of utility-first CSS? in your opinion

3 responses

Too much to keep track off

- Lots of boilerplate
- Can be hard to change a particular aspect of the styling across the application if it has not been created with that change in mind
- Sometimes custom classes are required if the utility-first library does not support a certain CSS option

can be hard to make a cohesive name where there could be a lot of different uses

Figure 95 - B.1.2 Q 25

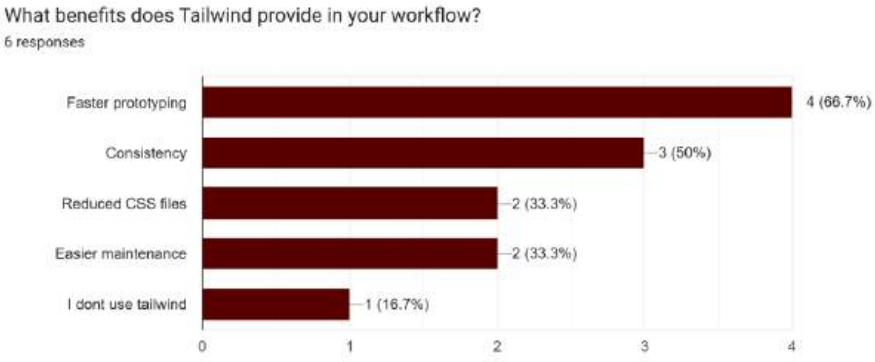


Figure 96 - B.1.2 Q 26

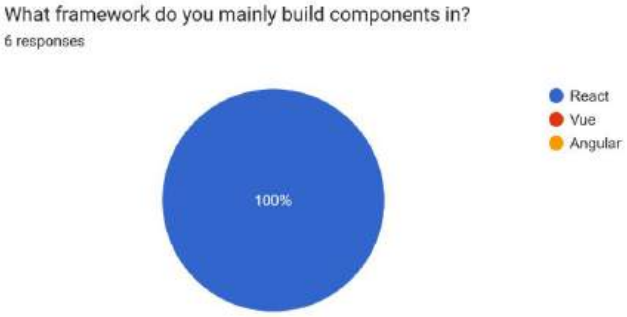


Figure 97 - B.1.2 Q 27

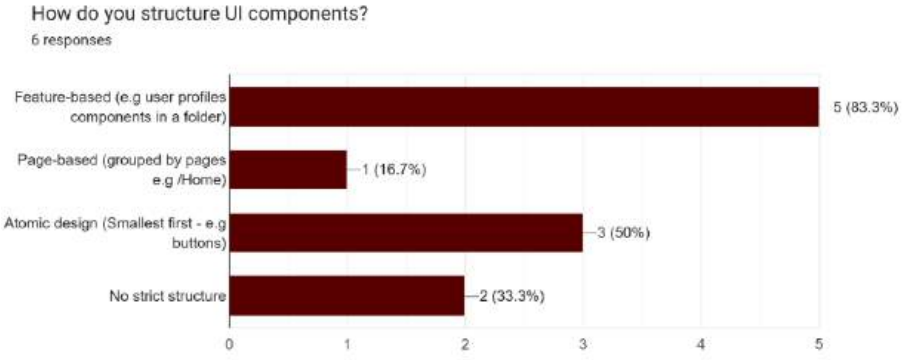


Figure 98 - B.1.2 Q 28

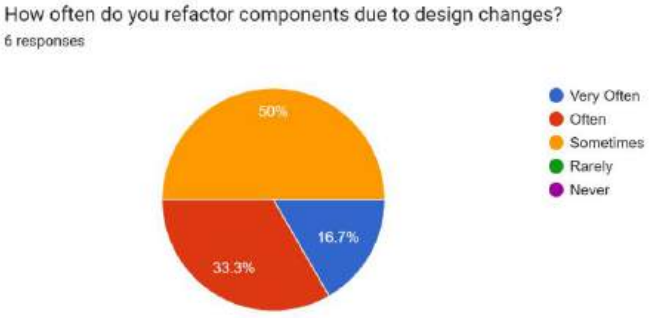


Figure 99 - B.1.2 Q 29

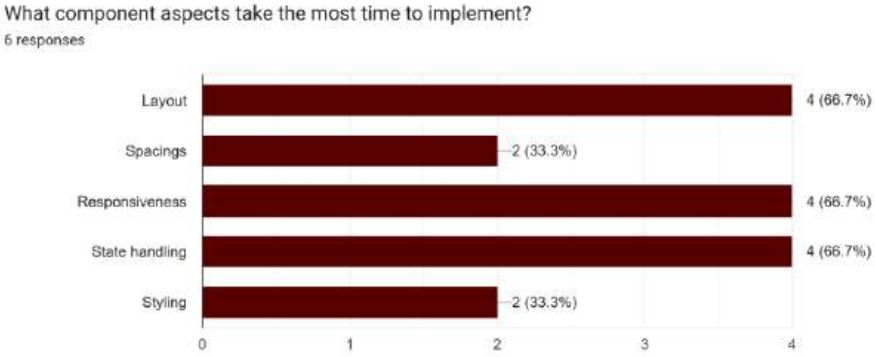


Figure 100 - B.1.2 Q 30

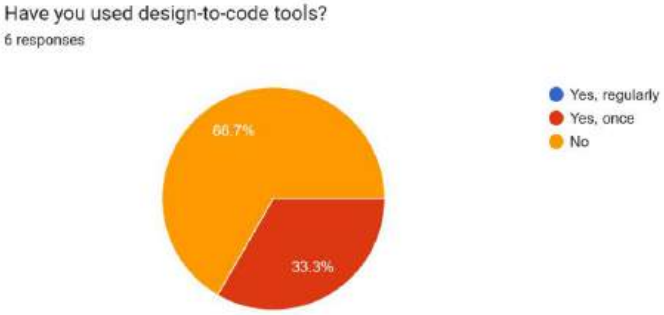


Figure 101 - B.1.2 Q 31

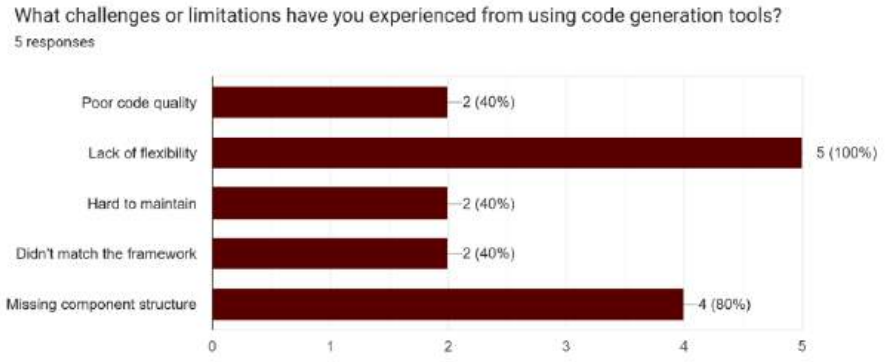


Figure 102 - B.1.2 Q 32

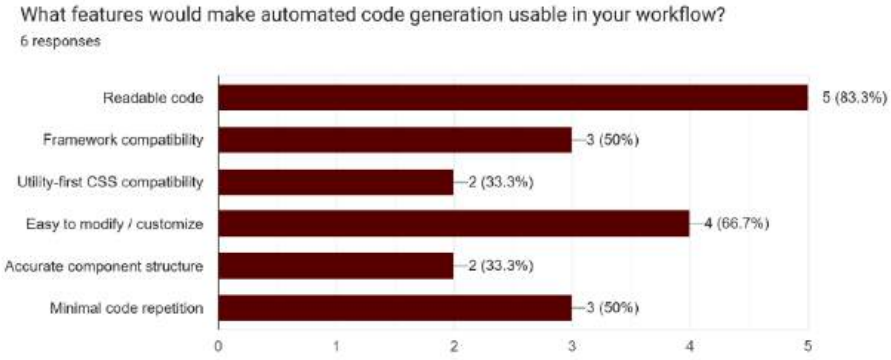


Figure 103 - B.1.2 Q 33

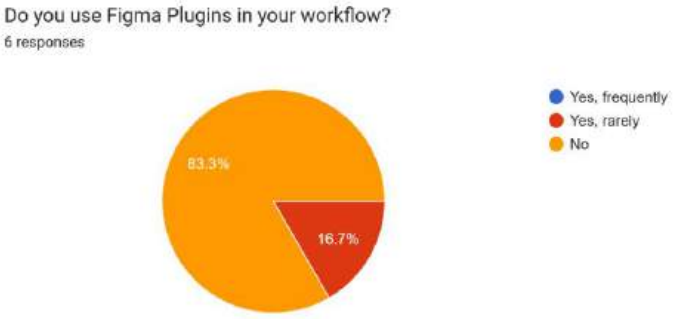


Figure 104 - B.1.2 Q 34

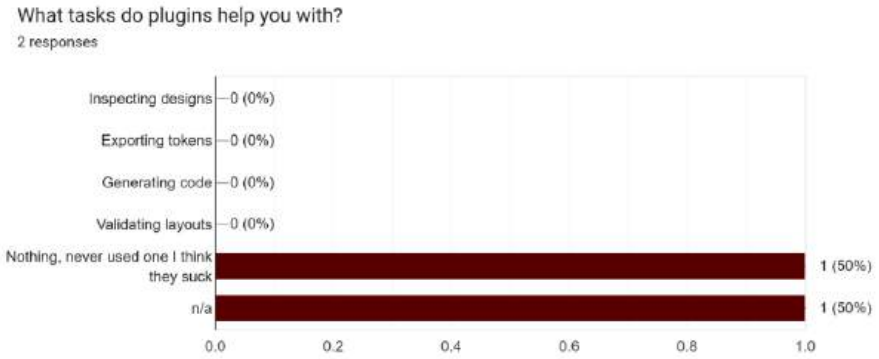


Figure 105 - B.1.2 Q 35

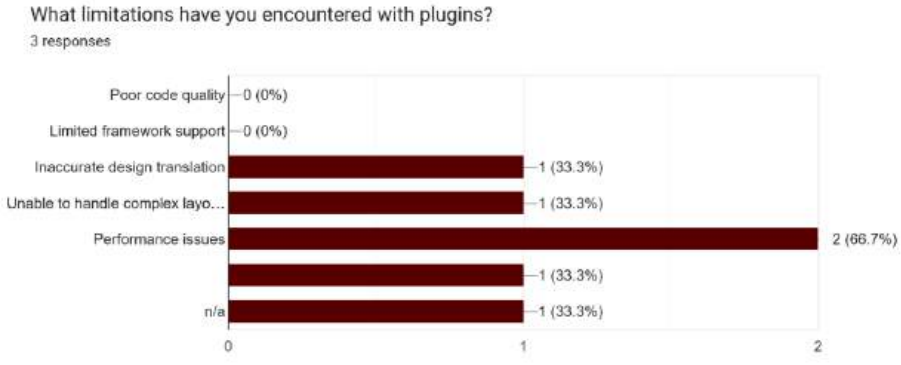


Figure 106 - B.1.2 Q 36

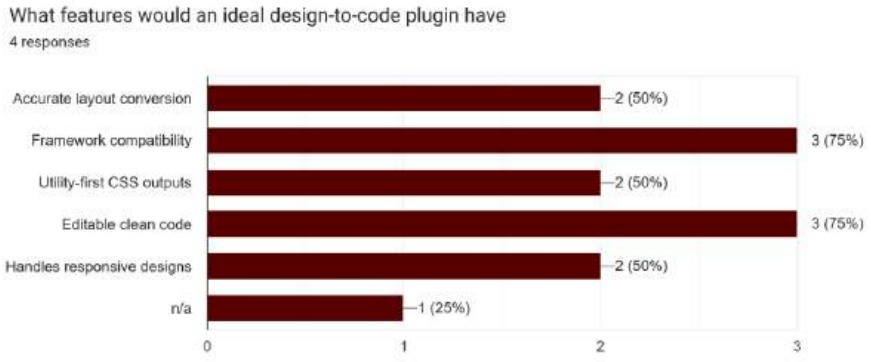


Figure 107 - B.1.2 Q 37

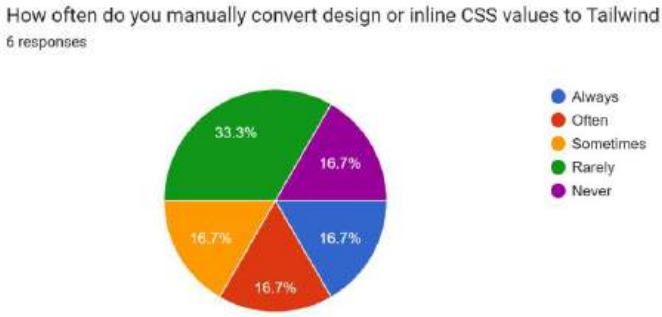


Figure 108 - B.1.2 Q 38

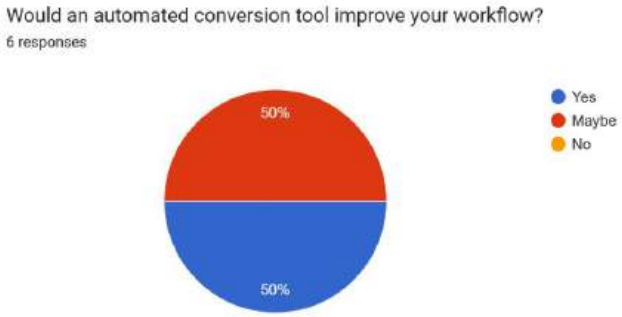


Figure 109 - B.1.2 Q 39

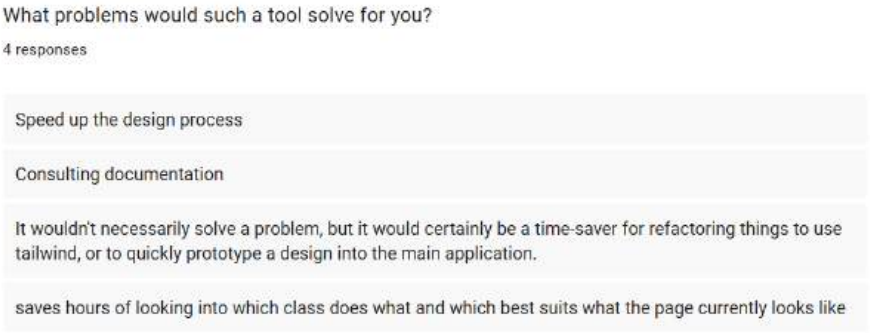


Figure 110 - B.1.2 Q 40

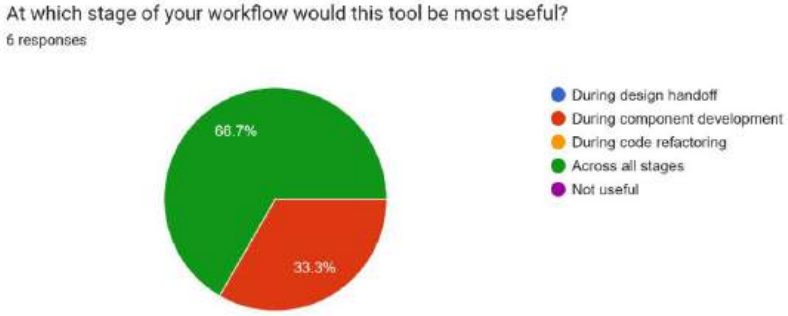


Figure 111 - B.1.2 Q 41

Figure 112 - B.1.2 Q 42

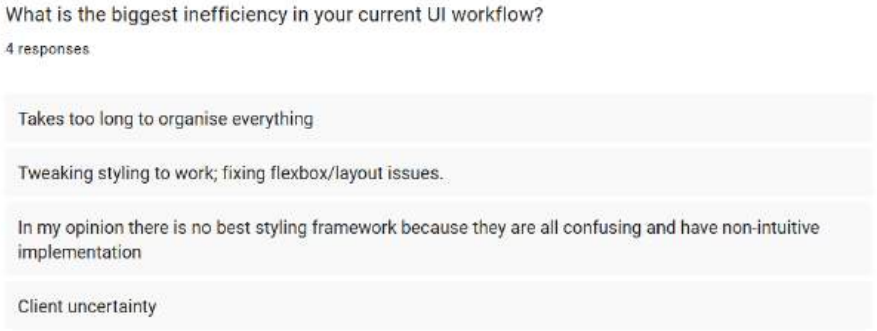


Figure 113 - B.1.2 Q 43

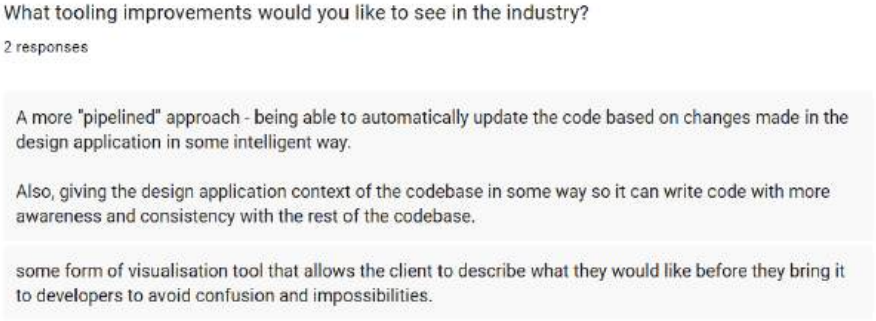


Figure 114 - B.1.2 Q 44

B.1.3 Results Summary

The results of the survey showcase trends from the responses collected from 6 respondents. These findings present that respondents were primarily new to the career and currently pursuing a degree as a student. This assures the requirement for an

application that is easy to use and learn. Through reviewing the results manual translation of design components into code was highlighted as an issue, design properties such as layout, spacing and styling were inconsistent when translating into code. The use of frameworks such as Tailwind CSS and React were prominent in the data formed, this guides the choice of frameworks to incorporate into the functionality. Finally, the data indicates the need for code quality to provide accurate visual designs and maintainability of code.

12.2.3. Appendix B.2 Figma-to-Code Plugin User Survey

B.2.1 Survey Questions

Q1. Did you complete the design recreation task?

Q2. How long did the task take?

Q3. How would you describe your Figma experience level?

Q4. How familiar were you with design tokens before this task?

Q5. How clear were the task instructions overall?

Q6. How helpful was the guided footer tutorial?

Q7. After the tutorial, how confident were you in completing the remaining sections?

Q8. Did the annotated reference design make the task easier?

Q9. Overall, the plugin was easy to use (1-5 agreement).

Q10. Navigating colour and typography tokens was straightforward.

Q11. Applying spacing tokens (padding, gap, radius) was straightforward.

Q12. Applying auto layout styles was straightforward.

Q13. Which plugin area was most difficult?

Q14. Which action caused most confusion?

Q15. How close was your final design to the target reference?

Q16. How confident are you that you could repeat this task without help?

Q17. Overall satisfaction with the plugin experience.

Q18. Would you use this plugin again for similar design-to-code tasks?

Q19. Would you recommend this plugin to others?

Q20. What worked best for you in the plugin?

Q21. What was the most frustrating part of the experience?

Q22. What specific improvement would most help you complete tasks faster?

B.2.2 Raw Results

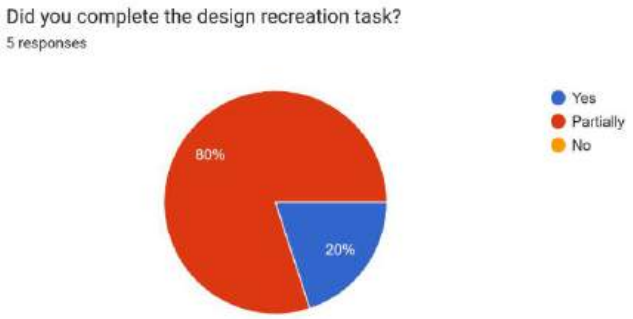


Figure 115 - B.2.2 Q 1

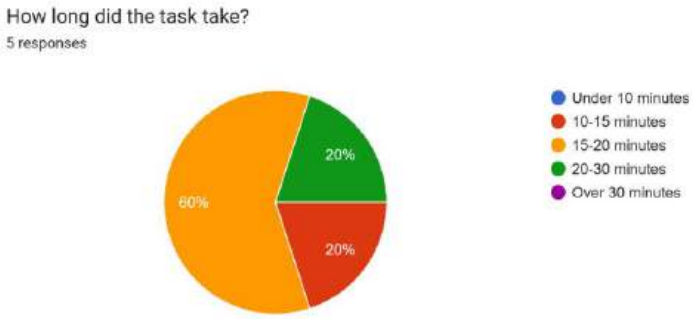


Figure 116 - B.2.2 Q 2

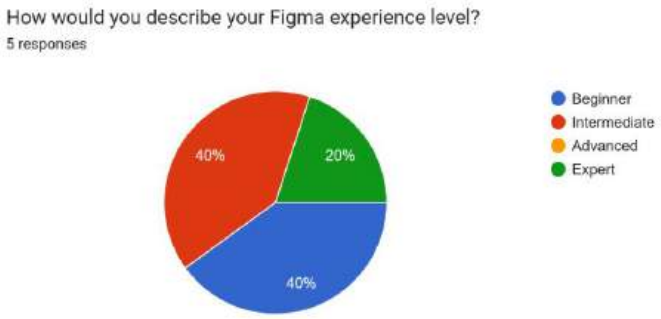


Figure 117 - B.2.2 Q 3

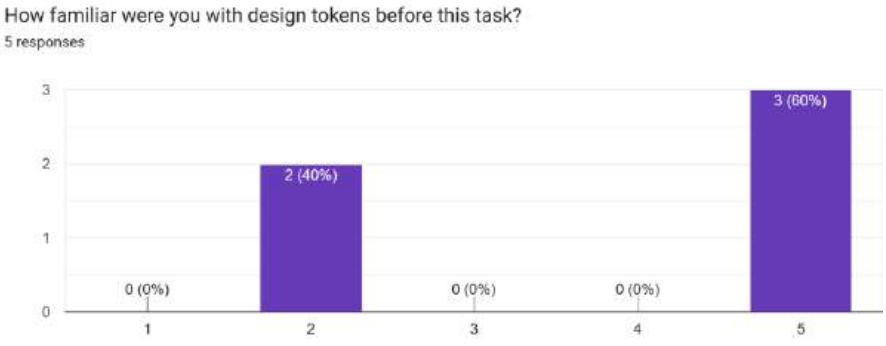


Figure 118 - B.2.2 Q 4

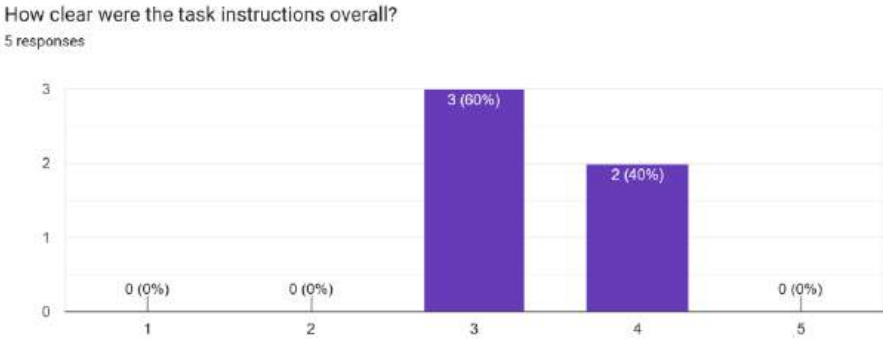


Figure 119 - B.2.2 Q 5

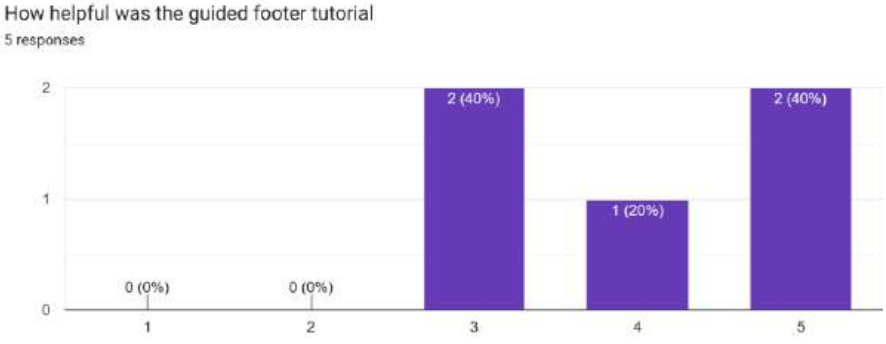


Figure 120 - B.2.2 Q 6

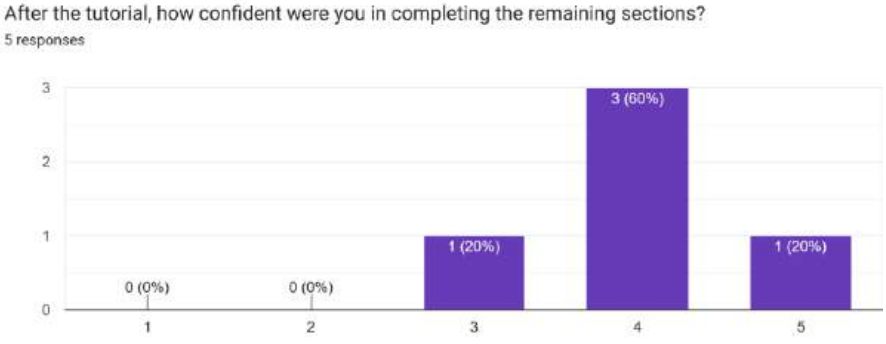


Figure 121 - B.2.2 Q 7

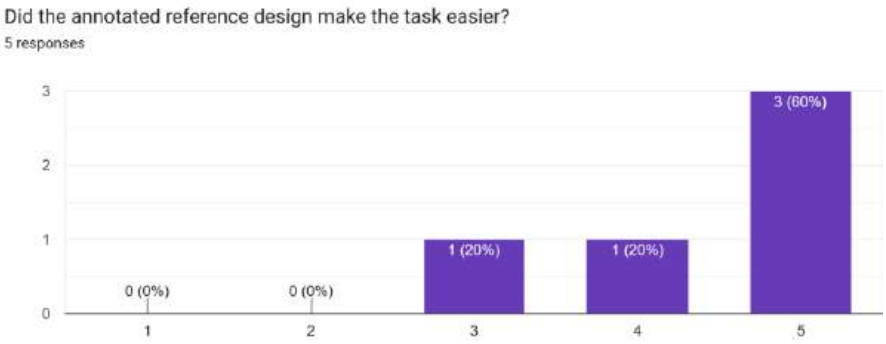


Figure 122 - B.2.2 Q 8

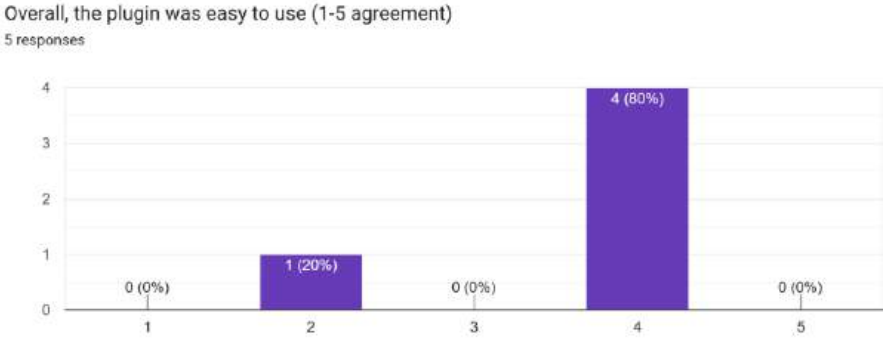


Figure 123 - B.2.2 Q 9

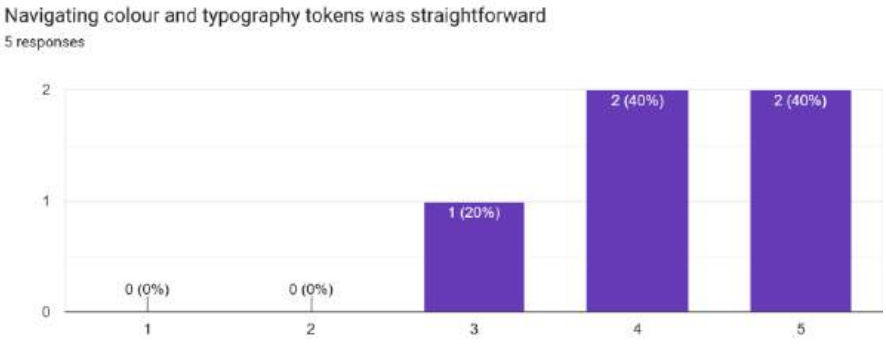


Figure 124 - B.2.2 Q 10

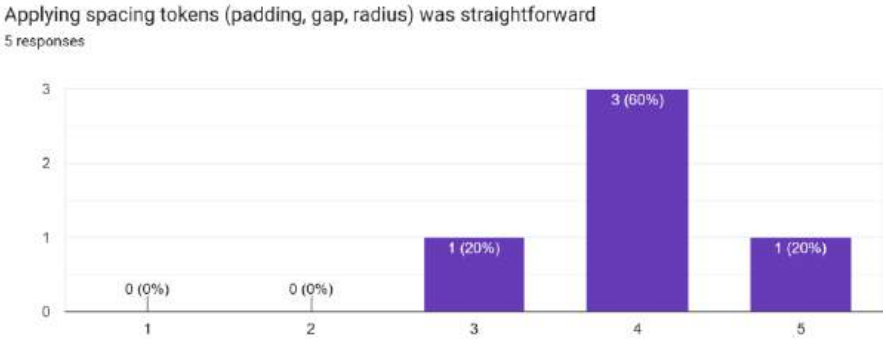


Figure 125 - B.2.2 Q 11

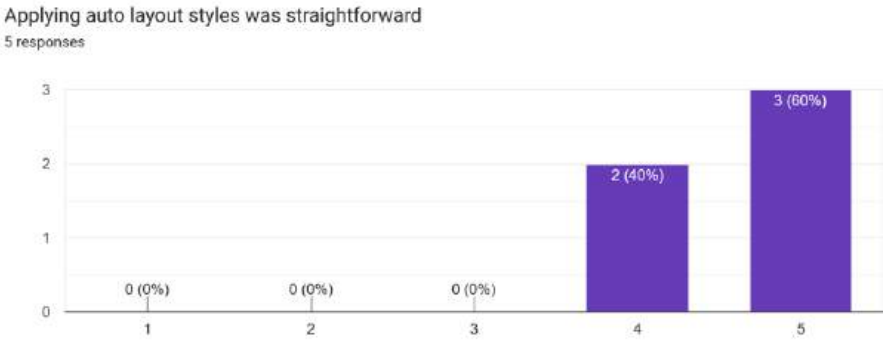


Figure 126 - B.2.2 Q 12

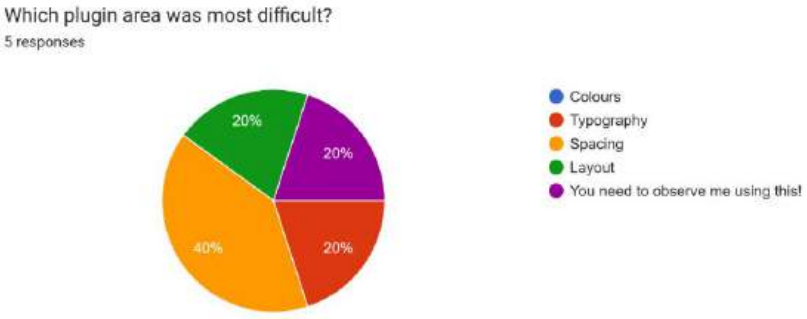


Figure 127 - B.2.2 Q 13

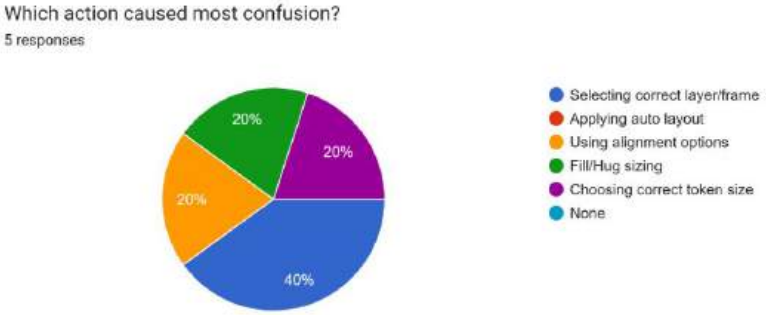


Figure 128 - B.2.2 Q 15

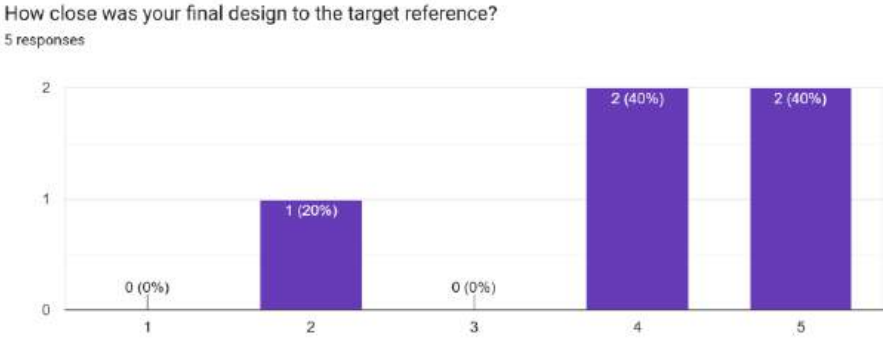


Figure 129 - B.2.2 Q 16

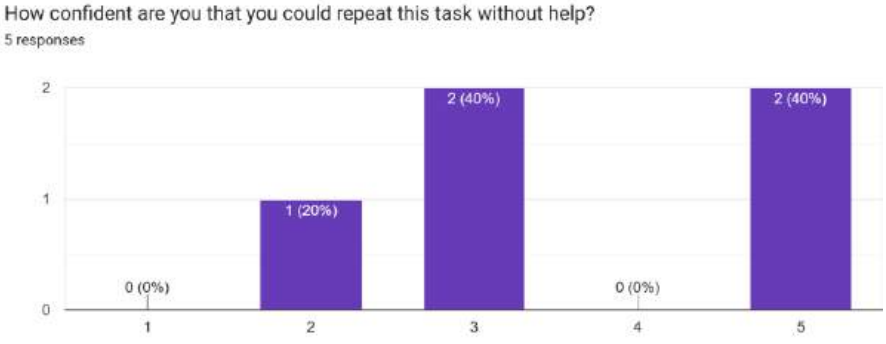


Figure 130 - B.2.2 Q 17

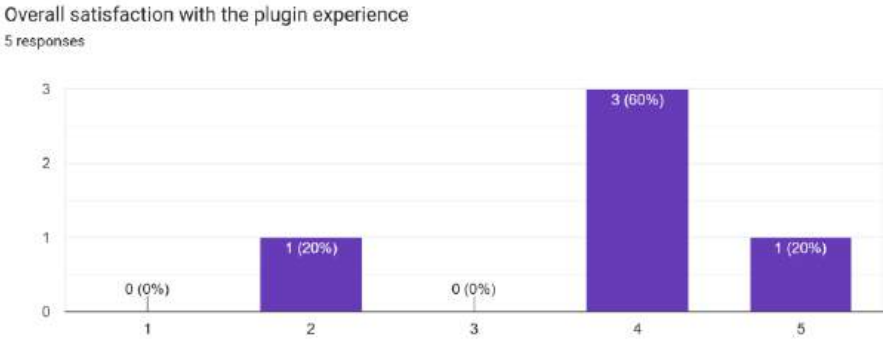


Figure 131 - B.2.2 Q 18

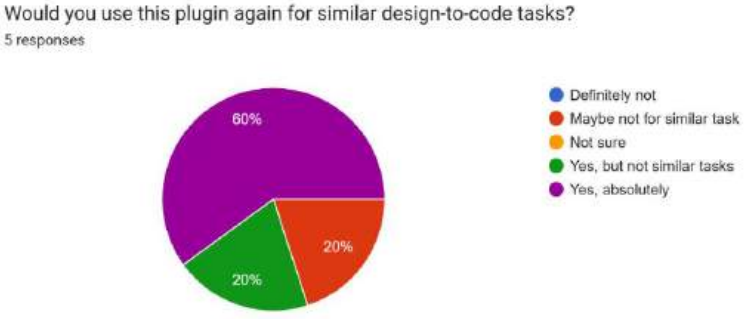


Figure 132 - B.2.2 Q 19

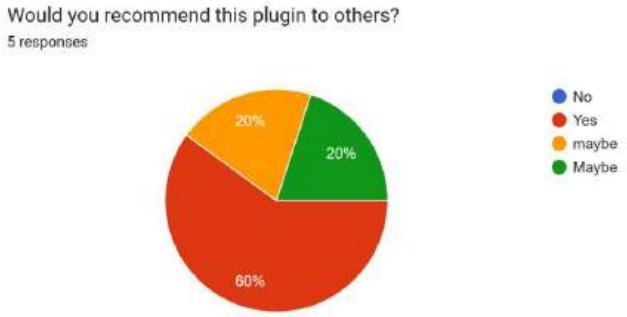


Figure 133 - B.2.2 Q 20

Figure 134 - B.2.2 Q 21

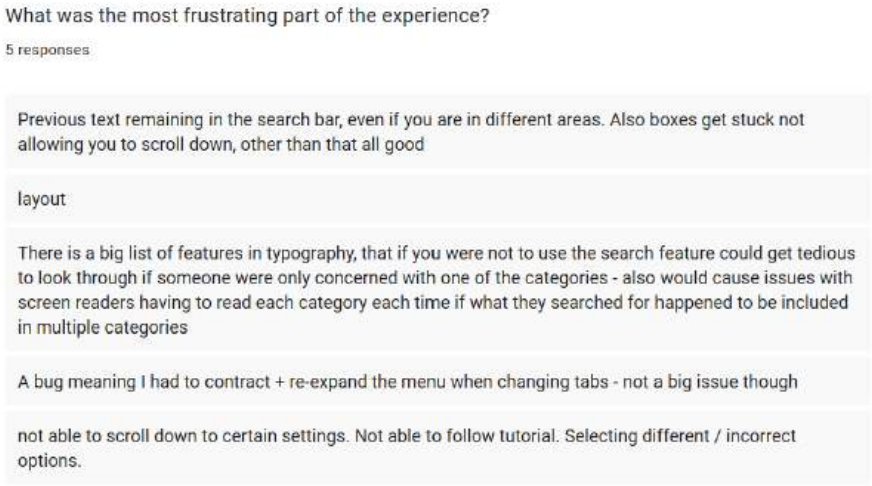


Figure 135 - B.2.2 Q 22

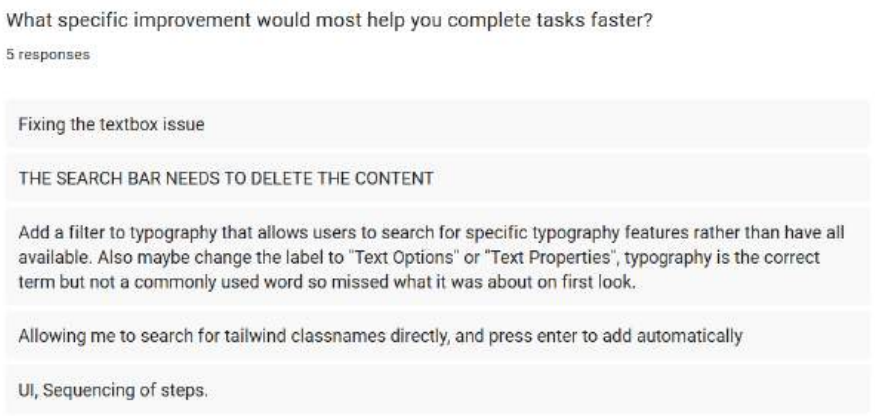


Figure 136 - B.2.2 Q 23

B.2.3 Results Summary

The results of the survey provided feedback that guided the alterations required to improve the usability, accessibility and functionality of the plugin. These findings were collected after the completion of user testing tasks. Participants mostly completed the tasks, as they followed the tutorial guide and successfully designed multiple components of the design, however the average time it took for partial completion was 15 to 20 minutes.

The findings indicated that usability was the cause of confusion which extended the task timer and reduced efficiency of the plugin. The feedback ratings informed that the usability and ease of use of the system are highly approved. Features such as the guided tutorial and annotated diagrams assisted in the learning process of the plugin.

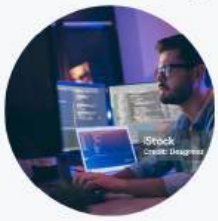
The feedback provides insight into bugs and errors that became problematic throughout the user testing process. These included user interface sizing, navigation difficulties, search script faults and lack of filter options. However, users stated their satisfaction with the system remained high, implying that the base functionality performs accurately.

12.3. Appendix C

12.3.1. Diagrams

C.1.1 Personas

Persona 1 - Frontend Developer
Name: Alex Murphy
Age: 27
Role: Frontend Developer
Experience: 4 years
Tech Stack: React, Tailwind CSS, TypeScript, Figma



Background:
Works in a product team building responsive web applications. Frequently collaborates with designers using Figma and converts designs into React components styled with Tailwind.
Goals:
Quickly convert Figma designs into production-ready code
Maintain consistent styling using Tailwind utilities
Reduce repetitive manual CSS translation
Frustrations:
Manually translating design values to Tailwind classes
Inconsistent spacing and colour values from design files
Time lost recreating components
Needs:
Automatic extraction of layout and styling from Figma
Tailwind class suggestions mapped from design properties
Clean React component output ready for development
How the Plugin Helps:
The plugin extracts node properties, applies Tailwind tokens, and generates React + Tailwind code that can be directly integrated into the project.

Figure 137 - C.1.1 - Persona 1

Persona 2 - Full Stack Developer
Name: Daniel Cliff
Age: 32
Role: Full Stack Developer
Experience: 8 years
Tech Stack: React, Next.js, Tailwind, Node.js

Background:
Builds complete applications from backend APIs to frontend UI. Often works independently or in small teams where efficiency is critical.
Goals:
Rapidly prototype interfaces from design files
Maintain scalable design systems
Keep frontend code consistent with Tailwind scaling
Frustrations:
Designs not structured properly for development
Time spent rebuilding layout structures from scratch
Lack of tools that connects design and production code
Needs:
Ability to select a Figma frame and extract structure
Token-based styling that aligns with Tailwind standards
React components ready for production workflows
How the Plugin Helps:
The plugin reads the selected frame, maps design properties to Tailwind tokens, and exports structured React components suitable for real-world applications.

Figure 138 - C.1.1 - Persona 2

Persona 3 - UI/UX Designer
Name: Sofia Sloan
Age: 29
Role: UI/UX Designer
Experience: 6 years
Tools: Figma, Prototyping Tools, Adobe XD

Background:
Designs user interfaces for web applications and collaborates closely with developers to ensure accurate implementation.
Goals:
Ensure design consistency across components
Reduce design-to-development miscommunication
Use design tokens to maintain visual consistency
Frustrations:
Developers misinterpreting spacing, typography, or colours
Design systems not translating cleanly into code
Repeated clarification of design specifications
Needs:
Clear mapping between design tokens and development styles
Tools that maintain consistency between Figma and code
Faster handoff from design to development
How the Plugin Helps:
The plugin allows designers to apply Tailwind-based tokens within Figma, ensuring the design values directly translate into developer-ready code.

Figure 139 - C.1.1 - Persona 3

C.1.2 Wireframes

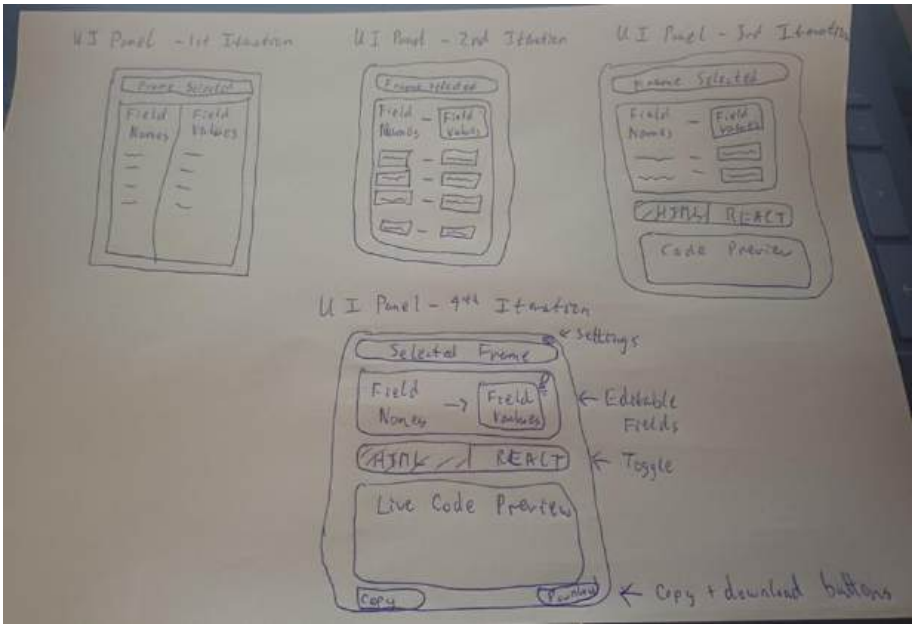


Figure 140 - C.1.2 - Paper Prototypes

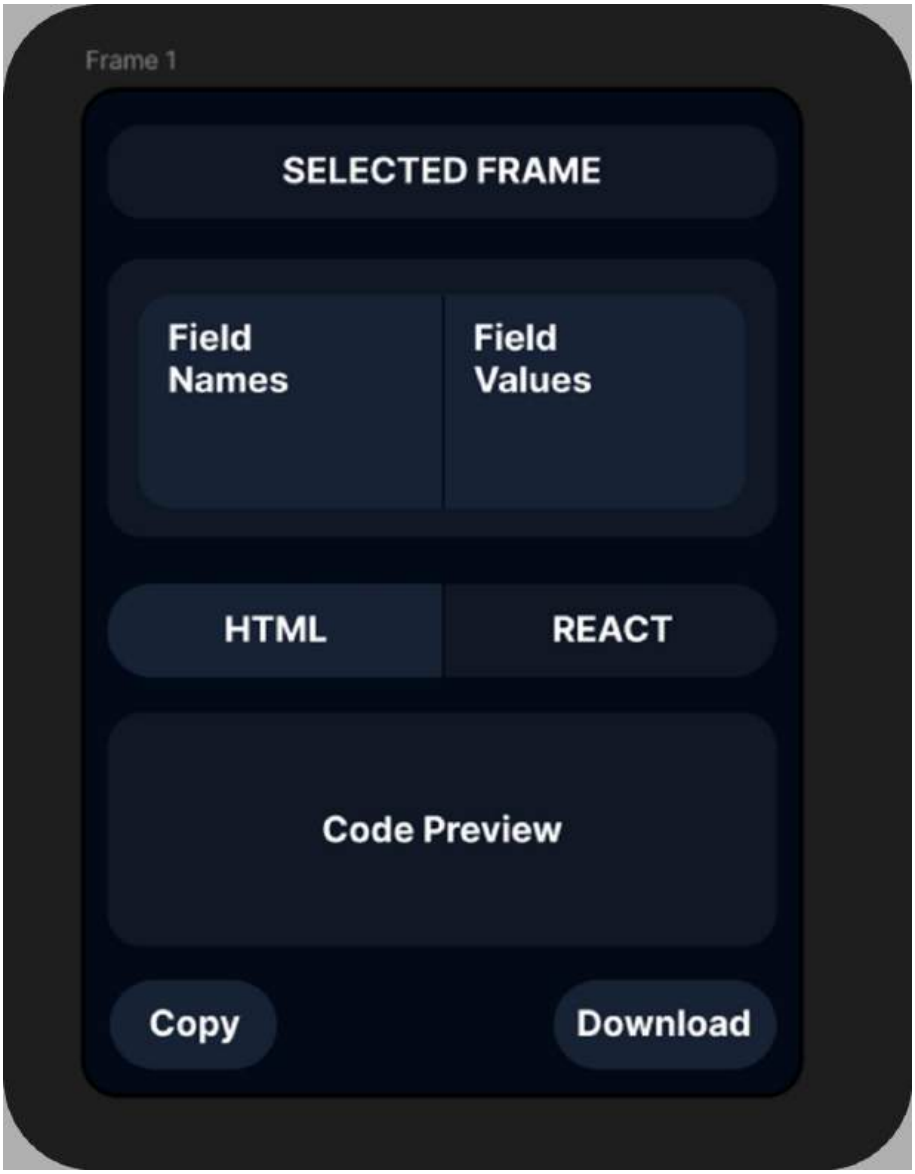


Figure 141 - C.1.2 - Wireframe 1

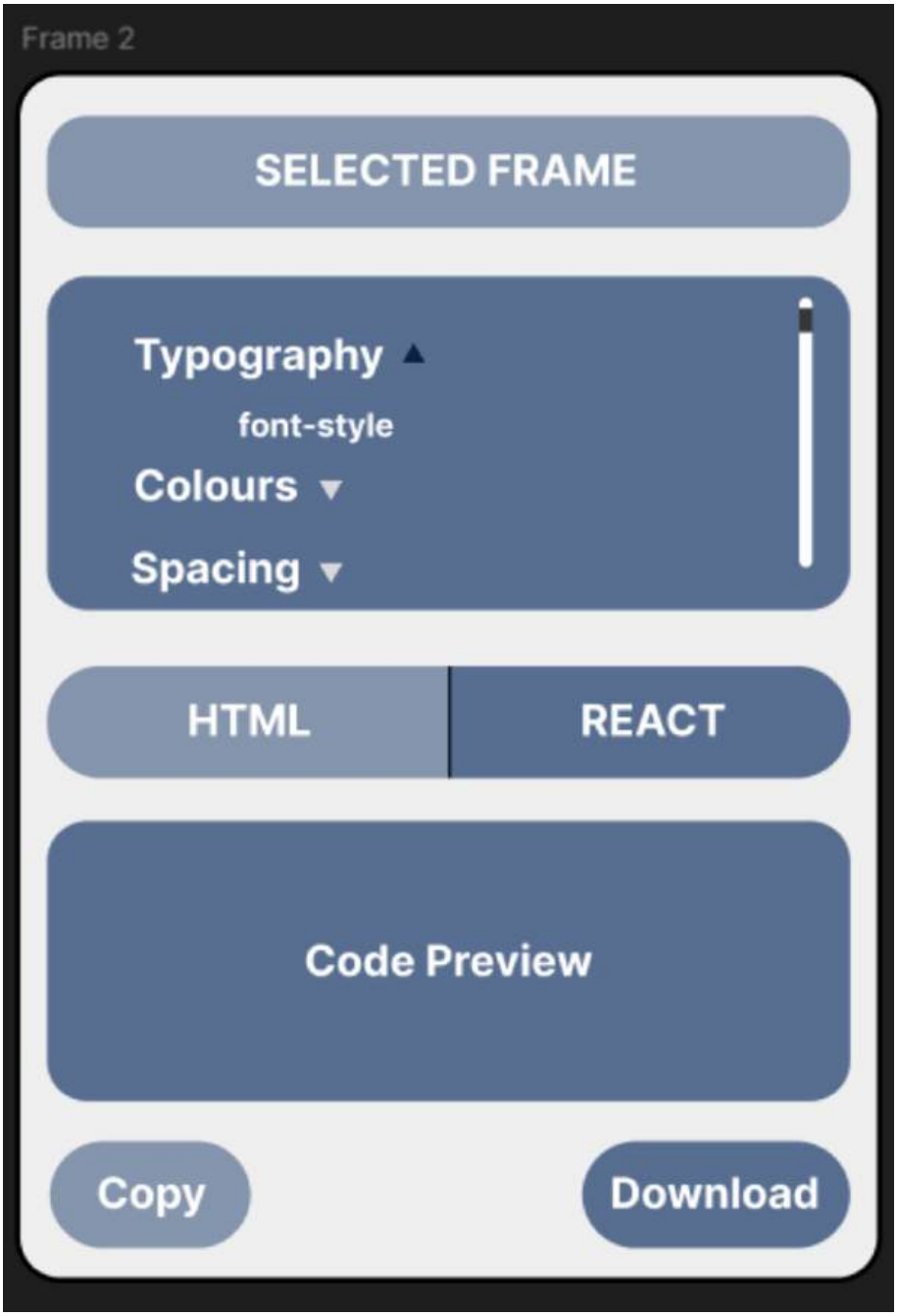


Figure 142 - C.1.2 - Wireframe 2

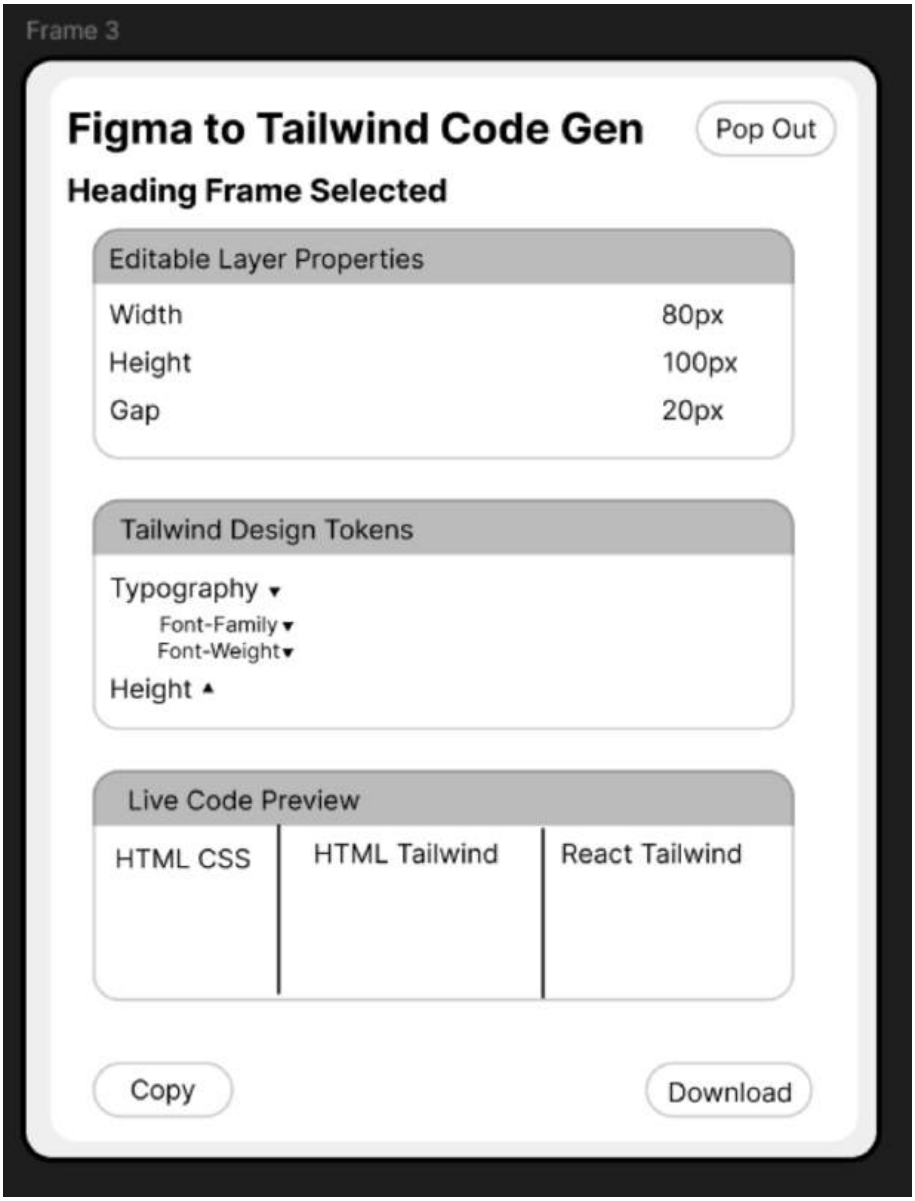


Figure 143 - C.1.2 - Wireframe 3

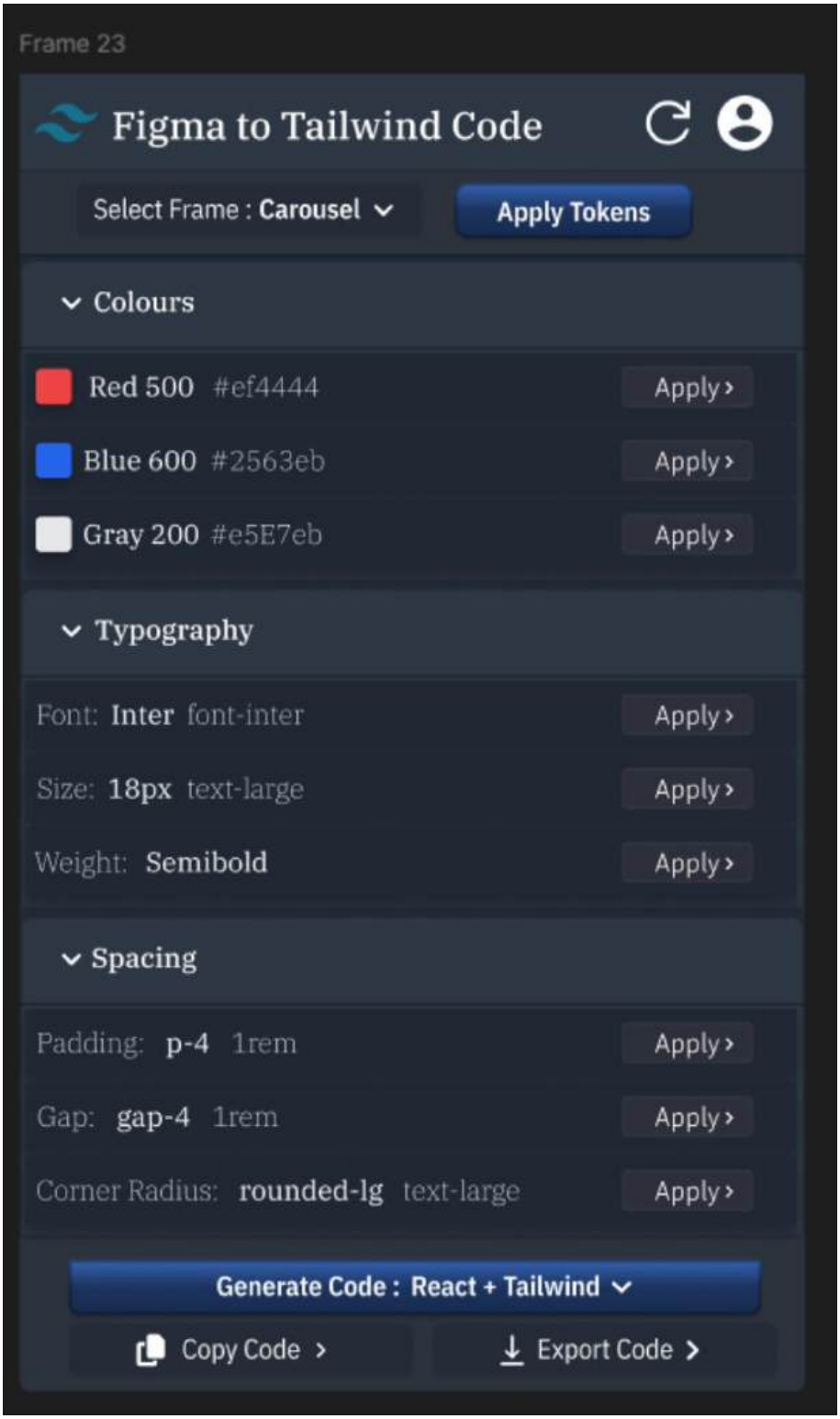


Figure 144 - C.1.2 - Wireframe 4

C.1.3 Use Case Study

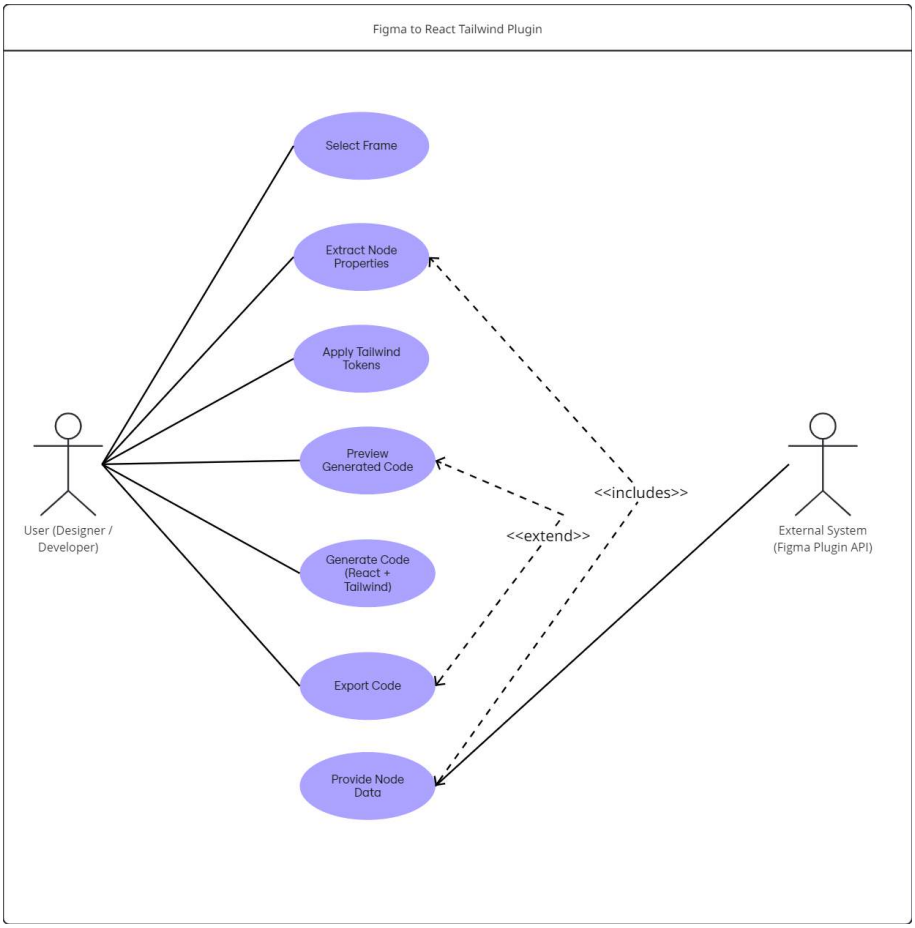


Figure 145 - C.1.3 - Use Case Study

C.1.4 System Architecture

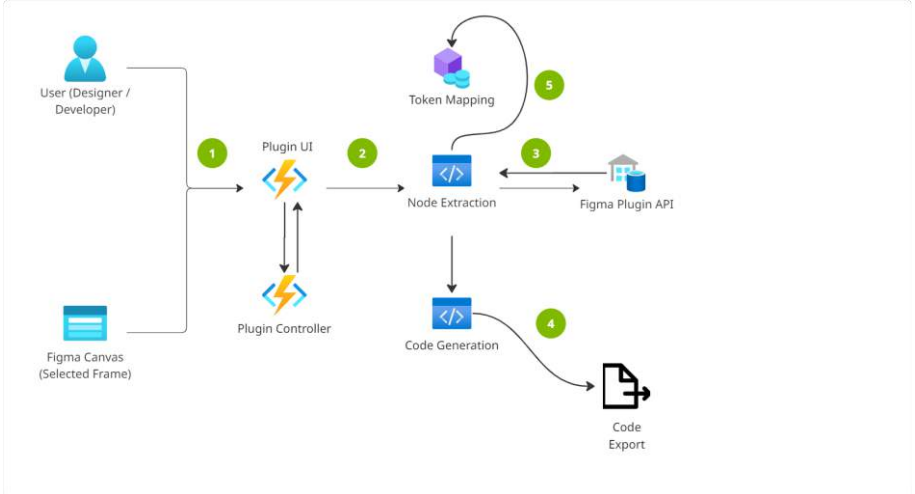


Figure 146 - C.1.4 - System Architecture

C.1.5 User Flow Diagram

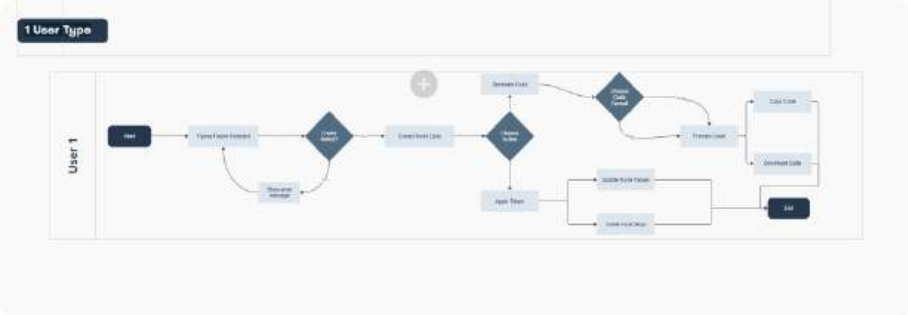


Figure 147 - C.1.5 - User Flow Diagram

C.1.6 Plugin vs No Plugin Workflow

Figure 148 - C.1.6 - No Plugin Workflow

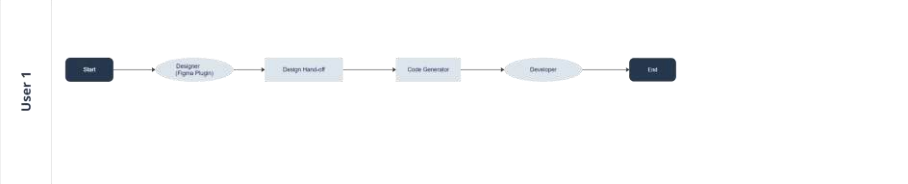


Figure 149 - C.1.6 - Plugin Workflow

C.1.7 Annotated Diagrams

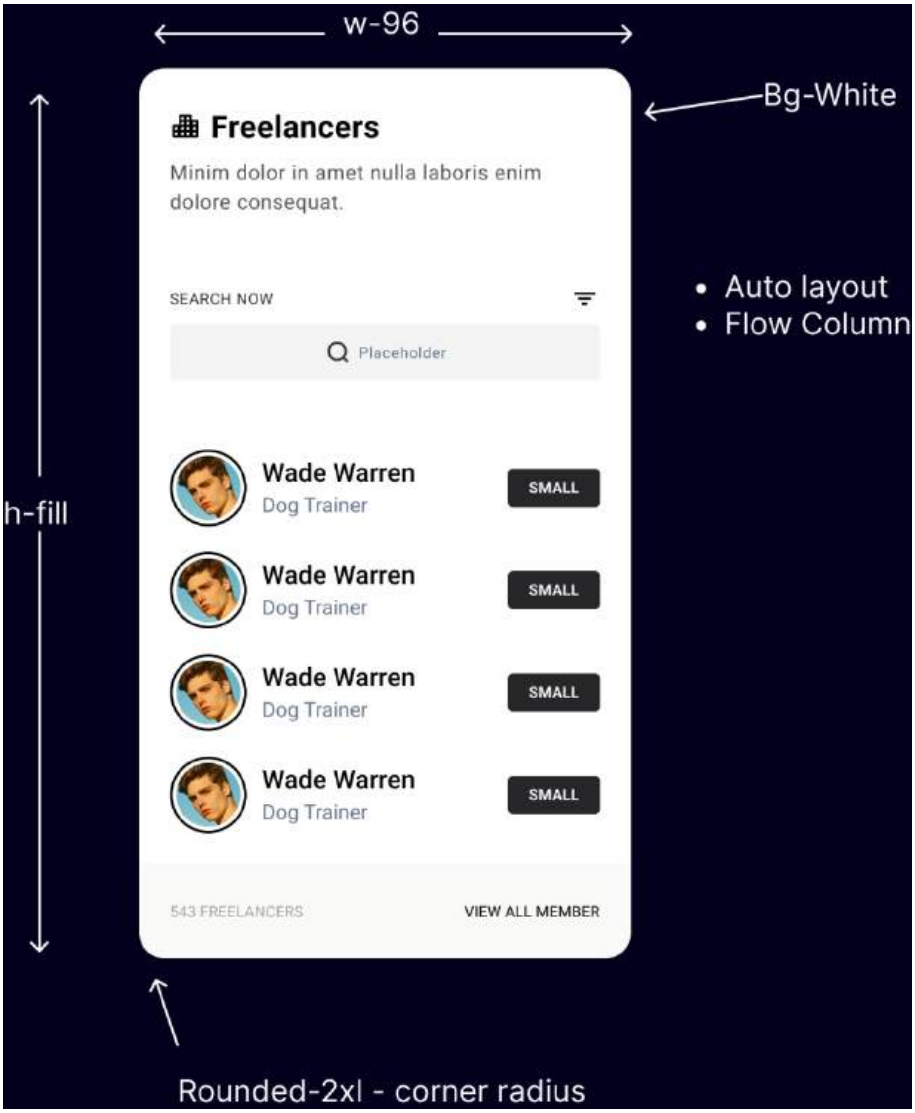


Figure 150 - C.1.7 - Annotated Diagram Card

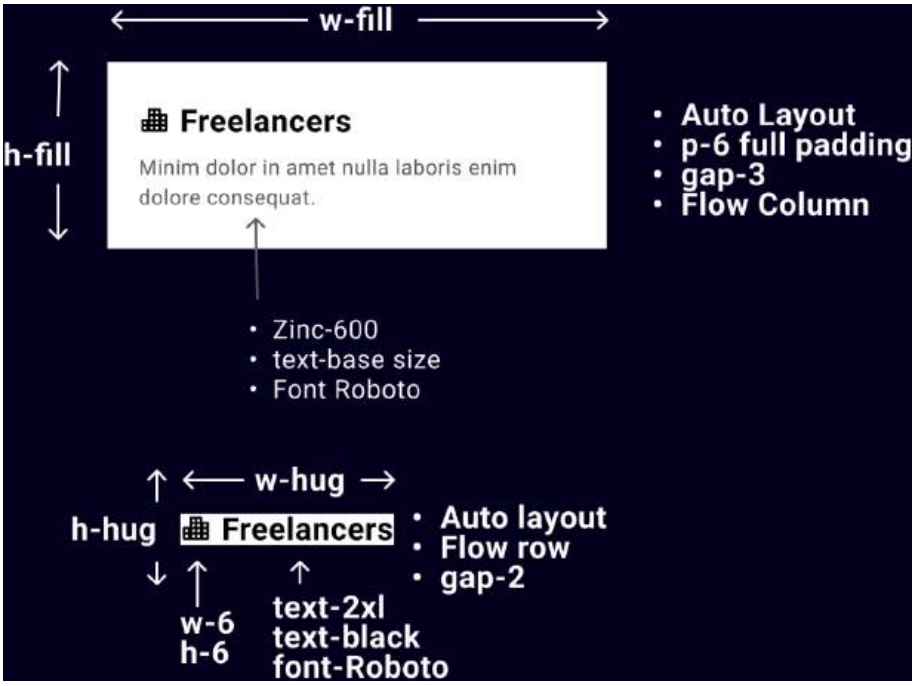


Figure 151 - C.1.7 - Annotated Diagram Header

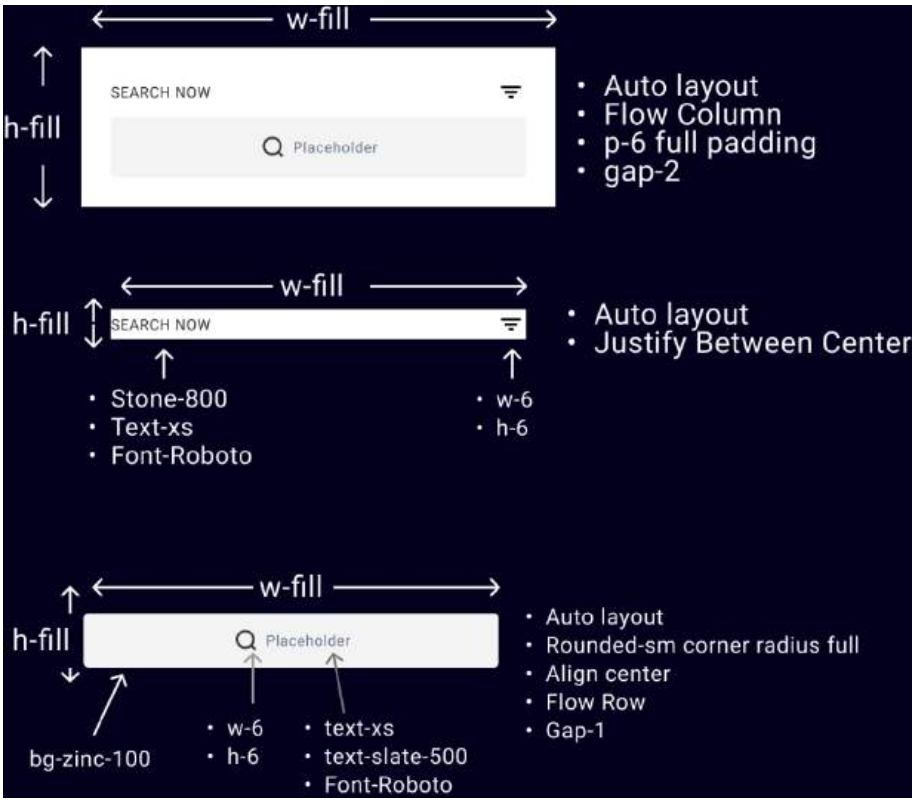


Figure 152 - C.1.7 - Annotated Diagram Search

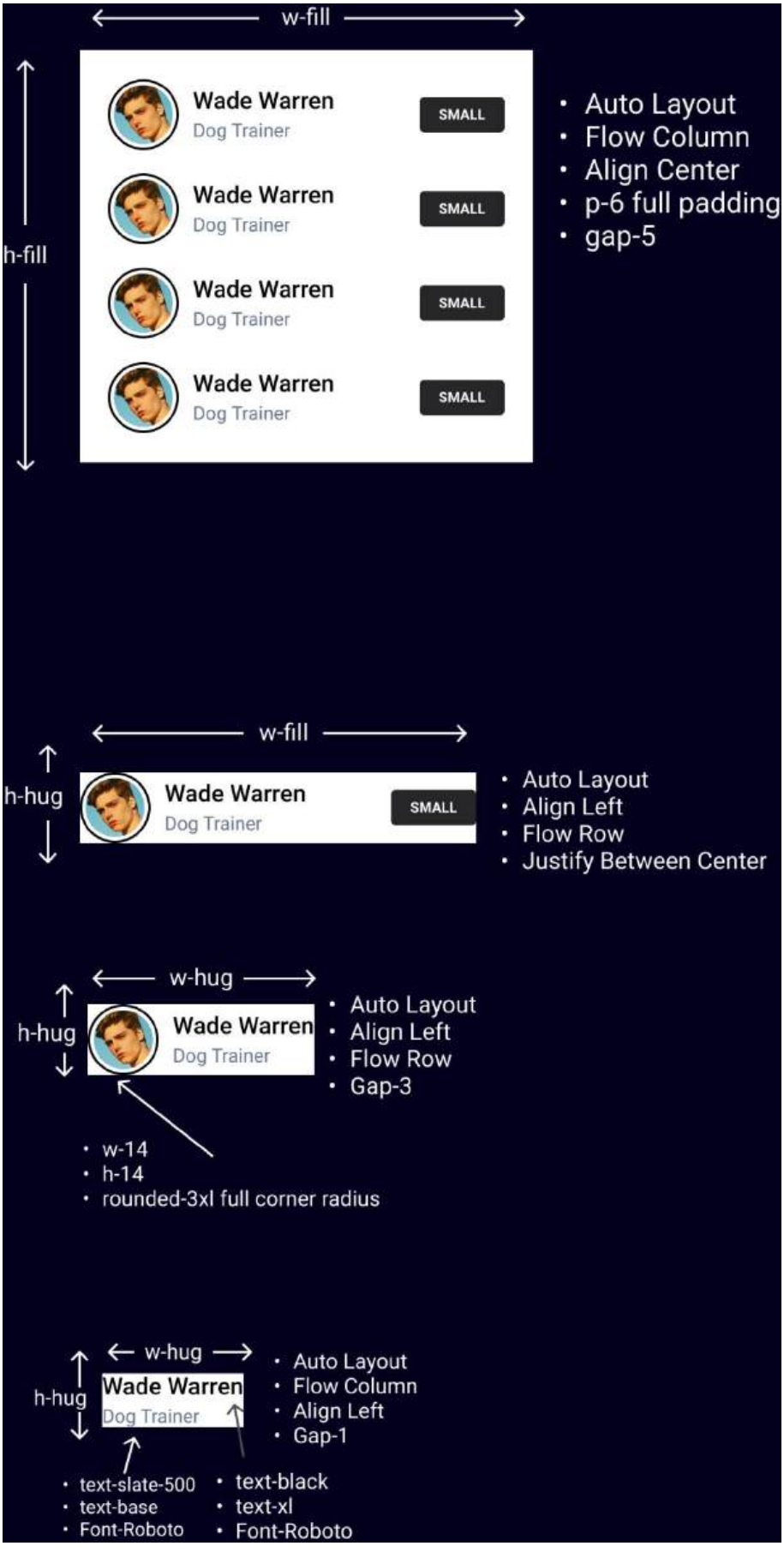


Figure 153 - C.1.7 - Annotated Diagram Image + Text

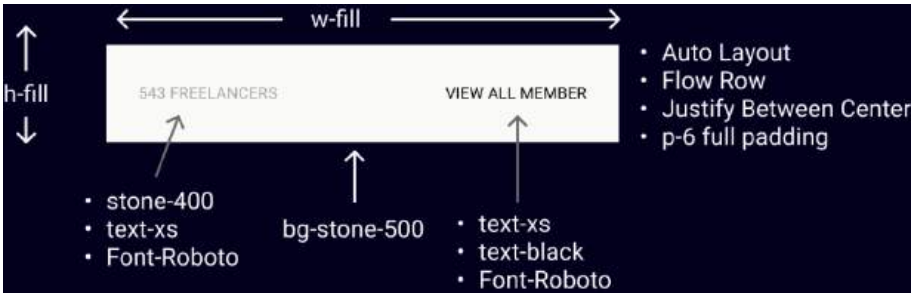


Figure 154 - C.1.7 - Annotated Diagram Tutorial Footer

12.4. Appendix D

12.4.1. Key Code Snippets

D.1.1 Bundling Tools

This section displays the code within the package.json file that includes the use of the ESLint bundlers. The bundler extends to the TypeScript ESLint and the Figma Plugin ESLint.

Figure 155 - D.1.1 - ESLint Bundler Script

Figure 156 - D.1.1 - ESLint Bundler Config

D.1.2 Node Extraction

This section of code snippets demonstrates the process of communicating with the Figma node and extracting the properties requested, including width, height, padding, gap.

```
const pushSelectionUpdate = (node: SceneNode) => {
  const props = extractNodeProperties(node);
  const html = convertNode(node, 0);
  const tailwind = convertNodeTailwind(node, 0);
  const reactTailwind = convertNodeReactTailwind(node, 0);
  figma.ui.postMessage({
    type: "selection-update",
    data: props,
    selectionIds: figma.currentPage.selection.map(
      (selectedNode) => selectedNode.id,
    ),
    html,
    tailwind,
    reactTailwind,
  });
};
```

Figure 157 - D.1.2 - Extraction of Figma Node

```
export function extractNodeProperties(node: SceneNode) {
  const out: any = {
    id: node.id,
    name: node.name,
    type: node.type,
  };

  // geometry / transform properties
  if ("width" in node) out.width = (node as any).width;
  if ("height" in node) out.height = (node as any).height;
  if ("x" in node) out.x = (node as any).x;
  if ("y" in node) out.y = (node as any).y;
  if ("rotation" in node) out.rotation = (node as any).rotation;
  // corner radius - numeric fields
  out.cornerRadius = getCommonRadius(node);
  if ("padding" in node) out.padding = (node as any).padding;
  if ("gap" in node) out.gap = (node as any).gap;
  if ("itemSpacing" in node) out.itemSpacing = (node as any).itemSpacing;
  if ("layoutMode" in node) out.layoutMode = (node as any).layoutMode;
  if ("layoutGrow" in node) out.layoutGrow = (node as any).layoutGrow;
  if ("layoutAlign" in node) out.layoutAlign = (node as any).layoutAlign;
```

Figure 158 - D.1.2 - Extracting Spacing Properties

D.1.3 Code Generation

The code shown below contains the If statement used to display the options for the code generated outputs, the options include HTML + CSS, HTML + Tailwind, React + Tailwind.

```
// === CODEGEN MODE ===
// When Dev Mode asks, return HTML or Tailwind snippets for the selection.
if (figma.editorType === "dev" && figma.mode === "codegen") {
  figma.codegen.on("generate", ({ node, language }) => {
    if (!node) {
      return [
        {
          title: "HTML",
          language: "HTML",
          code: "<!-- No selection -->",
        },
      ];
    }

    if (language === "HTML") {
      return [{ title: "HTML", language: "HTML", code: convertNode(node, 0) }];
    }

    if (language === "TAILWIND_HTML") {
      return [
        {
          title: "Tailwind HTML",
          language: "HTML",
          code: convertNodeTailwind(node, 0),
        },
      ];
    }
```

Figure 159 - D.1.3 - Code Generator HTML + Tailwind

```
    if (language === "REACT_TAILWIND") {
      return [
        {
          title: "React Tailwind",
          language: "JAVASCRIPT",
          code: convertNodeReactTailwind(node, 0),
        },
      ];
    }

    return [{ title: "HTML", language: "HTML", code: convertNode(node, 0) }];
  });
} else {
  // Allow editor UI to stay open
}
```

Figure 160 - D.1.3 - Code Generator React + Tailwind

D.1.4 Figma Plugin API Communication

The code in this section displays examples of the communication method used to send and receive between the plugin system and the Figma Plugin API.

```
figma.ui.onmessage = async (message) => {
  if (!message || !message.type) return;
```

Figure 161 - D.1.4 - Example Communication with Figma API

```
figma.on("selectionchange", () => {
  const node = figma.currentPage.selection[0];

  if (!node) {
    figma.ui.postMessage({ type: "no-selection" });
    return;
  }

  pushSelectionUpdate(node);
});
}
```

Figure 162 - D.1.4 - Figma API PostMessage Communication

D.1.5 Token Registry

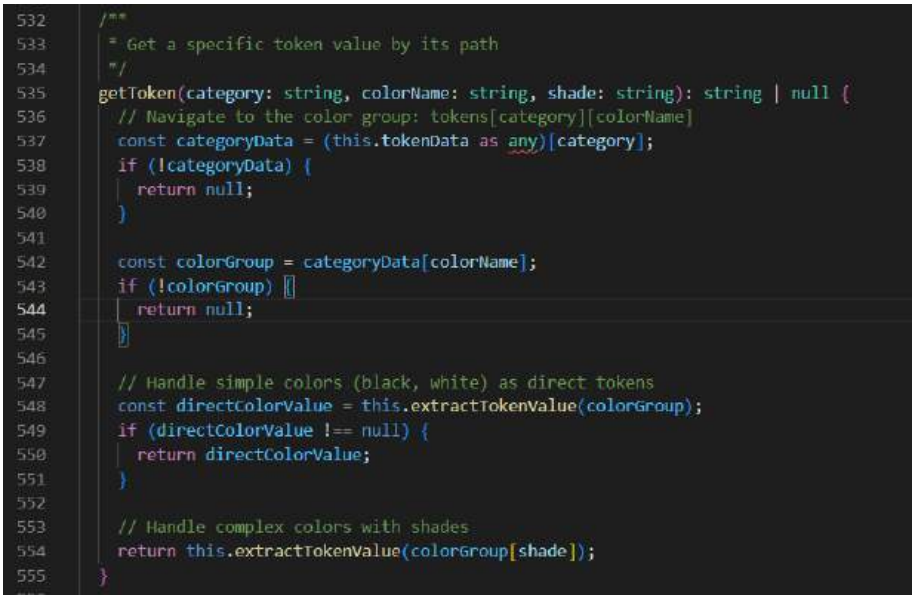
The Token Registry code consists of the Tailwind tokens that are key-value pairs, for example a colour token is in a colour group, the key is the name, the value is the colour shade. The later code demonstrates the communication with the token registry to call stored tokens for matching.

```
1  // === Tailwind Token Registry (raw values) ===
2  const RAW_TAILWIND_TOKENS = {
3    colors: {
4      slate: {
5        "50": "#f8fafc",
6        "100": "#f1f5f9",
7        "200": "#e2e8f0",
8        "300": "#cbd5e1",
9        "400": "#94a3b8",
10       "500": "#64748b",
11       "600": "#475569",
12       "700": "#334155",
13       "800": "#1e293b",
14       "900": "#0f172a",
15       "950": "#020617",
16     },
```

Figure 163 - D.1.5 Token Registry Key-Value Pair

```
/**
 * TokenRegistry manages access to design tokens
 */
export class TokenRegistry {
  private tokenData: typeof TAILWIND_TOKENS;

  constructor(tokens: typeof TAILWIND_TOKENS) {
    this.tokenData = tokens;
  }

  private extractTokenValue(token: any): string | null {
    if (typeof token === "string") {
      return token;
    }
    if (token && typeof token === "object" && "$value" in token) {
      const tokenValue = (token as any).$value;
      return typeof tokenValue === "string" ? tokenValue : String(tokenValue);
    }
    return null;
  }

```

Figure 164 - D.1.5 - Token Registry Extraction

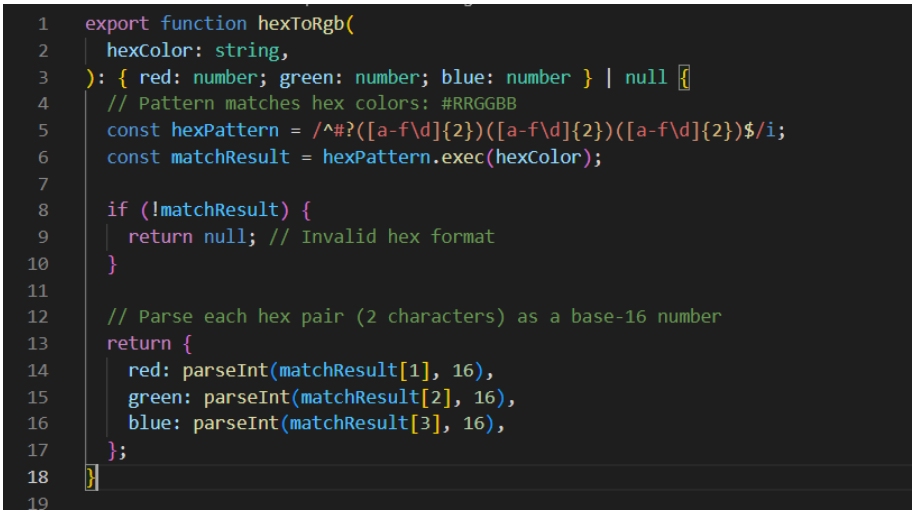


```
  /**
   * Get a specific token value by its path
   */
  getToken(category: string, colorName: string, shade: string): string | null {
    // Navigate to the color group: tokens[category][colorName]
    const categoryData = (this.tokenData as any)[category];
    if (!categoryData) {
      return null;
    }

    const colorGroup = categoryData[colorName];
    if (!colorGroup) {
      return null;
    }

    // Handle simple colors (black, white) as direct tokens
    const directColorValue = this.extractTokenValue(colorGroup);
    if (directColorValue !== null) {
      return directColorValue;
    }

    // Handle complex colors with shades
    return this.extractTokenValue(colorGroup[shade]);
  }
```

Figure 165 - D.1.5 - Token Call Specific

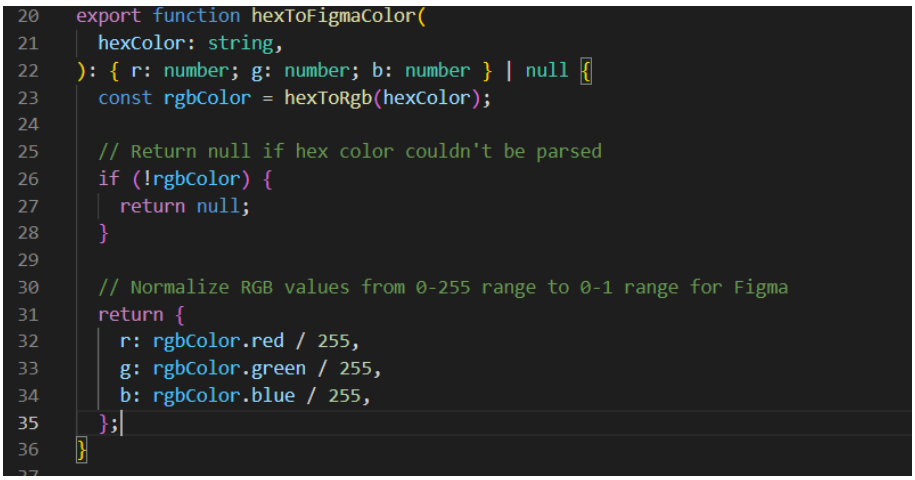
D.1.6 HEX to RGB Algorithm

The algorithms shown are methods to translate colours into HEX codes, as well as HEX into Figma RGB which allows the translated colours to interact with the Figma design environment accurately.



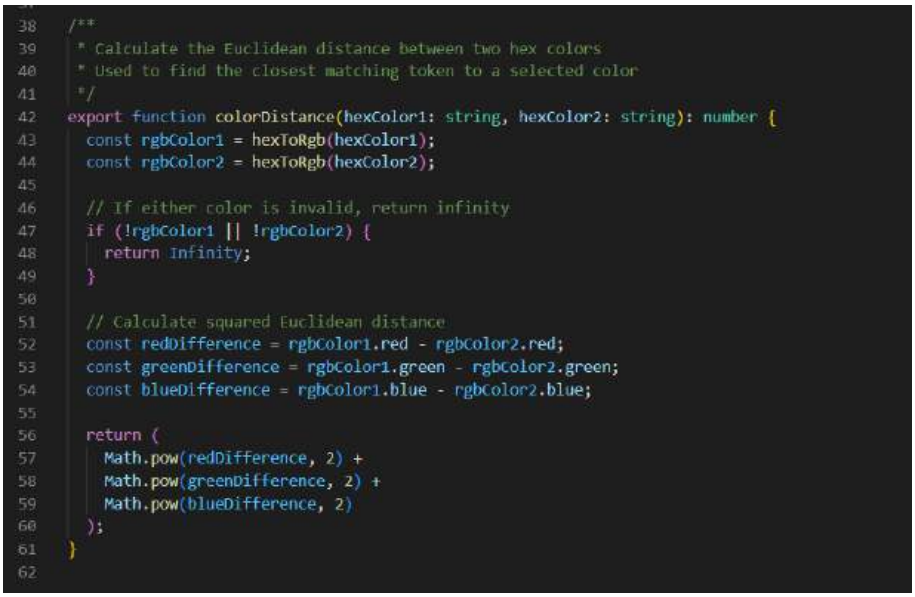
```
export function hexToRgb(
  hexColor: string,
): { red: number; green: number; blue: number } | null {
  // Pattern matches hex colors: #RRGGBB
  const hexPattern = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i;
  const matchResult = hexPattern.exec(hexColor);

  if (!matchResult) {
    return null; // Invalid hex format
  }

  // Parse each hex pair (2 characters) as a base-16 number
  return {
    red: parseInt(matchResult[1], 16),
    green: parseInt(matchResult[2], 16),
    blue: parseInt(matchResult[3], 16),
  };
}
```

Figure 166 - D.1.6 - Parsing Colour to HEX



```
export function hexToFigmaColor(
  hexColor: string,
): { r: number; g: number; b: number } | null {
  const rgbColor = hexToRgb(hexColor);

  // Return null if hex color couldn't be parsed
  if (!rgbColor) {
    return null;
  }

  // Normalize RGB values from 0-255 range to 0-1 range for Figma
  return {
    r: rgbColor.red / 255,
    g: rgbColor.green / 255,
    b: rgbColor.blue / 255,
  };
}
```

Figure 167 - D.1.6 - Colour HEX to Figma RGB



```
/**
 * Calculate the Euclidean distance between two hex colors
 * Used to find the closest matching token to a selected color
 */
export function colorDistance(hexColor1: string, hexColor2: string): number {
  const rgbColor1 = hexToRgb(hexColor1);
  const rgbColor2 = hexToRgb(hexColor2);

  // If either color is invalid, return infinity
  if (!rgbColor1 || !rgbColor2) {
    return Infinity;
  }

  // Calculate squared Euclidean distance
  const redDifference = rgbColor1.red - rgbColor2.red;
  const greenDifference = rgbColor1.green - rgbColor2.green;
  const blueDifference = rgbColor1.blue - rgbColor2.blue;

  return (
    Math.pow(redDifference, 2) +
    Math.pow(greenDifference, 2) +
    Math.pow(blueDifference, 2)
  );
}
```

Figure 168 - D.1.6 - Colour Matching Algorithm

D.1.7 Tailwind Mapping

The mapping of tailwind is showcased below, the code consists of comparing the Figma property naming convention with the Tailwind CSS, allowing for smooth generation of code and interaction with the Figma design environment.

```
export function addNodeSizeClasses(node: SceneNode, classList: string[]): void {
  const size = nodeSize(node as LayoutMixin);

  if (size.width === "fill") classList.push("w-full");
  else if (size.width === null) classList.push("w-auto");
  else classList.push(pxToSize(size.width as number, "w"));

  if (size.height === "fill") classList.push("h-full");
  else if (size.height === null) classList.push("h-auto");
  else classList.push(pxToSize(size.height as number, "h"));
}
```

Figure 169 - D.1.7 - Tailwind Mapping H-Full W-full

```
export function pxToFontSize(px: number): string {
  const tailwindClass = findClosest(px, fontSizeMap);
  if (tailwindClass) return `text-${tailwindClass}`;
  return `text-[${px}px]`;
}

export function pxToLineHeight(px: number, fontSize: number = 16): string {
  const ratio = Math.max(1, Math.min(px / fontSize, 2));
  const rounded = Math.round(ratio * 100) / 100;
  const tailwindClass = findClosest(rounded, lineHeightMap);
  if (tailwindClass) return `leading-${tailwindClass}`;
  return `leading-[${px}px]`;
}

export function pxToLetterSpacing(px: number, fontSize: number = 16): string {
  if (px === 0) return "";
  const em = px / fontSize;
  const rounded = Math.round(em * 1000) / 1000;
  const tailwindClass = findClosest(rounded, letterSpacingMap);
  if (tailwindClass) return `tracking-${tailwindClass}`;
  return `tracking-[${px}px]`;
}

export function pxToSpacing(px: number, prefix: string = "gap"): string {
  if (px === 0) return "";
  const tailwindClass = findClosest(px, spacingMap);
  if (tailwindClass) return `${prefix}-${tailwindClass}`;
  return `${prefix}-[${px}px]`;
}

export function pxToPosition(px: number, prefix: string = "left"): string {
  if (px === 0) return "";
  return `${prefix}-[${px}px]`;
}
```

Figure 170 - D.1.7 - Conversions of Figma to Tailwind

12.5. Appendix E

12.5.1. Sprint Logs

This appendix displays the sprints from the early stages of development and planning to the late stages. The early stages prioritise researching the scope of the project and generating the basic communication between the plugin and the Figma Plugin API. The middle stage sprints consist of implementing the functionality into the plugin, the evaluation of the requirements is performed during this stage. The late stage sprints display the testing of the application and through the feedback iterations are made to the functionality.

E.1.1 Early-Stage Sprints

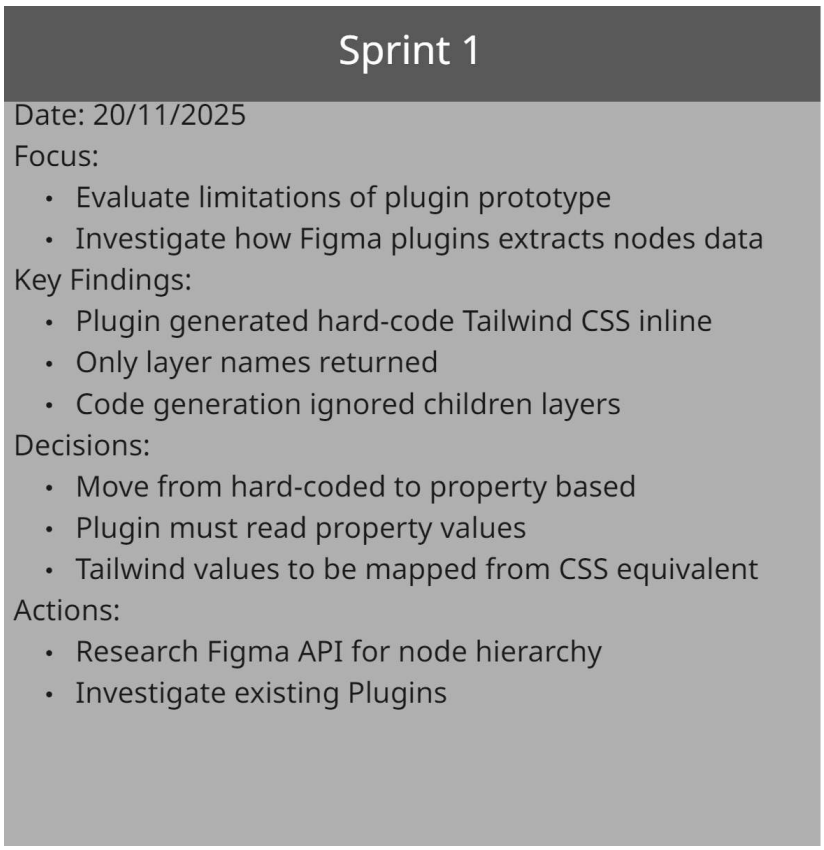


Figure 171 - E.1.1 - Sprint 1

Sprint 2

Date: 27/11/2025

Focus:

- Improve HTML generation and node property extraction.

Key Findings:

- Node variables retrieved but UI returned hard-coded names
- Generated absolute position
- Figma lacked CSS properties like Margin

Decisions:

- Replace hard-coded variables with retrieved node properties
- Create Margin calculation script
- Explore alternative positioning methods

Actions:

- Research Figma node extraction
- Prototype Margin script
- Implement UI for plugin to message Figma.
-

Figure 172 - E.1.1 - Sprint 2

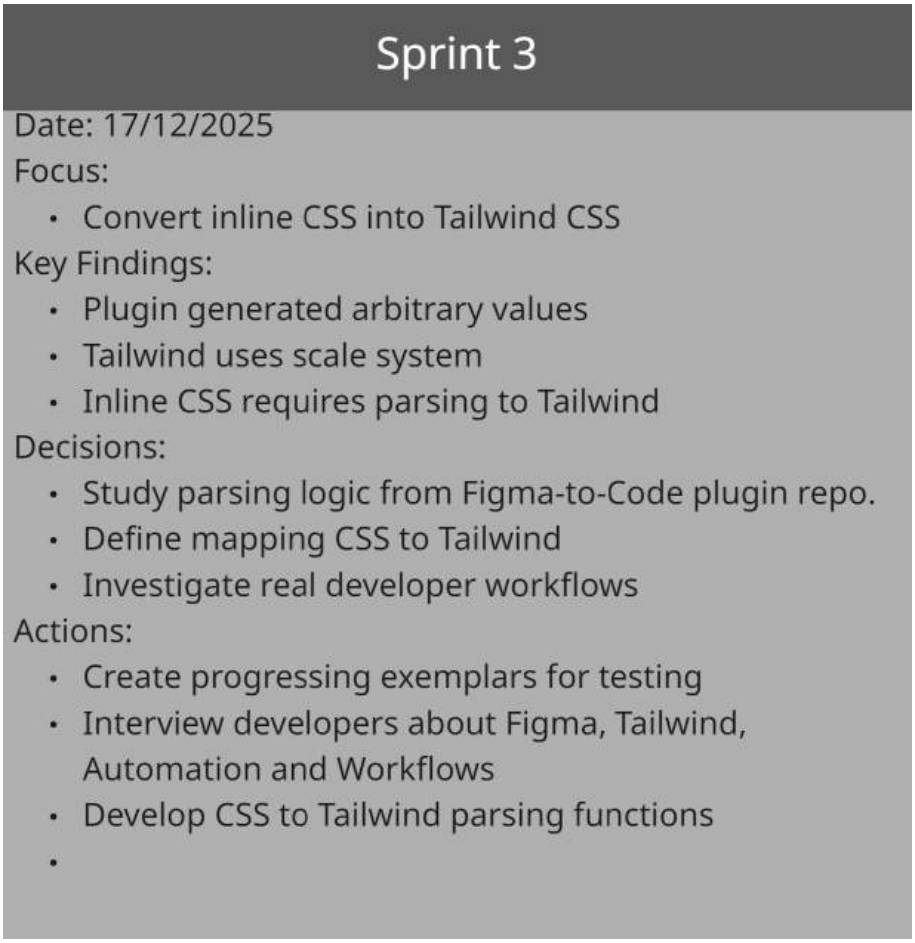


Figure 173 - E.1.1 - Sprint 3

Sprint 4

Date: 13/01/2026

Focus:

- Understand developer workflows and plugin usability

Key Findings:

- Survey responses collected from students and industry developers.
- Auto-layout structure for consistent code generation
- Open-source plugins provide parsing logic.

Decisions:

- Conduct surveys and interviews with developers
- Improve exemplars with auto-layout
- Clarify plugin goals for long-term guidelines.

Actions:

- Conduct surveys
- Document goals and proposal
- Expand parsing to auto-layout
-

Figure 174 -E.1.1 -Sprint 4

E.1.2 Mid-Stage Sprints

Sprint 5

Date: 30/01/2026

Focus:

- Improve plugin workflow and investigate token systems

Key Findings:

- Plugin lacked visual feedback when values were changed
- Tailwind CSS could be tokenized for design usage
- Plugin workflow required editing options

Decisions:

- Explore tokenization of Tailwind CSS
- Redesign plugin UI to mirror Figma panel
- Remove absolute positioning layouts

Actions:

- Research tokenization methods for Tailwind CSS
- Redesign plugin Ui panel for Figma editing.
-

Figure 175 - E.1.2 - Sprint 5

Sprint 6

Date: 12/02/2026
Focus:

- Shift plugin concept towards design token workflow

Key Findings:

- Plugin UI duplicated with Figma panel usage
- Design tokens provided better workflow model
- Studied Kigen UI token system

Decisions:

- Pivot plugin concept to token-based workflow
- Preload Tailwind CSS classes as tokens
- Allow tokens to be applied to frames

Actions:

- Research design token structure & storage
- Investigate preloading tokens in plugin start-up
- Develop method to apply token to Figma nodes.
-

Figure 176 - E.1.2 - Sprint 6

Sprint 7

Date: 26/02/2026

Focus:

- Standardising token system
- Expanding coverage of tokens
- Improving plugin usability

Key Findings:

- Token system works but lacks W3C formatting
- Token coverage is incomplete across design fields
- No efficient navigation method
- Typography is limited without full Figma API access

Decisions:

- Adopt W3C formatting scheme for consistency and industry standard
- Expand token system to include more design fields
- Implement a search bar for token filtering
- Integrate available token fonts through Figma API

Actions:

- Research Figma API for node hierarchy
- Investigate existing Plugins
-

Figure 177 - E.1.2 - Sprint 7

Sprint 8

Date: 12/03/2025

Focus:

- Implemented W3C tokenization
- Expanded on available tokens to include spacing and typography
- Added search functionality

Key Findings:

- Tokens lack layout values, effects are missing, gradients are missing
- Typography was accessible through Figma environment

Decisions:

- Increase token coverage of design
- Plugin must be navigational with filter options

Actions:

- Research tailwind tokens for layout and effects
- Integrate search and filter functionality
-

Figure 178 - E.1.2 - Sprint 8

E.1.3 Late-Stage Sprints

Sprint 9

Date: 20/03/2025

Focus:

- Implement custom token logic for colours and gradients
- Add missing effects and layout options such as auto-layout
- Review pre-designed cards on Figma community

Key Findings:

- Effects translate to Tailwind inaccurately for stroke position
- Auto-layout automatically assigns flex

Decisions:

- Test making designs through use of plugin
- Implement final missing token logic for layout

Actions:

- Create designs with plugin
- Follow guides to create auto layout designs
-

Figure 179 - E.1.3 - Sprint 9

Sprint 10

Date: 30/03/2025

Focus:

- Preparations for User testing
- Create user testing documentation
- Alter UI for testing

Key Findings:

- User testing focuses on token designing
- Testing environment is required for use cases

Decisions:

- Create a testing design through Figma cards
- Explore UI options for clarity
- Research digital user testing

Actions:

- Prepare user test case studies
- Create test designs for users to follow
- Create annotated diagrams of test designs
-

Figure 180 - E.1.3 - Sprint 10

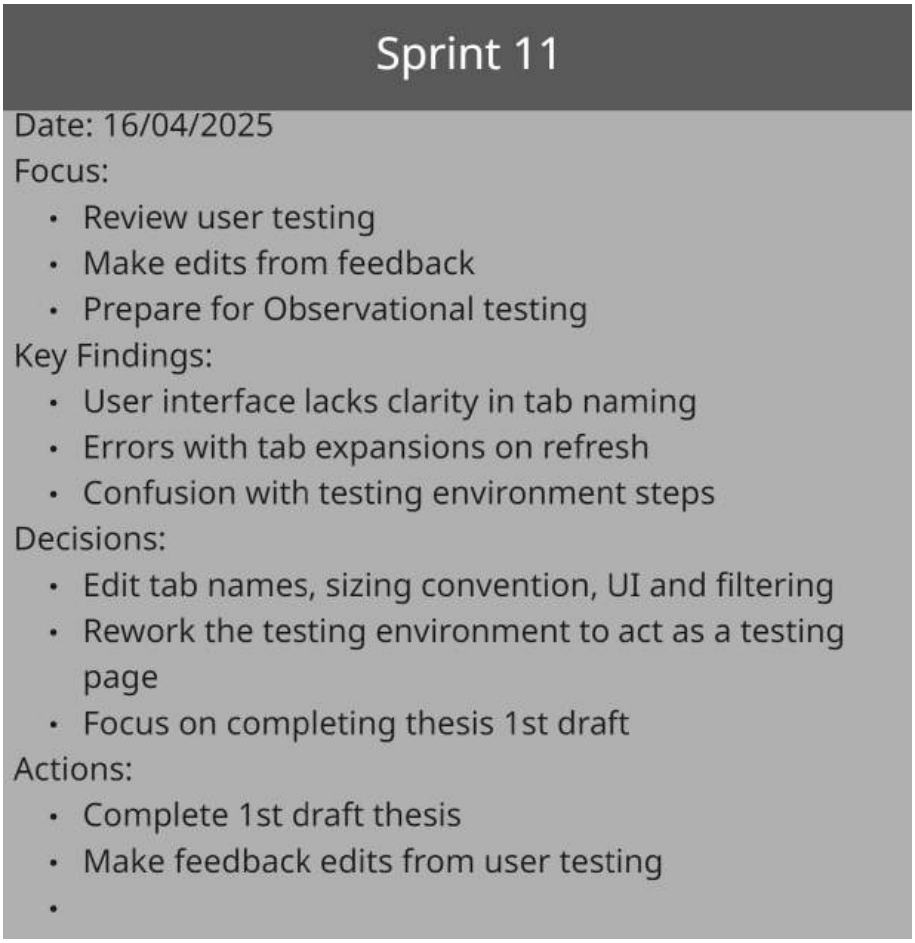


Figure 181 - E.1.3 - Sprint 11

12.6. Appendix F

12.6.1. Additional Materials

F.1.1 User Testing Consent Form

[redacted]

iadt
DUN LAOGHAIRE

Figure 182 - F.1.3 - User Consent Form Interactable Document

F.1.2 User Task Sheet

User Case Study – Figma-To-Code Plugin

Objective

The objective of this case study is to evaluate how effectively users can recreate a design card in Figma using the Plugin. This study focuses on whether users can understand the plugin's token-based workflow for styling, spacing, and layout design, and if users can apply these stylings to build a layout that matches a provided reference design.

Overview

Participants are provided with a Figma file containing an annotated reference design. The task is to recreate the design card inside the same file using the plugin. The case study is designed to observe how users:

- Navigate the plugin
- Apply typography, spacing, colour and layout tokens
- Use auto layout styles
- Translate the annotated diagram into a full design build

Setup

1. Install prerequisites

Node.js (LTS): install at https://nodejs.org/

Figma Desktop App: install at https://www.figma.com/downloads/

2. Clone the Repository in terminal

- SSH:
  git clone
  git@github.com:MatthewDent03/CodeGenTestPlugin.git

Figure 183 - F.1.2 - User Task Interactable Document

F.1.3 Repository

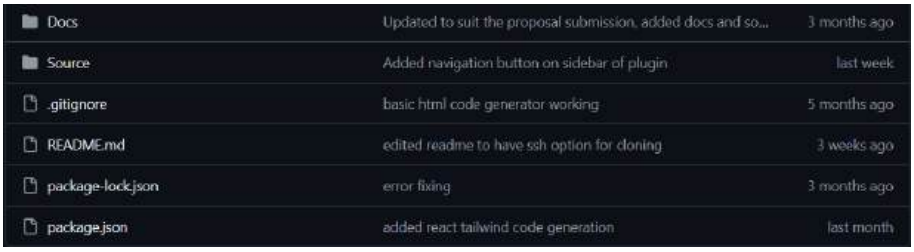


Figure 184 - F.1.3 - Repository Folder Structure

Figure 185 - F.1.3 - Repository Source Files

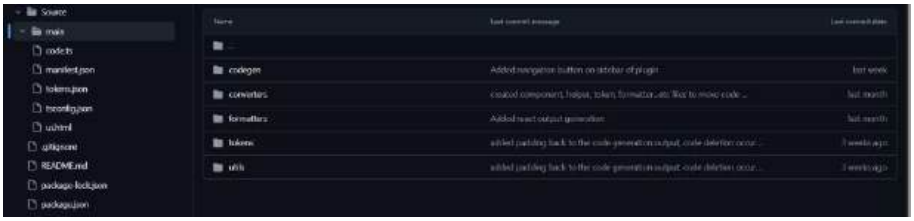


Figure 186 - F.1.3 - Repository Main Folders

Figure 187 - F.1.3 - Codegen Files

Figure 188 - F.1.3 - Repository Converter Files

Figure 189 - F.1.3 - Repository Formatter Files

Figure 190 - F.1.3 - Repository Token Files

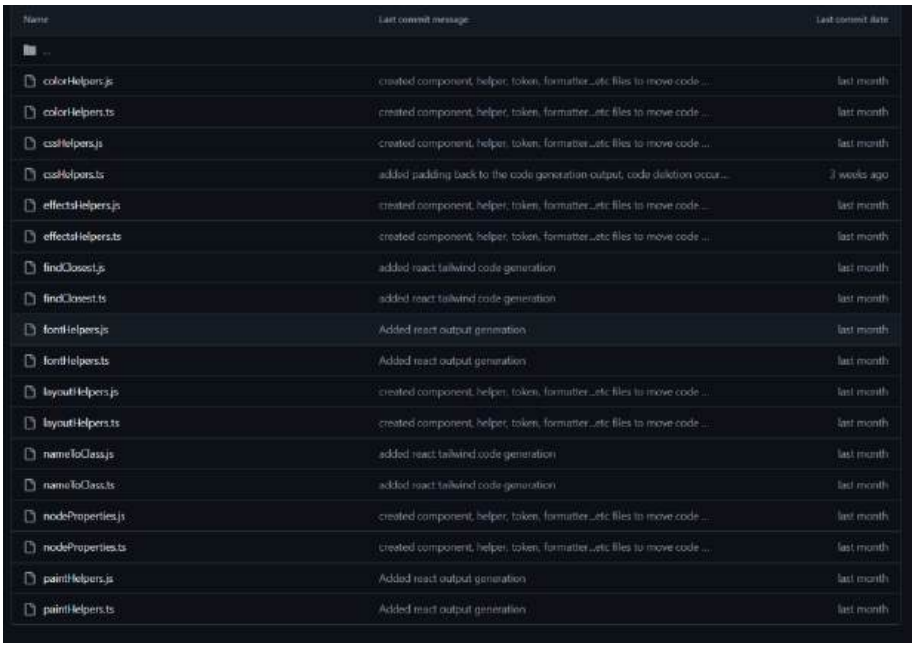


Figure 191 - F.1.3 - Repository Utility File

F.1.4 Observational Notes

Observation Notes:

Tabs & Navigation:
- Tabs don't open fully
- Tabs don't expand correctly when docked
- Reopening tabs is slow and disrupts workflow
- Tab labels are unclear or misleading
- Tab naming is inconsistent and needs simplification

Search Functionality:
- Search input persists across tabs
- Search doesn't support full token (stone-800)
- Search doesn't support special characters (-)
- Search appears in irrelevant sections

Layout & UI Behaviour:
- Plugin resizing is inconsistent
- Tabs don't scale properly when resized
- Collapsibles don't work when searching
- Typography section is overcrowded, needs dropdown menu

Functionality & Features:
- Undo button breaks and performs multiple undo actions
- No manual input support for advanced users
- Can't apply multi-side values (p-left & p-right)
- No visual indicator for applied token
- Hug/Fill not working on frames other than auto layout
- Auto layout naming convention is unhelpful

Accessibility:
- Low contrast UI (too many greys overlapping)
- Poor colourblind accessibility
- Success notifications lack visual feedback

Figure 192 - F.1.4 - Observational Notes

Survey Notes:

Participant Background:
- Mixed experience levels with Figma (beginner - advanced)
- Limited familiarity with design tokens
- Varied confidence levels at task start

Task Completion:
- All users completed the task somewhat
- No complete failures
- Indicates core functionality is working

Time & Efficiency:
- Completion times varied between users
- Slower times connected to - navigation confusion - searching for correct options
- Workflow interruptions reduced efficiency

Ease of use:
- Moderate usability
- Users could complete tasks but with friction
- Key areas - Finding correct tabs - Understanding features - Repeating actions across sections

Key Improvements:
- Simplify navigation to improve workflow
- Clarify search behaviour
- Add clearer visual feedback

Navigation & Structure:
- Users struggles to locate features
- Tab system caused confusion
- Switching between sections disrupted workflow

Search Experience:
- Search functionality caused confusion
- Users unsure of where search applies & how results filter

Feature Understanding:
- Tokens:
  - Generally understood after usage
  - Lack of visual feedback when applied
- Auto Layout:
  - Larger confusion
  - Unclear of its usage

Accuracy of Output:
- Final designs were mostly accurate
- Logic is correct
- Issues mainly persist in UX/UI

Pain Points:
- Navigation and tab structure
- Search behaviour
- UI inconsistencies

Figure 193 - F.1.4 – Survey Notes