Orbit: A Web Application Exploring Alternative Expressive and Accessible Web Design

Emma Ann Hynes

N00222012

Supervisor: Stefan Paz Berrios

Second Reader: John Montayne

Year 4 2025-26

DL836 BSc (Hons) in Creative Computing

Major Project

Abstract

Over the past two decades, the internet has become deeply embedded in our everyday lives. While early web spaces were often personal, experimental, and expressive, many contemporary websites and applications now follow increasingly standardised layouts, interaction patterns, and visual styles.

This project responds to concerns around web homogenisation, centralisation, algorithmic influence, and the decline of personal expression online. It asks whether standardisation is an unavoidable result of digital progress, or whether online spaces can still support creativity, individuality, and user autonomy.

To explore this, I designed and developed Orbit, a responsive web-based application that allows users to create personalised digital canvases through draggable, resizable, and customisable components.

Acknowledgments

I would first like to thank my family for supporting me throughout my journey in IADT. Whether it was driving me to and from my part-time jobs while I saved for a car or making sure I had enough food to get through the week, I genuinely would not have been able to complete this course without their support.

I would like to thank my supervisor Stefan Paz Berrios, for his guidance throughout this project. From the earliest stages of development through to its completion, his advice consistently helped steer me in the right direction and supported me throughout the process.

I am also grateful for my friends, those from before college and those I met during my time in IADT, for their support not just during not just the development of Orbit, but the entire four years.

I would also like to thank everyone who completed my surveys, tested the application and gave feedback. Their comments helped shape Orbit into what it is and watching everyone enjoy using the application was truly the best part of this entire process.

Lastly, I would like to thank my boyfriend, Ryan. His support and encouragement made a huge difference during the more stressful parts of developing Orbit, such as uncertainty about the design and bugs during the development process. Thank you for letting me complain at you, I probably wouldn't have gotten very far otherwise!

Thank you to everyone who was involved in making this happen!

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

- Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Emma Ann Hynes

Signed :

Failure to complete and submit this form may lead to an investigation into your work.

1. Introduction and Project Context

1.1 Aim & Objectives:

1.1.1 Final Aim

This project aims to explore the homogenisation of contemporary web design and considers how digital spaces might instead support individuality, autonomy, and creative self-expression. In recent years, many websites and social platforms have become increasingly standardised in both their visual language and interaction patterns. While this standardisation often improves usability and familiarity, it can also reduce opportunities for personalisation, experimentation and creativity.

In response to this, this project aims to design and develop a responsive web-based application for desktop that allows users to create a personalised digital canvas through draggable, resizable, and customisable card components. These components will support a range of content types, including text, images, GIFs, and links, enabling users to build a space that reflects their identity, interests, and creative preferences and share them easily with their friends and family. It will work similarly to a scrapbook or a pinboard, allowing users to stick what they want where they want.

The project is situated within a broader critique of current web and social media design practices. It engages with the ethical concerns surrounding addictive interface design, including the use of dark patterns and algorithmic systems that prioritise engagement over user wellbeing and autonomy. In this context, the application acts not only as a functional design outcome, but also as a response to the increasing control that platforms exert over how users interact, express themselves, and consume content online.

1.1.2 Objectives

- Investigate the homogenisation of contemporary web design and its impact on creativity, identity, and self-expression online.
- Examine ethical issues in modern web and social media design, including dark patterns, addictive design practices, and algorithmic control.
- Explore the historical evolution of social media and web platforms towards increasingly standardised interfaces.
- Research how accessibility can be maintained within more expressive and experimental approach to web design.
- Design and develop a responsive web-based application for desktop and mobile devices.
- Create a system of draggable, resizable, and customisable components that support different content types, including text, images, GIFs, and links.
- Provide users with a personalised canvas that encourages autonomy, creativity, and ownership over their online self-presentation.
- Evaluate whether expressive and visually distinctive websites can coexist with accessibility and usability.

1.2 Success Criteria & Scope

1.2.1 Success Criteria

The success of the project will be evaluated using functional technical, and usability-based criteria. The system should successfully allow users to create, customise and share a personal digital canvas using interactive draggable components.

It should also aim to function responsively across desktop and mobile devices and support multiple content types without loss of performance or stability.

1.2.2 Project Scope and Limitations

This project is not intended to function as a full social media platform and does not include features such as algorithmic content feeds, social networking systems, or content recommendation.

There may also be technical limitations in relation to performance when many components are displayed on the canvas such as reduced frame rate and slower rendering.

The application will aim to be fully responsive across desktop and mobile devices and ensure that the core idea of dragging, resizing and adding components remains across different screen sizes. However, touch based interaction on mobile requires adapted gesture handling and has reduced precision compared to the mouse input of a desktop or laptop, which may cause issues in terms of accuracy.

Finally, accessibility considerations are a priority and will be implemented at a foundation level, however full use of advanced technology standards such as WCAG AAA compliance will be beyond the scope of this application

2. Research and Background:

2.1 Literature Review

2.1.1 Similar Applications

To understand how existing platforms support self-expression, content organisation and visual customisation, a range of similar application were analysed. This helped identify both successful features and key limitations in current systems and informed the design direction of the application.

These have been chosen based on how similar they are to the application idea and how each represent a different approach to organising content, presenting user identity, or building an online presence.

Kinopio

Kinopio is a visual thinking tool designed by Piri using Vue and Node.js which allows users to organise ideas spatially through cards, links, and media.

Its main strength is the way it encourages the exploration of ideas rather than structure, which is more in alignment with how people think. It gives users a flexible space where ideas can be arranged visually, instead of forcing users into predefined layouts. It is relevant to this project due to its emphasis on creative freedom and the sense of ownership users have over how their information is arranged.

However, it is mainly for brainstorming and idea mapping rather than a personal identity or public facing profile page and as a result, it doesn't address how users might present themselves or curate a lasting digital presence.

Despite this limitation, Kinopio is still relevant from a technical and philosophical perspective. In its engineering principles, the creator emphasises performance, simplicity, and long-term maintainability. For example, the following principles are highlighted:

“Speed first, perfect later. You shouldn’t need to wait for server requests to complete before editing your spaces. Because fast software is the best software.”

“Favor easy maintenance. To help you sleep at night choose open technologies that are widely supported and well documented.”

“Building a foundation for decades. This mindset encourages writing small, simple code that is easy to re-read in the future.”

These principles influenced this project by reinforcing the importance of responsive interaction, lightweight design decisions and maintainable system architecture. In particular, the way cards behave and are designed, which feel immediate and fluid served as key inspiration. The persistence of card positions and content across sessions, combined with the smooth and uninterrupted interaction informed the decision to implement a similar state preserving card system within the project

Overall, Kinopio directly informed both the interaction design and underlying philosophy of this project

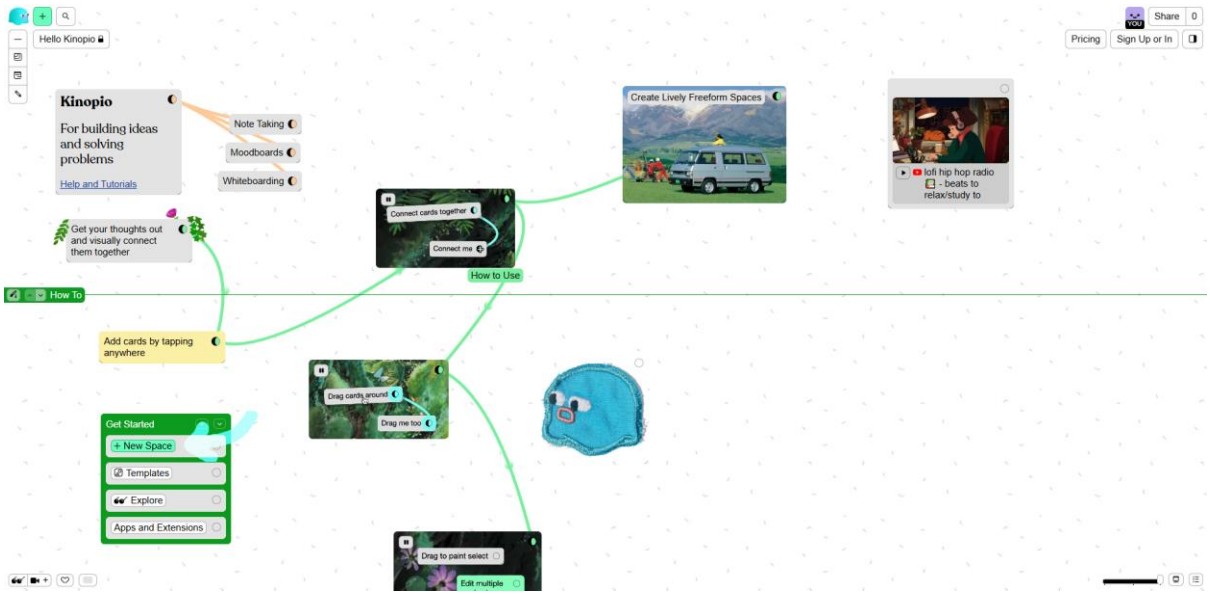


Figure 1 Kinopio

Instagram

Instagram is a free social media platform owned by Meta, created in 2010 by Kevin Systrom and Mike Krieger. It is one of the most widely recognised platforms for online presentation, allowing users to share images and videos in a unified interface.

Although Instagram supports personal expression, this expression takes place within a highly standardised system. All profiles follow the same structural layout, and content presentation is constrained due to this.

An advantage to Instagram is its simplicity and familiarity. The platform is highly accessible, visually consistent, and easy to navigate, which contributes to its widespread adoption. However, this ease of use and simplicity is achieved through the trade-off of reduced user autonomy as users are limited in how they structure their profile.

Following this, research highlights that unlike traditional media, the platform does not include natural stopping cues that encourage users to disengage, instead continuously

presenting new content that encourages prolonged scrolling behaviour. (Abrams, 2022). This design structure leads to users to remain engaged for extended periods without conscious decision points to stop.

In addition to this, Qiu (2024) argues that social media platforms intensify social comparison by exposing users to curated versions of other people's lives, where success, beauty, lifestyle and popularity are repeatedly displayed. This is particularly relevant to Instagram, as the platform is built around visual self-presentation and measurable engagement through likes, followers, comments and views. These features can encourage users to compare their own lives and identities against idealised online images. Qiu also highlights the role of algorithmic curation and influencers, who often become benchmarks of aspiration and success.

This reinforces the idea that Instagram is not only restrictive in terms of design but also shapes how users perceive and present themselves within digital environments.

Overall, Instagram demonstrates how mainstream social media platforms prioritise consistency, scalability, and engagement over individual control and expressive freedom. This directly informed the direction of this project, which deliberately avoids algorithmic systems and social feeds. Instead, the focus is placed on a user-controlled, non-hierarchical canvas where individuals have full control over layout, structure, and presentation.

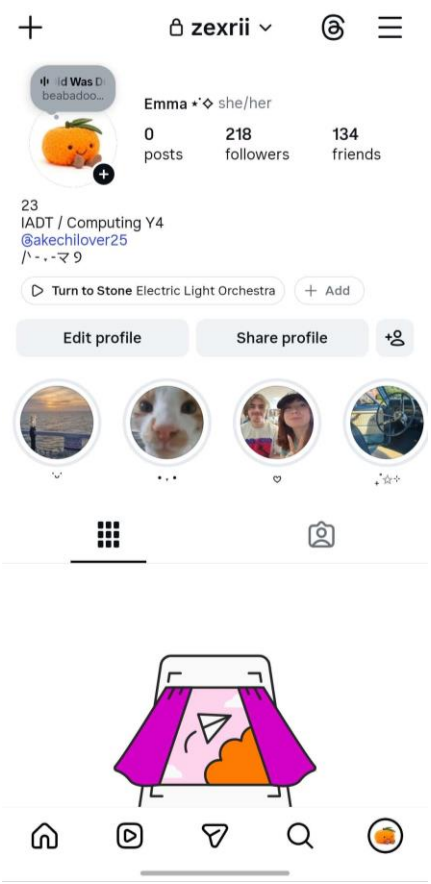


Figure 2 Instagram Profile

SquareSpace

Squarespace is an All-in-One Platform that includes website hosting, domain registration, and security (SSL) as a part of its subscription plans. It is designed to allow users to create professional websites, portfolios and online stores without coding knowledge.

Its strength lies in its high-quality templates, branding options, and accessible editing tools. The UI for customising and editing the profile is quick and easy to understand. However, its customisation still takes place within template-based systems, meaning the user is guided through a structured design process rather than being given complete creative freedom.

An advantage of Squarespace is that it balances usability with a degree of personalisation. A disadvantage is that its templates can still produce formulaic results, and the experience may feel more curated than expressive.

While it succeeds in making website creation accessible, it also highlights the limitations of structured template systems in supporting fully open-ended creative

expression. This directly informed the development of this project, particularly the decision to move away from fixed layout systems in favour of a more flexible, user-driven canvas environment.

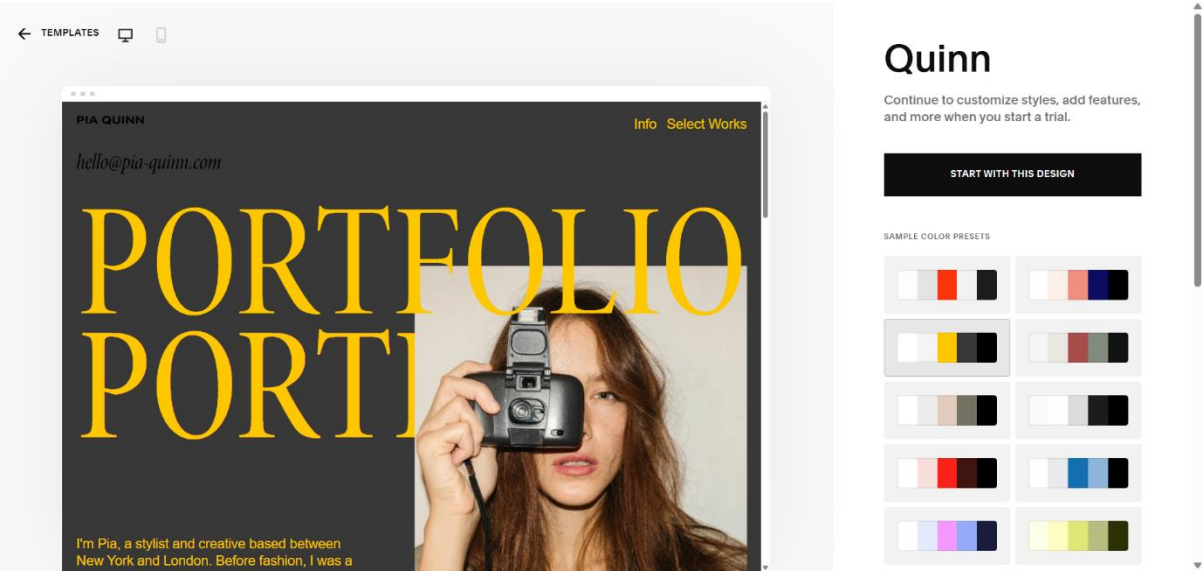


Figure 3 Squarespace

Neocities

Neocities is a free static website hosting service launched in 2013 by Kyle Drake. It was designed as a spiritual successor to the 1990s-era GeoCities, a platform that played a significant role in shaping early internet culture by enabling users to create highly personal and individually styled web pages.

Neocities is particularly relevant to this project because it reflects an earlier era of the web where personal expression, experimentation, and individuality were more visible and less constrained by standardised platform design. Websites hosted on Neocities often feel highly personal, with designs that reflect the identity, interests, and creativity of their creators rather than that of a centralised or commercially driven platform. In this sense, it represents a more decentralised and user-driven approach to web publishing.

However, Neocities is also limited by its reliance on manual web development. Users are required to have at least a basic understanding of HTML and CSS, or a willingness to build and maintain their site through code. While this provides a high degree of freedom, it also creates a barrier to entry for users who want expressive control without technical

complexity. In addition, the platform is restricted to static websites, meaning it does not support server-side functionality such as databases or backend systems.

Overall, Neocities highlights the tension between creative freedom and accessibility. It demonstrates the value of open web spaces while also exposing the limitations of code dependent creation.

This influenced the direction of this project by reinforcing the need for a system that supports expressive freedom and individuality, but in a way that remains accessible without requiring technical expertise.

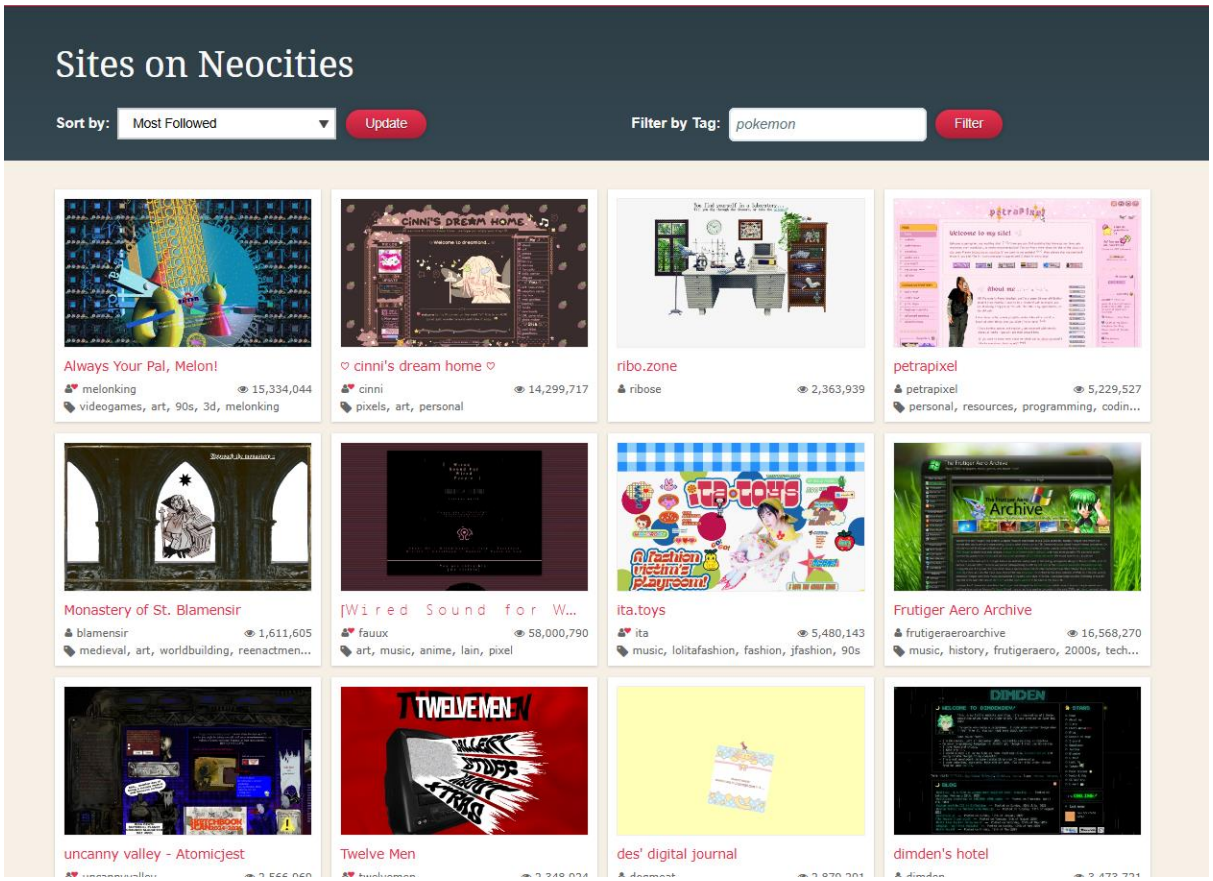


Figure 4 Neocities Featured Sites

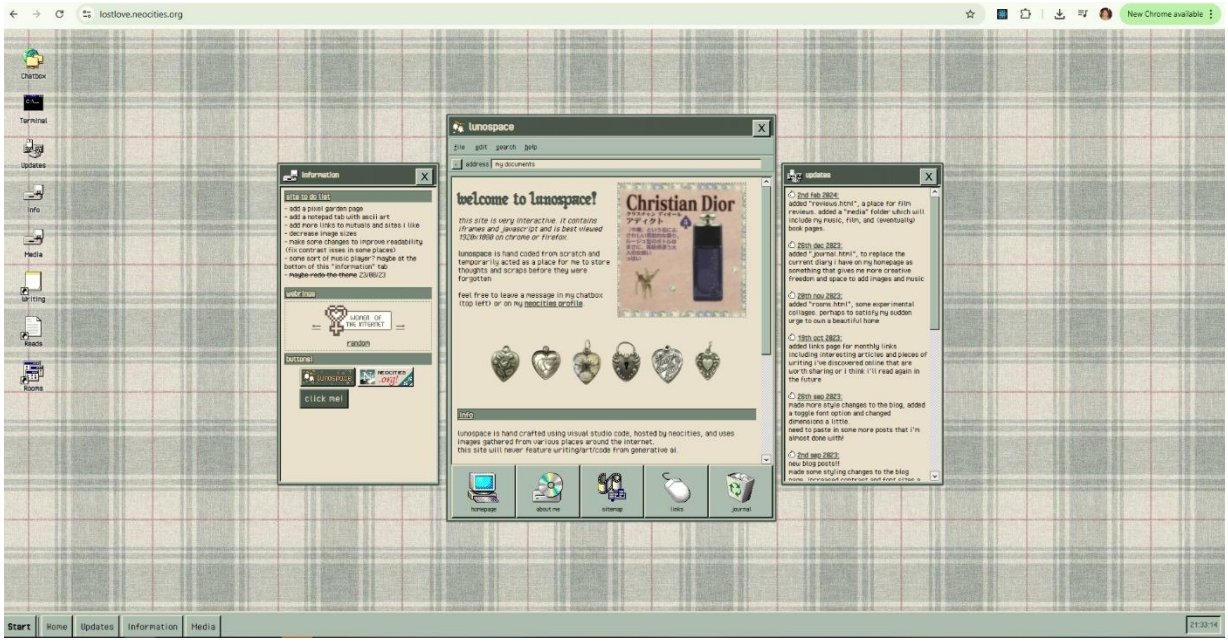


Figure 5 Neocities page (lostlove.neocities.org)

Figure 6 Neocities page (lydels.neocities.org)

Additional Neocities examples were collected to identify common patterns in personal web expression and used as visual inspiration. A wider set of examples is included in Appendix A.

Comparative Summary

The comparative analysis of existing platforms highlights a clear gap between creative freedom, usability, and accessibility in digital systems.

Kinopio provides a flexible, spatial interface that encourages creativity, but is not designed for identity focused or public-facing expression.

Instagram prioritises ease of use and consistency but constrains creativity through standardised layouts.

Squarespace offers structured design tools that enable polished outputs, but remains template-based, which can restrict users from further experimentation.

Neocities supports strong individuality and digital independence, but requires technical knowledge, which limits accessibility for users with no coding experience.

Overall, these platforms demonstrate a trade-off between control, creativity, and accessibility, with no single system achieving all three effectively.

In response, this project aims to combine the flexible interaction model of Kinopio with the independence associated with Neocities, while avoiding the template constraints of Squarespace and the controlled nature of Instagram. The resulting system focuses on a canvas that aims to prioritise user autonomy, creative freedom, and personal expression.

2.2 Technical Research

2.2.1 Frameworks, Libraries and Programming Languages

To identify an appropriate technical stack for the project, research was conducted into front-end frameworks, interaction libraries, backend services, and supporting development tools. The final stack was selected based on the project's specific requirements: node-based interaction, persistent user generated content, maintainability, and flexibility in visual design.

Front-end frameworks, Interaction and UI Libraries

An early technical decision involved choosing between React and Svelte for the front end. Svelte was considered because of how lightweight it was and through research it was found that it was good for building small interactive interfaces. However, React was ultimately selected because it offered reusable components, worked well with typescript, would scale well as the project developed, had broader community support, and a greater range of third-party libraries relevant to the project.

Initial experiments were carried out in both React and Svelte, with these comparative explorations documented in the project's design development materials. This prototyping process helped confirm that React was the more practical choice for the final design of the project.

Because drag-and-drop interaction is central to the project, research was also conducted into libraries that could support movement and interactive placement. The two main candidates were dnd-kit for React and SvelteDnD for Svelte. Since React had already been chosen as the preferred front-end framework, dnd-kit became the more appropriate option.

Development Environment

Vite was selected as the main development environment for the application. This decision was based on its fast development server and lightweight setup compared with other build tools. For a project involving frequent interface experimentation and iterative development, this improved the speed of testing changes and reduced friction during implementation.

Vite also integrates cleanly with React and TypeScript, making it well suited to a modern front-end workflow. Its simplicity supported the practical needs of the project.

Back-end and Database

Similarly to trying to decide between React and Svelte, Firebase was initially considered because of its popularity, lightweight setup, and strong support for rapid prototyping. However, Supabase was ultimately chosen because the project required a relational data model with stronger control over ownership and access.

Supabase is an open-source "Backend-as-a-Service" (BaaS) platform built on top of PostgreSQL. The application will store entities such as users, canvas nodes, and guestbook entries. These relationships are more naturally represented in a relational

SQL database than in a document-based NoSQL structure. Supabase, being built on PostgreSQL, provided a clearer way to model these relationships and query related data efficiently. It also offered Row Level Security, which was especially important because the application needed to ensure that users could edit their own content while restricting access to other users' data.

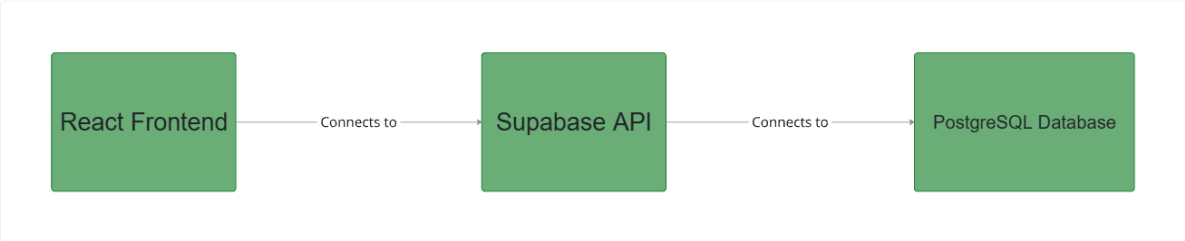


Figure 7 Diagram showing the simple system architecture.

Supabase was therefore a better match for both the technical structure and security needs of the project. Its open-source nature also made it appealing as a transparent alternative to Firebase, while still providing many of the convenience associated with Backend-as-a-Service platforms, including authentication, database hosting, APIs, and storage.

Programming Languages and Styling

The main programming languages used in the project were TypeScript, HTML, and CSS. TypeScript was selected as the primary development language because the application contains multiple interacting parts with different properties and behaviours. In this context, TypeScript improved maintainability by allowing these structures to be explicitly defined and checked during development. This reduced the likelihood of errors as the project grew.

TypeScript was also well suited to React, particularly when splitting the application into reusable components. As the project became larger, type definitions helped make component props, state structures, and shared utility functions easier to understand and manage.

For styling, Tailwind CSS was initially considered and briefly tested. However, it was later decided against because the project's interface required a more custom visual identity than utility first styling supported. Since the application aims to encourages creative visual customisation, it was important to have more control over the appearance and theming.

Therefore, CSS was selected over Tailwind because the project required a more flexible and structured theming approach. Inspiration was also taken from Neocities for this use of CSS.

Additional Libraries

Several supporting libraries were also selected to address specific needs within the project.

html2canvas was used to generate a visual snapshot of the user's canvas for marketplace sharing. This approach was chosen after experimentation showed that reconstructing a canvas entirely from database data was slower and did not always preserve embedded media reliably, particularly content such as YouTube or Spotify embeds. By capturing the rendered canvas as an image, the system could preserve the appearance of the page better for previews.

Motion was used to add lightweight animations and transitions throughout the interface. These animations were not included only for decoration, they also supported usability by making interactions feel more responsive and by providing visual feedback during state changes. Given the playful and expressive aims of the application, subtle animations when changing pages helped reinforce the intended tone of the user experience.

Lucide React was selected as the icon library because it provides lightweight, scalable SVG-based icons as React components. This suited the project's component architecture and allowed interface controls to remain visually consistent without adding anything excessive.

2.2.2 Key Challenges and Considerations

Several technical challenges emerged from the chosen direction of the project. One of the main challenges was supporting a highly interactive canvas interface while keeping the code and UI manageable and consistent. As the application allows users to create and edit different node types, each with their own properties and behaviours, the system needed a flexible structure that could support variation without becoming overly complex.

The project also requires a balance between expressive interface design and technical maintainability. Since the application is intended to support individuality and visual customisation, the technical stack needed to allow flexible styling, theming and component variation without compromising the structure, reusability or scalability.

2.3 Critical Discussion: Homogenisation of Web Design

Homogenisation is the process by which different cultures, systems or forms become increasingly similar. Within web design, it refers to the growing trend of online spaces, websites and applications looking, sounding and feeling the same.

2.3.1 Causes of Homogenisation

Gorre et Al (2021) identified homogenisation as both a technical and cultural outcome of modern web development practices. Their study uses a mixed-method approach to identify large scale patterns of homogenisation. It demonstrates that websites increasingly rely on similar layouts, typography and navigation due to using the same frameworks and templates.

Using computer vison to analyse over 10,000 web page images from 2003 to 2019, it found that page layouts had grown over 30% more similar since 2003, with a sharp rise past 2007 and continuing until 2016.This sharp rise appears to have begun following the release of the iPhone, which necessitated a move away from web page focused design and led to designers adopting responsive design. Designers began using increasingly similar front end libraries to efficiently support mobile platforms. Goree et Al (2021) found that the correlation between mobile support and visual similarity is 0.84 and search engine optimization strategies such as those enforced by Google (e.g., the 2015 "Mobilegeddon" update) pressured the industry to adopt mobile friendly responsive designs.

2.3.2 The Role of AI in Accelerating UX/UI Trends

In addition to this the integration of artificial intelligence into web design and development has significantly accelerated the homogenisation of web design. These

systems rely on large datasets that use existing design patterns and tend to reuse dominant UI and UX practices.

Muthazhagu (2024) describe how automated code generation eases the process of turning ideas and images into web code. One of the major challenges identified is the requirements for high-quality training models. AI models are primarily trained on a limited set of successful and common design practices which in turn leads to the reinforcement of homogenised user interfaces. These underlying algorithms and training data restrict the diversity and unique outputs leading to everything feeling the same. Standardisation is a potential, albeit unstated consequence of this level of AI automation.

This is further backed up by research by Deriba et al (2025) which shows that students believed AI tools can support accessibility by identifying usability barriers and suggesting improvements during the design process, but while this can improve inclusion, accessibility is often treated as a compliance checklist rather than a flexible design approach. When applied rigidly, accessibility standards unintentionally contribute to these homogenised designs, which reinforces the same layouts and interaction patterns across the web

While the integration of AI tools has positively brought advancements within accessibility. There will always still be a place in the industry for creative minds and even though AI algorithms excel in automating tasks and finding patterns, they lack imagination and human judgement. They cannot understand human emotion and can struggle with understanding abstract situations that require a deeper understanding of human behaviour. (Baxter, n.d.).

2.3.3 Centralisation, Accessibility and Algorithmic Influence

Accessibility is a key requirement and is widely studied area in modern web design. It is important that web content is usable by everyone and adheres to the Web Content Accessibility Guidelines (WCAG) to create inclusive online spaces. These guidelines have helped establish a baseline standard for inclusive design and it involves a wide range of disabilities, including visual, auditory, physical, speech, cognitive, language, learning, and neurological disabilities. Nielsens usability heuristics have also endured for similar reasons, with users expecting websites to behave in familiar and predictable ways. These standards while certainly necessary, their widespread adoption has encouraged designers to stop thinking outside the box. Gibbons and Jen (2025) highlight an overreliance on frameworks as undermining developers critical thinking and that

they are falling into what is called the ‘template trap’. This is due to not only a need for efficiency but also unfortunately laziness with the belief that downloading the latest framework or UI components will immediately solve any design issues.

This effect is further strengthened by the increasing centralisation of the internet. Centralisation is the idea that something is controlled by one entity and with the internet currently, everything is decided for you. Platforms such as Google decides what you see, Meta decides what you feel and Amazon decides what you buy. What began as useful services have evolved into monopiles largely because centralisation is efficient. It is easier to control, moderate and monetize a unified system than a decentralised one (Davis, 2025). These large tech companies see our time and attention as their final frontier. Everything we watch is decided by an algorithm.

An algorithm has no soul, no duty no consciousness. It cannot be moved by art, it cannot act, and it cannot love. Chayka (2024) discusses this within his book which is a critique and investigation of this and asks what happens when our cultural and artistic lives are driven by an algorithm, and your success is curated by how much engagement you receive.

Kahle (2015) creator of the internet archive discusses the idea of building a "Distributed Web" that would be more reliable, private, and fun than the current World Wide Web. The key points being that the current web is fragile, with web pages only lasting about 100 days on average before changing or disappearing. The author believes we now have the technological capabilities to build a more robust, distributed web and this would leverage technologies like public key encryption, blockchain, peer-to-peer networking, and JavaScript running in browsers to create websites that are decentralized and resistant to censorship or surveillance

2.3.4 Digital Reclamation

In response to the homogenisation of web design and increasing centralisation of digital platforms, movements have emerged that put an emphasis on personal expression, decentralisation and user autonomy and most importantly freedom. This research will be focusing on the IndieWeb movement

The origins of the IndieWeb can be seen as early as 1999. With some of the earliest evidence being a boycott that was caused by Yahoo! in 1999. Users felt that Yahoo changing Terms of Service granted them corporate access to the content hosted on the GeoCities servers and in turn caused a boycott due to users feeling that this threatened their creative control. (Reynold and Hallinan, 2020)

However, the main IndieWeb movement emerged in the 2010s as a result to a growing dominance of platforms such as Facebook and Instagram which centralised control over content, data and identity. IndieWeb co-founder Çelik (2014) aimed the movement as a direct response to these centralised platforms. He mentions that Twitter and Facebook were the beginning of the downfall of personal autonomy online and the IndieWeb was a direct response to try bringing the web back to how it was in 2003. Concerned about the erosion of personal agency, a community of designers and developers aimed to promote personal websites with domain ownership, and independent publishing as alternatives (Gillmor, 2014). Central to the movement is the principle that users should "own their data," enabling them to control how content is created, shared, and archived. This ethos reflects a broader desire to preserve the decentralised and fun culture of the early web, where individuals could communicate and innovate without seeking permission from centralised authorities (Lindahl, 2025).

The IndieWeb is tied closely to another moment called the weird web or yester web, which focuses heavily on the early webs aesthetics and embrace experimentation while rejecting modern interfaces. The designs challenges usability norms by today's standards but have a uniquely human quality to them due to their imperfect nature. They include animated gifs, non-linear navigation and extremely unconventional layouts. Baker (2022) reflects on the internet from its early personal and community driven days and how we have lost the ability to close the door on the internet and how a lot of these movements are a wish to return to a simpler time before the internet was the force it is today. Many people within the IndieWeb community wish to recapture this feeling of the early internet.

The Indie web is guided by a core ten principles

1. Own your data.

2. Use & publish visible data for humans first, machines second.

3. Make what you need.

4. Use what you make.

5. Document your stuff.

6. Open source your stuff.

7. UX and design is more important than protocols, formats, data models, schema etc.
8. Modularity.

9. Longevity.

10. Plurality.

It is a series of interlinked personal sites with little commercial value. It's made up of a community of people who have created a decentralised social network built on older web ideologies. It is a hidden subculture that prides itself on being human and personal (Lindahl, 2025)

This movement does face notable limitations such as user's needing coding knowledge or hosting infrastructure which restricts widespread adoption. Reach can also be a challenge as IndieWeb content generally exists outside of general social media ecosystems. The majority of the IndieWeb community are participating as a hobby. And it has no formal organised structure.

2.3.5 Resistance and Agency in Web Design

In recent years, online hubs such as Neocities and Yesterweb have played a central role in reviving the ethos of the early web. Following the IndieWeb principles, it emphasises small, personal websites where users can exercise and maintain ownership of their own content. There has also been a growing interest in the IndieWeb across platforms such as YouTube and Tumblr as members of these communities share resources, tips and code and help newcomers explore this alternative web. The collaboration fosters a sense of belonging within the community and they reflect deliberate response to homogenisation and the issues with centralisation.

Homogenisation in web design is not merely an aesthetic issue. As discussed previously it emerges from broader structural, economic and technological factors. Centralised platforms widely adopted design frameworks and AI encourage predictable patterns that prioritise efficiency and consistency. These benefits are undeniable, such as faster development cycles and improved usability but have contributed to our current digital environment that feels overwhelmingly corporate and defined by a lack of unique ideas and passion. Designers are encouraged to replicate familiar patterns rather than experiment and economic pressures to reach audiences as quickly as possible and optimise engagement often reinforces these established conventions.

3. Requirements Analysis:

3.1 Requirements Gathering

3.1.1 Insights from Existing Platforms

As identified in Section 2.1.1, existing platforms such as Neocities, Kinopio, and Squarespace highlight a consistent gap between creative freedom and accessibility in digital systems. This analysis, combined with research into the homogenisation of web design and the project's focus on user autonomy directly informed the system requirements for the proposed application.

3.1.2 User Research (Survey)

Expanding on the research done previously, a survey was conducted to gather perspectives on self-expression, customisation, and the current sentiments online amongst peers and potential users of the application. The purpose of the survey was to better understand whether potential users felt constrained by existing online spaces and whether there was interest in a website with a more personalised and editable environment. 22 responses were gathered.

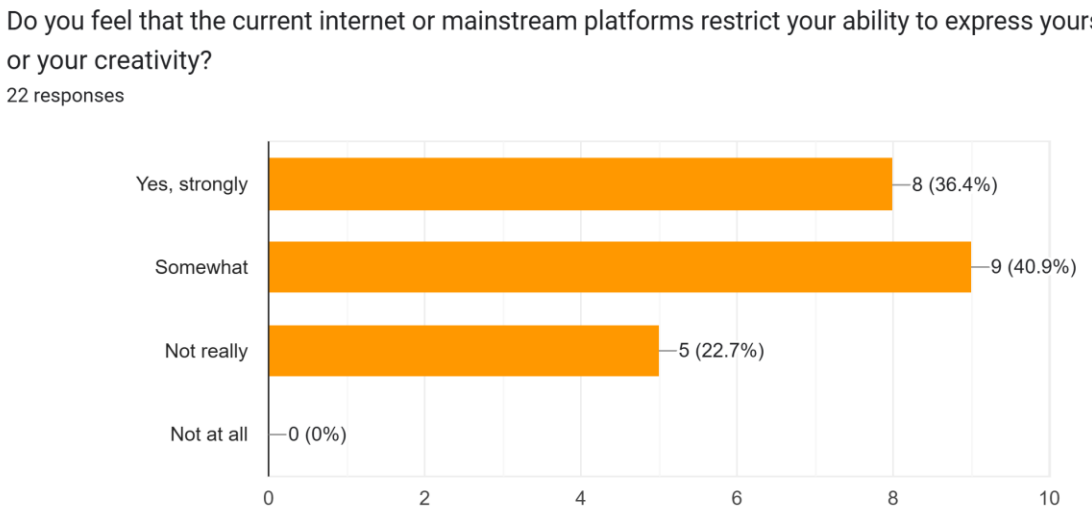


Figure 8 Survey Response demonstrating feelings of platform restriction

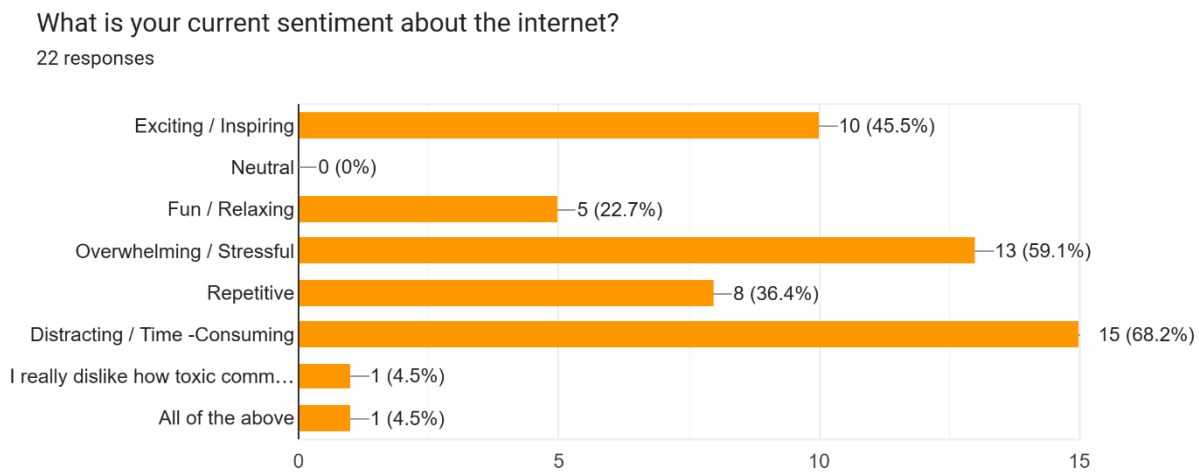


Figure 9 Users current feelings surrounding the internet

These findings supported the inclusion of features such as theme customisation, flexible layout editing, and draggable components. It was also found that many users felt that the internet currently felt frustrating and overwhelming.

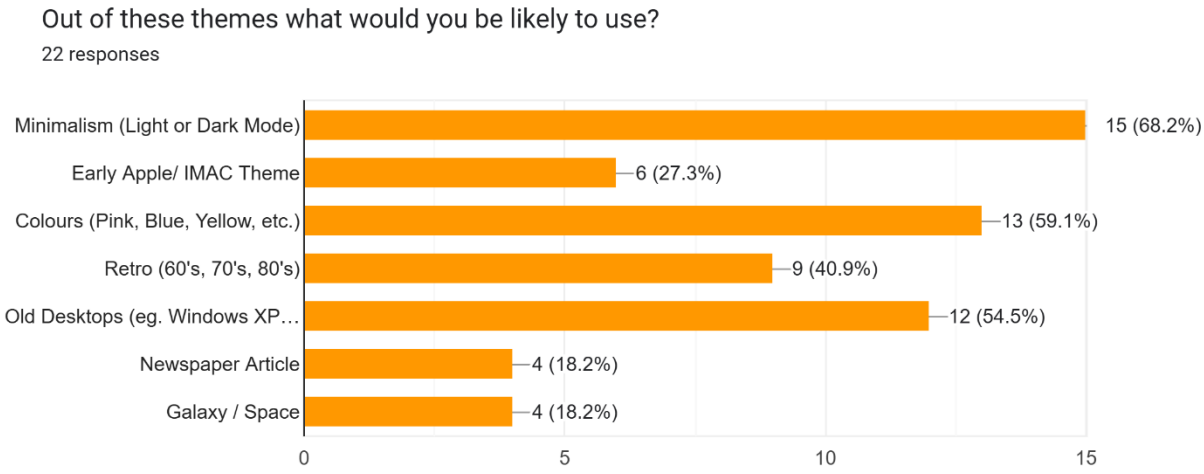


Figure 10 Theme choices

"Most people using social media as their portfolio have to adhere to trends and audios that the algorithm will push rather than create something unique from scratch. Often I think the truly most creative people get lost in the algorithm"

Anonymous Survey Response

Additional screenshots documenting the survey can be found in Appendix B.

3.2 Requirements Modelling

3.2.1 Personas

A persona is a fictional representation of a typical user of a system. Personas support the design process by helping developers and designers consider real user needs, goals, behaviours and frustrations rather than designing only from their own assumptions. As Dykes (2025) states, personas help product teams empathise with users.

For this project, three personas were created to represent different potential users of the application. Each persona focused on a different use case, including creative self-expression, casual profile customisation and sharing content with others.

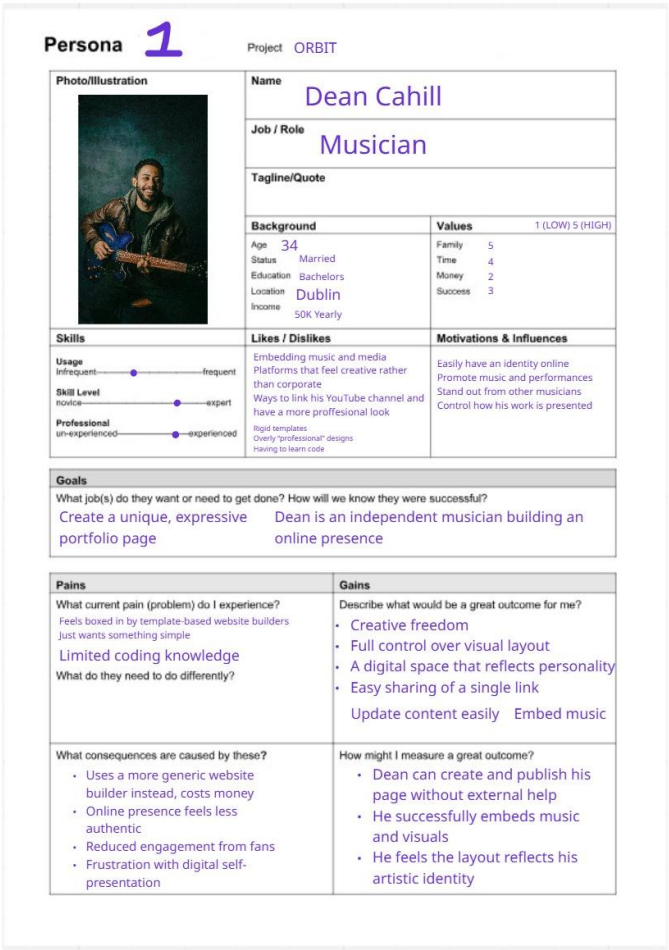


Figure 11 Persona #1

Additional Personas can be found in Appendix C.

3.2.2 Use case diagrams

A use case diagram is a visual modelling tool used to show how users, or actors, interact with a system and the main functions the system provides. It helps describe system behaviour from the user's perspective by showing the actions users should be able to complete within the application (Atlassian, 2025).

For this project, the use case diagram was used to identify the main interactions between users and Orbit, including registration, login, canvas editing, publishing, marketplace browsing and how users would interact with the different components.

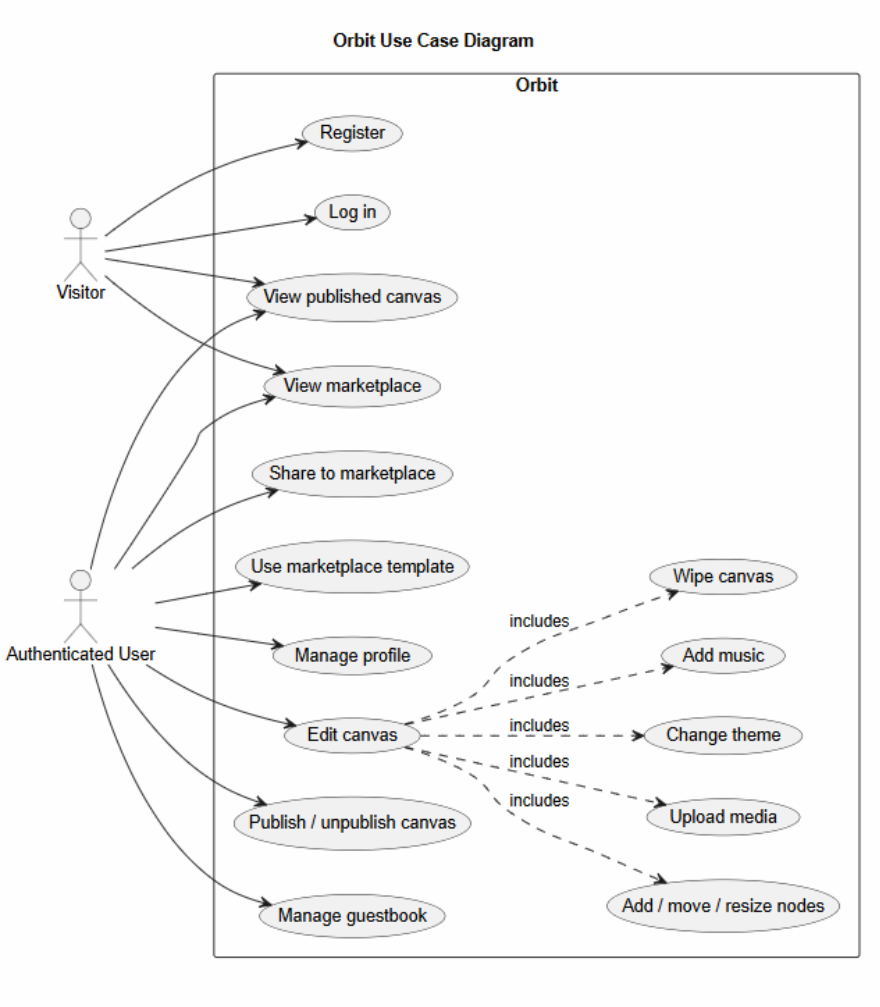


Figure 12 Use Case Diagram

3.2.3 Functional requirements

Functional requirements were needed to define what Orbit should do, while non-functional requirements were needed to define how well it should work. This helped turn the project idea into a clear set of features that could be designed, implemented, and tested.

This ensured that Orbit was designed as a real software system with different features, not just a visual design.

Function	Why?	Priority
The system shall allow users to create an account, log in, and securely authenticate	To ensure each user has a private account and their canvas data is linked to their identity	High
The system shall provide each user with a personal canvas upon login.	To give every user their own editable canvas	High
The system shall allow users to create and use different components	To support creative expression through different content types such as cards, images and different embeds	High
The system shall allow users to drag and reposition components freely on the canvas	To give users control and allow them to design the canvas freely	High
The system shall allow users to resize cards	To let users adapt components to fit different amounts of content and layout styles	High
The system shall allow users to edit the existing components at any time after creation	Support customisation and usability.	High
The system shall automatically save the position, size and content of all components so that the canvas is restored when the user returns	To ensure user work is persistent and not lost during a refresh or logout	High
User Security (Passwords Hashed in the database)	To protect user accounts	High
The system will allow users to delete components from the canvas	Allow users to remove unwanted content	High

Users will be able to swap between multiple pre-designed themes at anytime	Support personalisation without requiring users to design themes manually	Med
The system shall support both desktop interactions (mouse) and mobile interactions (touch)	To make the application usable across different devices and screen sizes	Med
The system will allow users to upload their canvas to a shared market	To encourage community-based creativity	Med
The system will allow users to upload their card components to a shared market	To allow smaller reusable designs to be shared without requiring users to share and entire canvas	Med
Users will be able to remove their canvas's visibility at anytime	To give users control over privacy and whether their canvas is publicly accessible	Med
The system will allow users to full screen their canvas	To provide a better view when working with multiple components	Med
Components z-index can be changed whenever.	To allow users to control layering when components overlap	Low
Users will be able to change canvas colour and background	Increase personalisation and make canvases feel visually distinct.	Low
Users will be able to wipe their canvas and restart.	To allow users to quickly reset the canvas without deleting the whole account.	Low
Export functionality for card components (JSON)	Allows cards to be reused, backed up and shared.	Low
Users can preview other users uploaded canvas/cards on the marketplace	Visual to help users decide before downloading.	Low
Users can download and use other users uploaded canvas/cards on the marketplace	Community creativity.	Low
Users can update personal information (Pronouns, Avatar and Email, Password and Username)	To allow users to keep their profiles accurate, personalised and secure.	Low
Users can upload MP3 files	Further personalisation.	Low

3.2.4 Non-functional requirements

Function	Why?	Priority
The system shall be designed to be usable by individuals without prior coding knowledge, and that all core interactions can be performed intuitively	The application is aimed at creative self-expression, so users should be able to build a canvas without needing technical skills.	High
The system shall provide a baseline level of accessibility, including readable typography, colour contrast and keyboard actions	Ensures the interface is easy to read, navigate and interact with for a wide range of users.	High
The system shall not constrain users to fixed templates or predefined layouts, allowing unrestricted arrangement of components within the canvas	Central to the projects aim and justification.	High
The system shall maintain smooth performance during typical usage, including the creation, manipulation, and rendering of multiple interactive components	Smooth performance is needed so dragging and resizing is not frustrating.	High
The system shall provide a consistent interaction model, ensuring that core behaviours remain consistent	Consistency helps users learn the interface more quickly, similar actions should perform the same across all nodes.	Med
The interface shall be full responsive and adapt across both desktop and mobile devices	Allow user to access and view application across different devices.	Med
Smooth animations and transitions while loading pages	Adds polish, but not essential to core-functionality.	Low
The system shall function consistently across modern browsers such as Chrome, Firefox and Edge	Browser compatibility ensures users are not restricted	Low

4. Design:

4.1 System Architecture

4.1.1 UML Diagrams: Structure and Behaviour

According to Atlassian (2025), a UML (Unified Modeling Language) diagram is a visual representation of a system that shows how different software or business components interact. UML diagrams help organize complex ideas into clear, structured visuals.

The structural diagrams were useful as they showed how the major parts of the application were organised, while behavioural diagrams were used to show how users interacted with the platform and how their actions would cause the system to respond.

Components Diagram

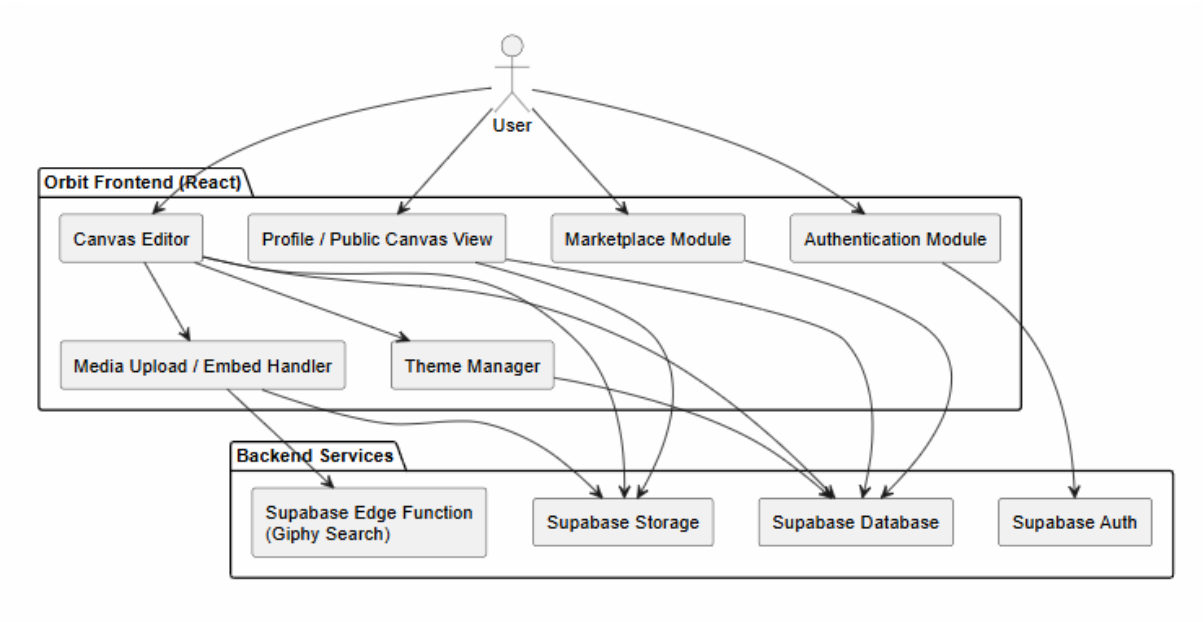


Figure 13 Components Diagram

Activity Diagram

Two activity diagrams were created to represent the behaviour of the system's main user flows. The first models the canvas interaction flow, showing how an authenticated user logs in, edits their canvas, saves changes, and triggers updates to the backend database.

The second models the marketplace flow, showing how a user interacts with marketplace content, such as browsing items, selecting a card/canvas, and completing a download or upload action. These diagrams were useful in illustrating how user actions move through the system and how the platform would respond at each stage.

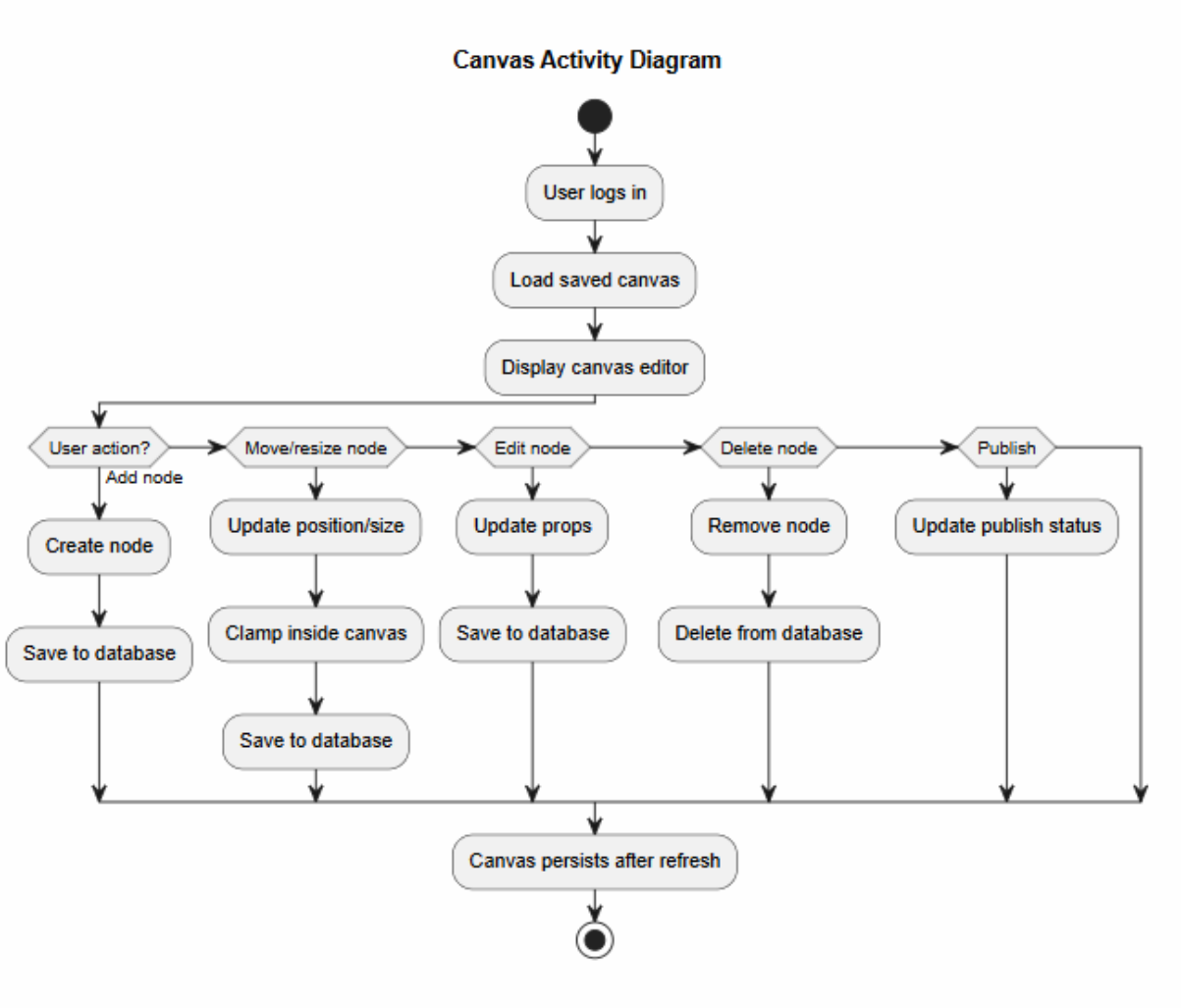


Figure 14 Canvas Activity Diagram

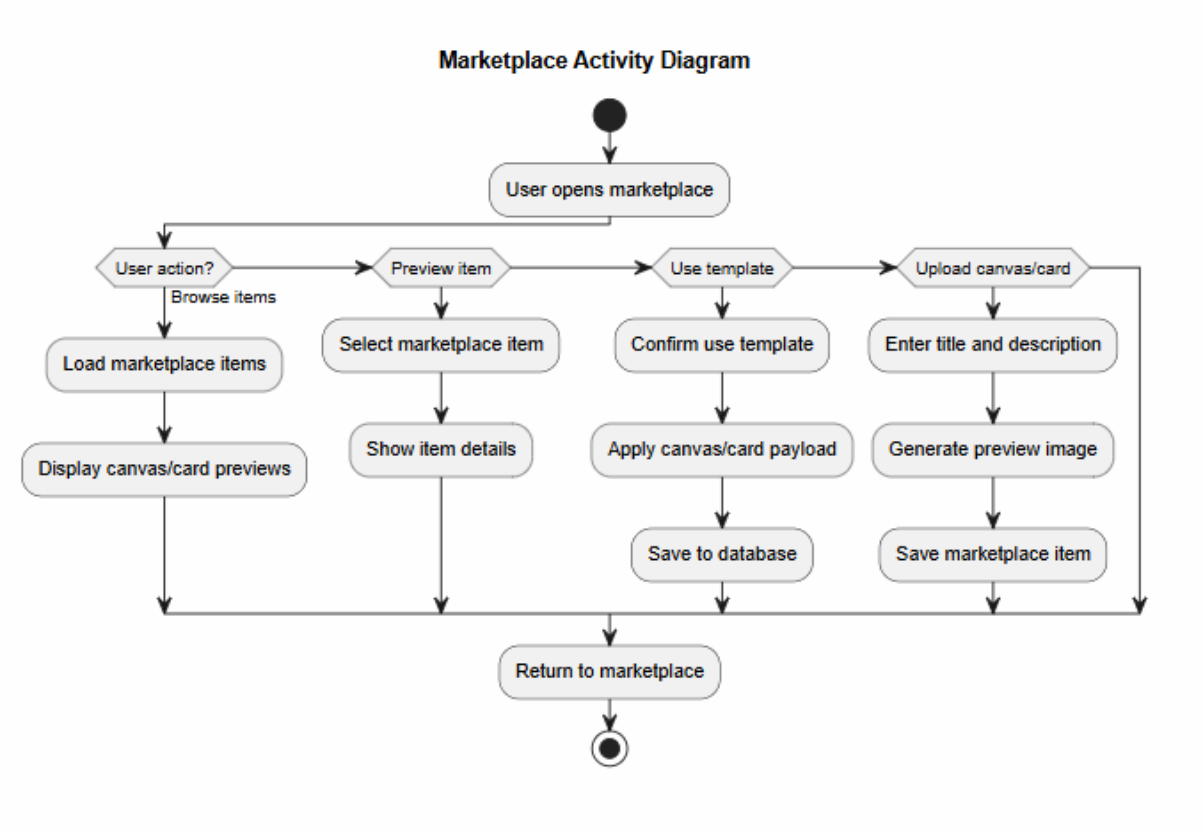


Figure 15 Marketplace Activity Diagram

4.1.2 Architectural and Design Patterns

Orbit was designed using a component-based architecture, which is well suited for React development. This approach allowed the interface to be turned into smaller reusable elements such as cards, modals, MP3 and a theme selector. This component-based structure supported iterative development as it allowed for adding individual features which could be changed without requiring major changes to the application. Reusability was a key design principle as inspired earlier by Kinopio.

The system was also structured around separation of concerns between the frontend, backend, and persistence layer. The React frontend was responsible for rendering the user interface and handling the interaction logic. Supabase then provided the application with authentication, database storage and file handling.

The canvas editor section followed an event driven interaction model. User actions such as dragging, resizing and changing themes would all trigger state updates and UI feedback. This was essential as it supported an immediate and satisfying visual response

4.2 Interface Design:

4.2.1 Wireframes

A wireframe is a basic blueprint or skeletal outline of a website, app or digital product. It is usually created during the early stages of the design process and focuses on layout, navigation and essential interface elements rather than final colours, branding or detailed graphics (Kushwah, 2025).

Low-fidelity wireframes were initially sketched on paper to explore the basic layout, structure and user flow of Orbit without focusing on detailed visual design. These early wireframes helped define the placement of key areas such as the canvas, toolbox, profile view and marketplace.

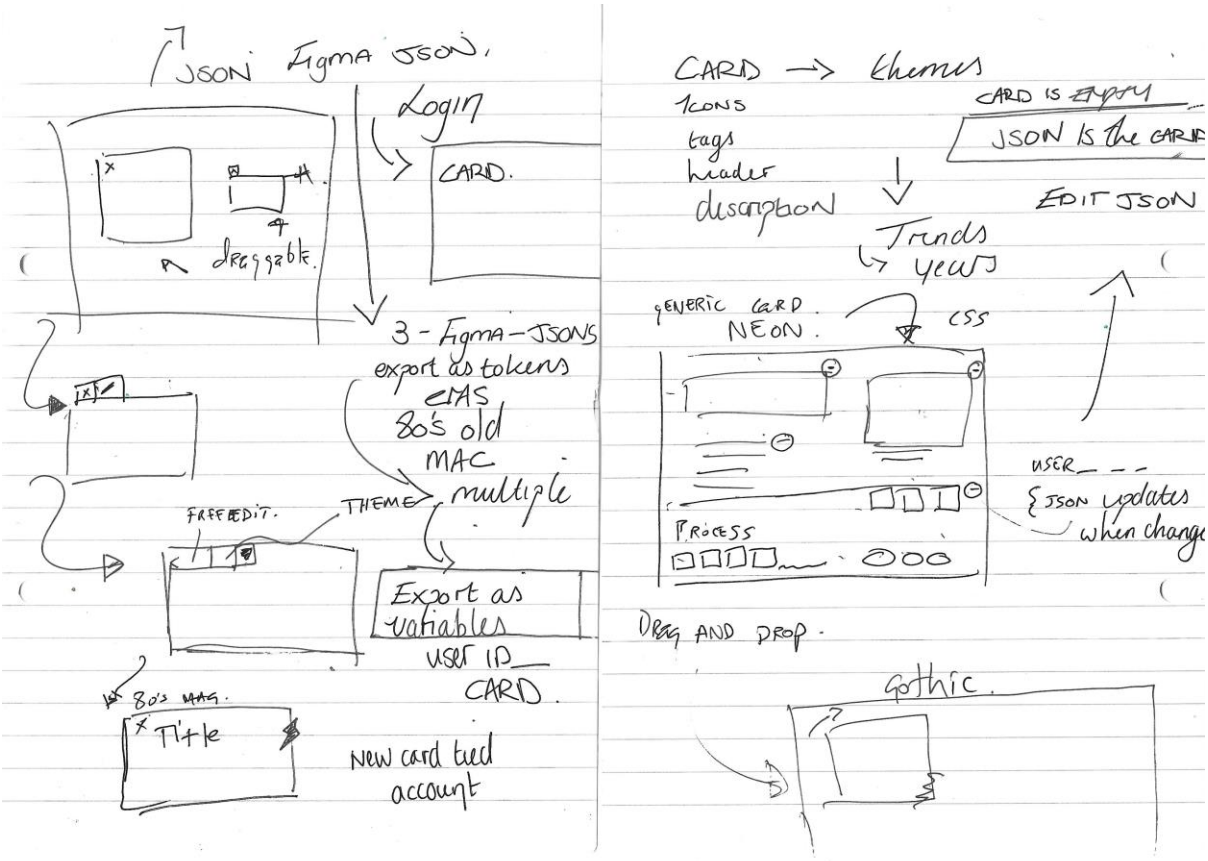


Figure 16 Initial Concept

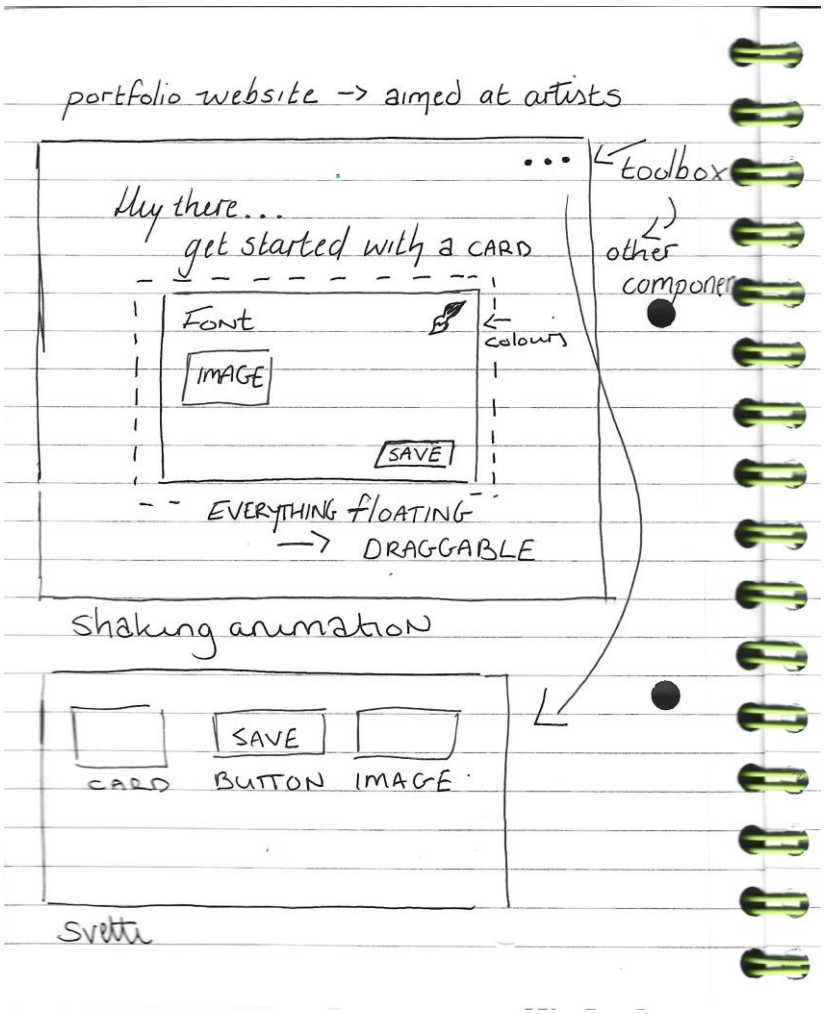


Figure 17 Initial Concept #2

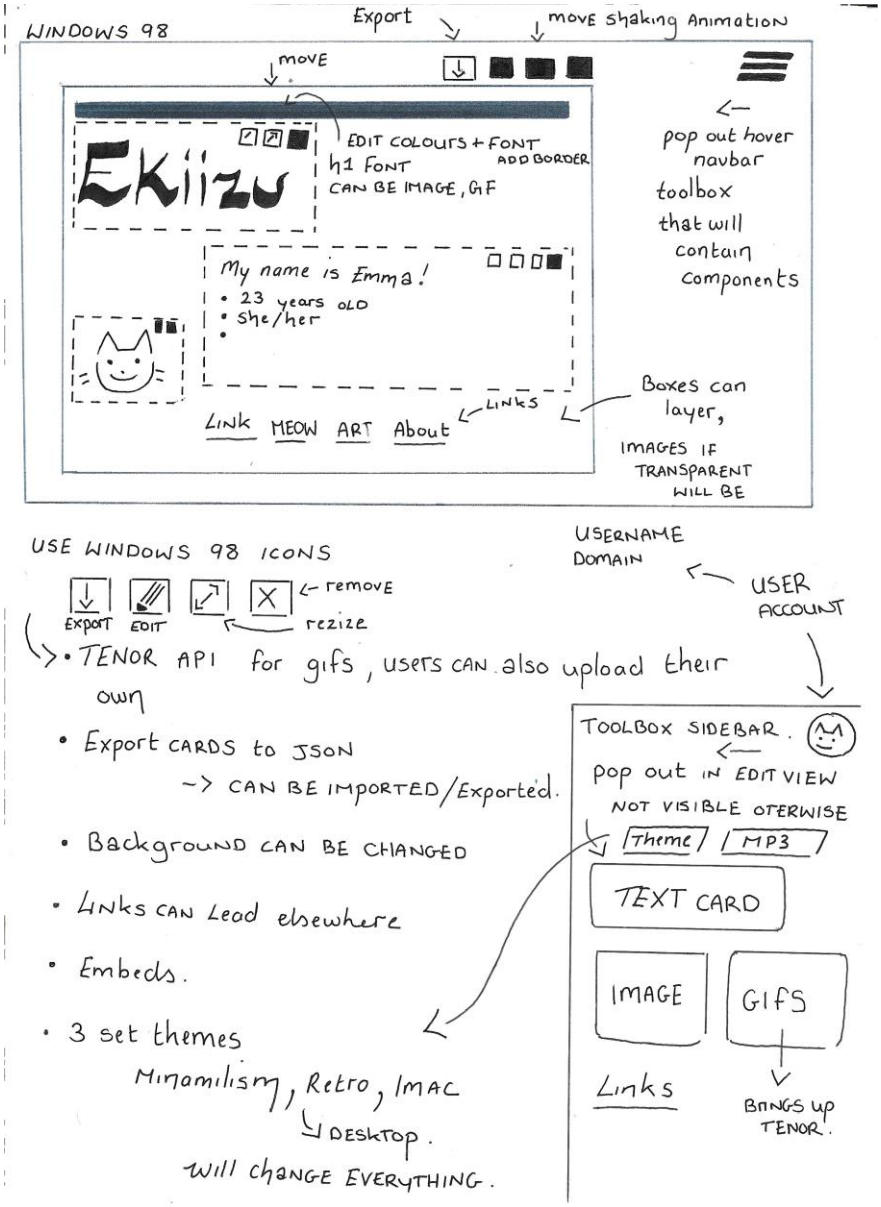


Figure 18 Wireframe

Additional images documenting wireframes can be found in Appendix C and on Miro.

4.2.2 User Flow

A user flow is a visual representation of the path a user takes through a website or application to complete a task. It shows the sequence of screens, actions and decision points involved in the user journey. User flows are useful in UX design because they help designers to see the process from a user's perspective. (Visily 2025)

For this project, user flows were created to plan how users would move through Orbit's main features, including registration, login and using the canvas.

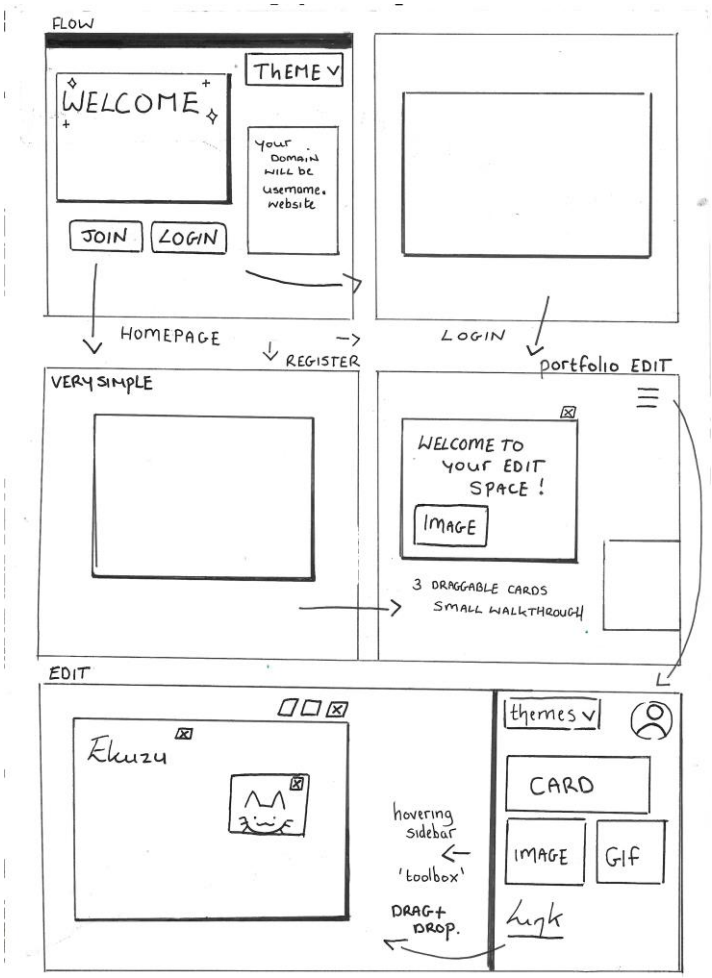


Figure 19 User Flow #1

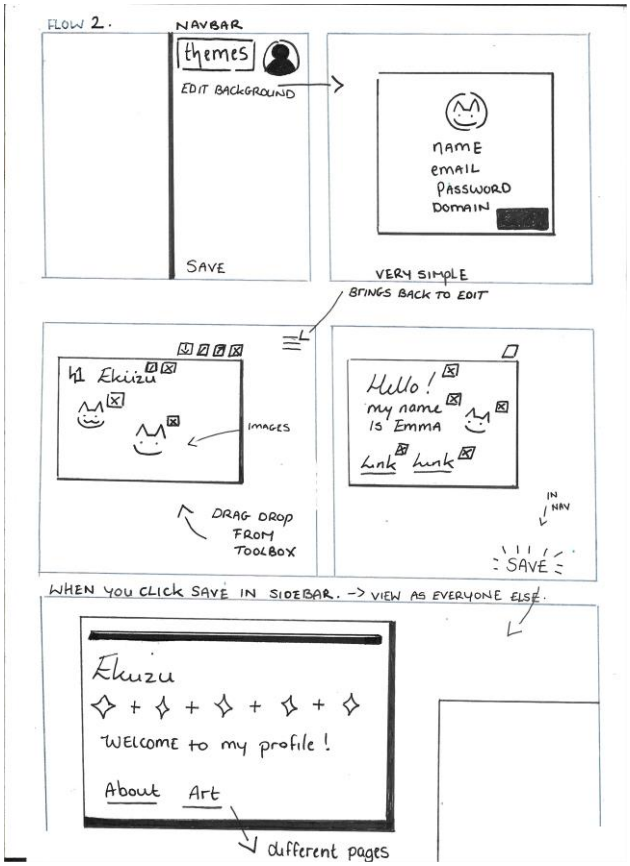


Figure 20 User Flow #2

4.2.3 Rationale for UX/UI

The user interface and experience were designed using established usability principles to ensure that Orbit was intuitive and accessible. A key influence was Nielsen Norman Group's usability heuristics, which were used to guide decisions around clarity, feedback, consistency and user control. These principles were particularly relevant because the application relies on interactive editing, drag-and-drop movement and user-generated layouts. Therefore, users needed clear feedback when adding, moving, editing or deleting components.

Theme-aware design was introduced to support accessibility, comfort and personal expression across different viewing conditions. Since Orbit planned to have several visually distinct themes, the interface needed to allow users to switch styles without disrupting the layout or usability of the application. This was achieved through a CSS variable based theme system, where colours and interface tokens could change while the underlying structure remained consistent.

Visual hierarchy was also applied to make key actions and controls easy to identify. Buttons such as save, publish, marketplace, delete and edit were positioned consistently and given clear visual emphasis. This helped guide users through the editing process and reduced confusion, particularly when using the toolbox or contextual menu controls.

Overall, the UI/UX design aimed to balance creative freedom with usability. While the project intentionally challenges rigid template-based design, it still uses clear well documented interaction patterns and consistent controls.

4.2.4 Design Decisions and Style Guide

The visual design of the system draws heavily from early graphical user interfaces and interaction design history. A major influence was the work of Susan Kare, whose iconography for early Apple systems helped define how digital interfaces communicate meaning through simple, symbolic visuals. Kohlstedt (2018) highlights how Kare's designs translated complex digital functions into approachable visual symbols, while Cesarato (2023) discusses how her work helped make early computer interfaces feel more human and accessible. This influenced Orbit's use of icons, nostalgic interface references and visually expressive themes.

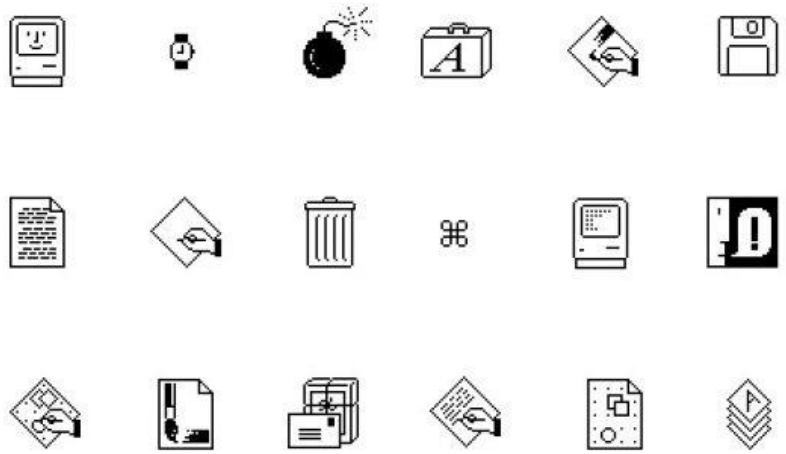


Figure 21 Early Mac Icons. Reprinted from Kohlstedt (2018).

Building on the digital reclamation and IndieWeb concepts discussed in Chapter 2, these principles informed the design direction of the system, particularly in relation to user autonomy, customisation and interface flexibility. Rather than forcing users into a fixed template, the interface was designed around a freeform canvas where users can arrange, layer and personalise content.

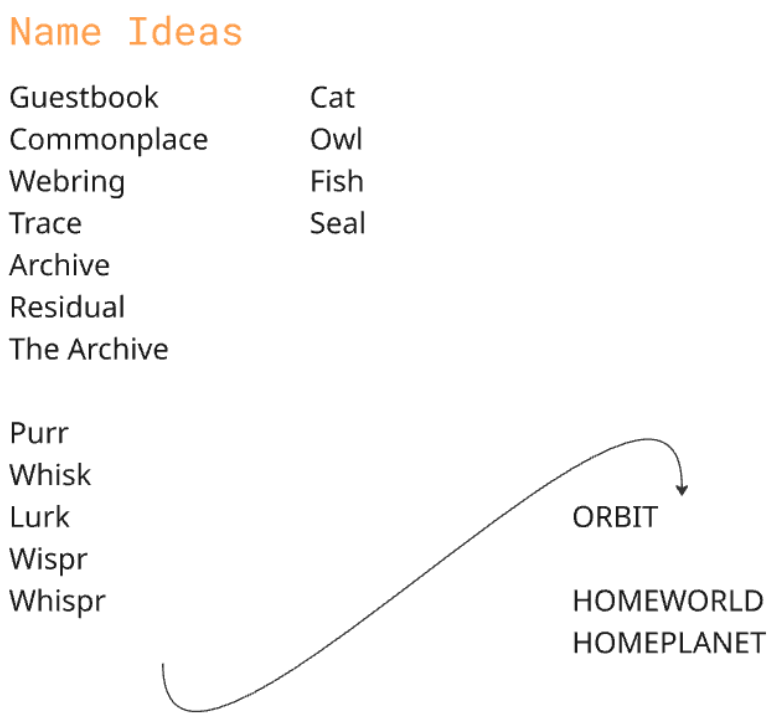


Figure 22 Initial name ideas.

The name Orbit was chosen to represent the idea of a personal digital space. Each user's canvas acts like their own small universe. The application went through many different names before landing on the name Orbit

A consistent colour system was developed using modern theming principles. Bell (2024) discusses how modern CSS theme systems can be built through reusable foundations, allowing visual styles to change while keeping the underlying structure consistent. This informed Orbit's use of CSS variables, where colours, backgrounds and interface tokens can change between themes without breaking layout or usability. This will be further discussed in chapter 5.

Building on from this, the theme system was one of the main design decisions used to support Orbit's concept of expressive digital spaces. Instead of presenting the application through a single interface, multiple themes were created to give users greater control over the look and feel of their workspace. This directly supports the

project's aim of resisting visual homogenisation by allowing the same functional system to take on different visual identities.

These were the themes that were selected for Orbit:

- Default
- Orange
- Lime
- Blue
- Pink
- Macintosh 1984
- Windows 98
- Terminal

The Macintosh 1984, Windows 98, and Terminal themes were influenced by earlier eras of computing and web culture. These themes connect directly to the project's research into digital nostalgia, early interface design, and the more experimental visual identity of the pre-standardised web. They were included to show how historical visual styles can be reinterpreted within a modern application. The correct fonts for each theme were also used. Default themes and colour themes all had a corresponding light and dark mode in line with modern accessibility standards.

Typography was selected to support readability across different devices. Clear, legible typefaces were prioritised so that headings, controls and content could be scanned quickly. The visual style also drew inspiration from music and CD album artwork, particularly Blur's *The Great Escape*.

Button and interactive element design was guided by usability principles, particularly Nielsen Norman Group's usability heuristics. Interactive elements were designed to be consistent, predictable and responsive, with clear hover and active states. In the CSS, shared classes such as *.btn* helped ensure that buttons looked and behaved consistently across the application

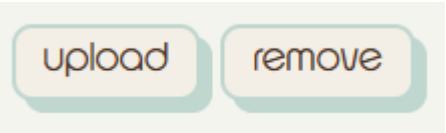


Figure 23 Button Design.

Modal designs were kept minimal and focused so that user attention remained on the task. This supported error prevention and reduced distraction during important actions such as publishing, marketplace submission or wiping the canvas.

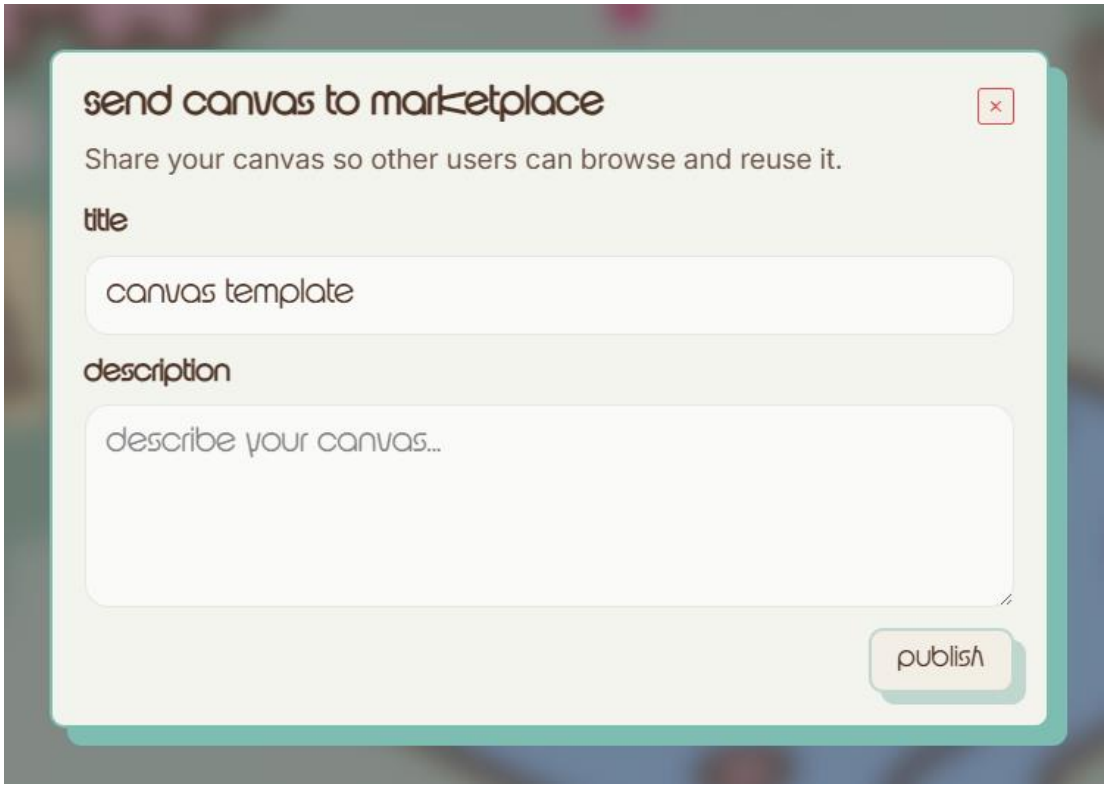


Figure 24 Modal Design.

Consistency across pages was maintained through reusable components and structured layout rules. This helped users transfer knowledge between different parts of the system without needing to relearn interface behaviours, improving the overall usability of the application.

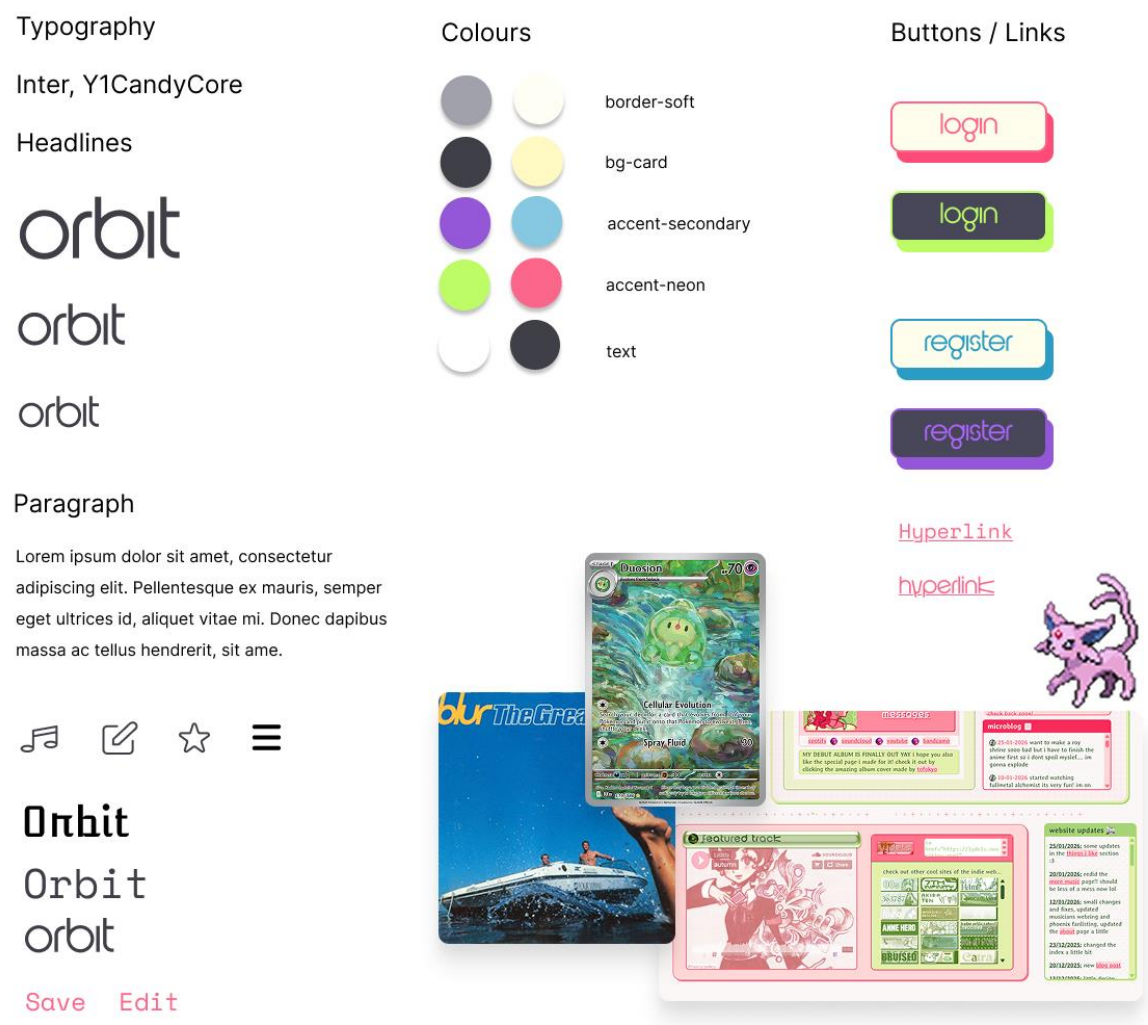


Figure 25 Style Guide

4.2.5 Figma Make

During the design phase, Figma Make was used as an exploratory tool to overcome design blocks. Rather than using the generated outputs as final designs, the tool was used to compare alternative interface layouts, styling approaches, and component arrangements against my own design direction. This helped identify which parts of the

interface were working well and which areas needed refinement, particularly in relation to layout clarity, visual hierarchy, spacing, and consistency.

Figma Make was therefore used as a supporting design aid, not as a replacement for the design process.

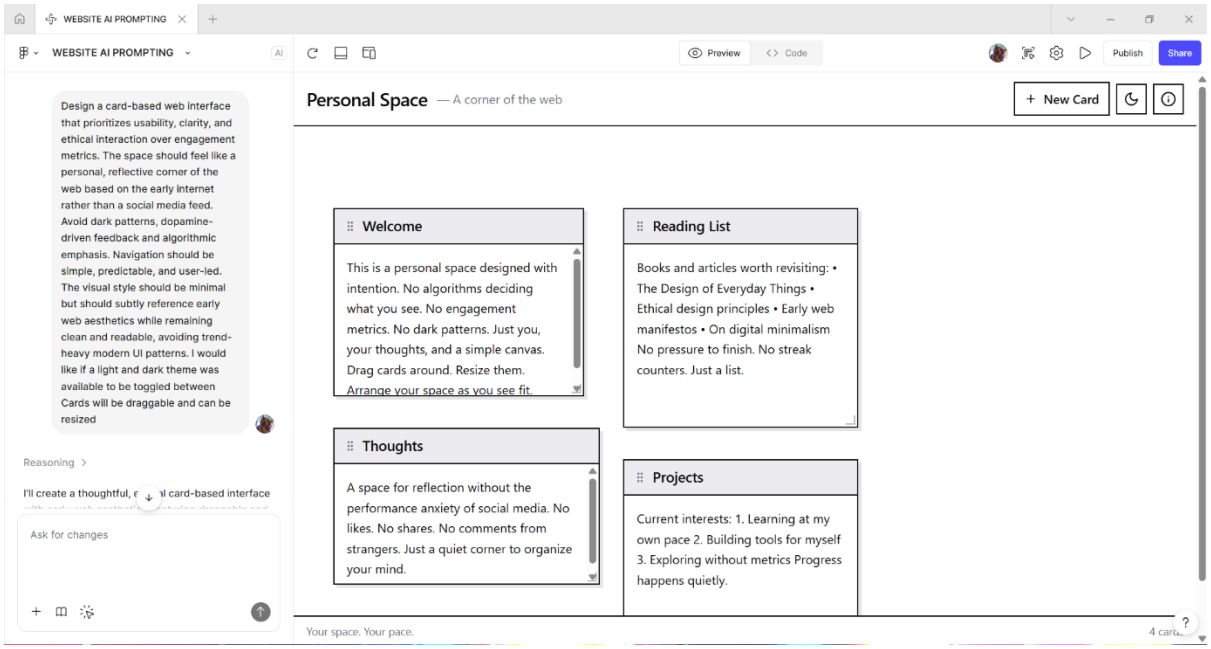


Figure 26 AI prompting in Figma Make

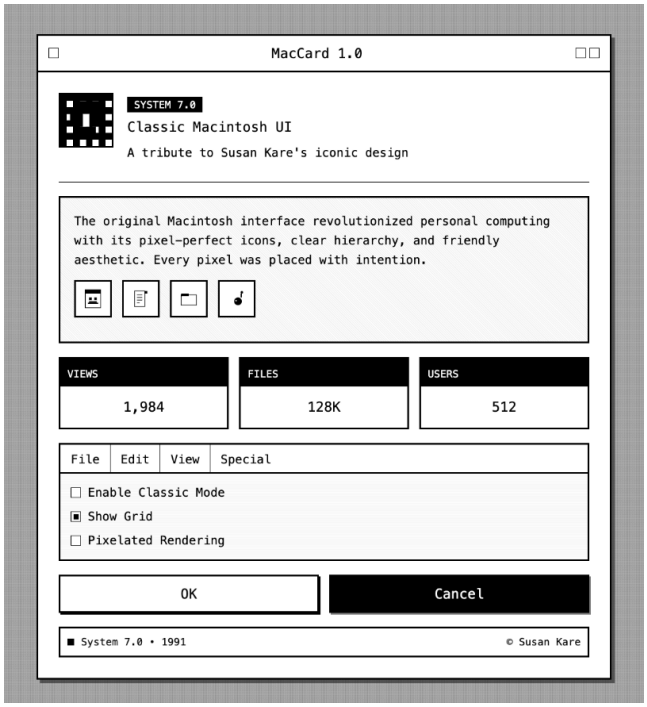


Figure 27 Figma Make Mac 1984 inspired card design.

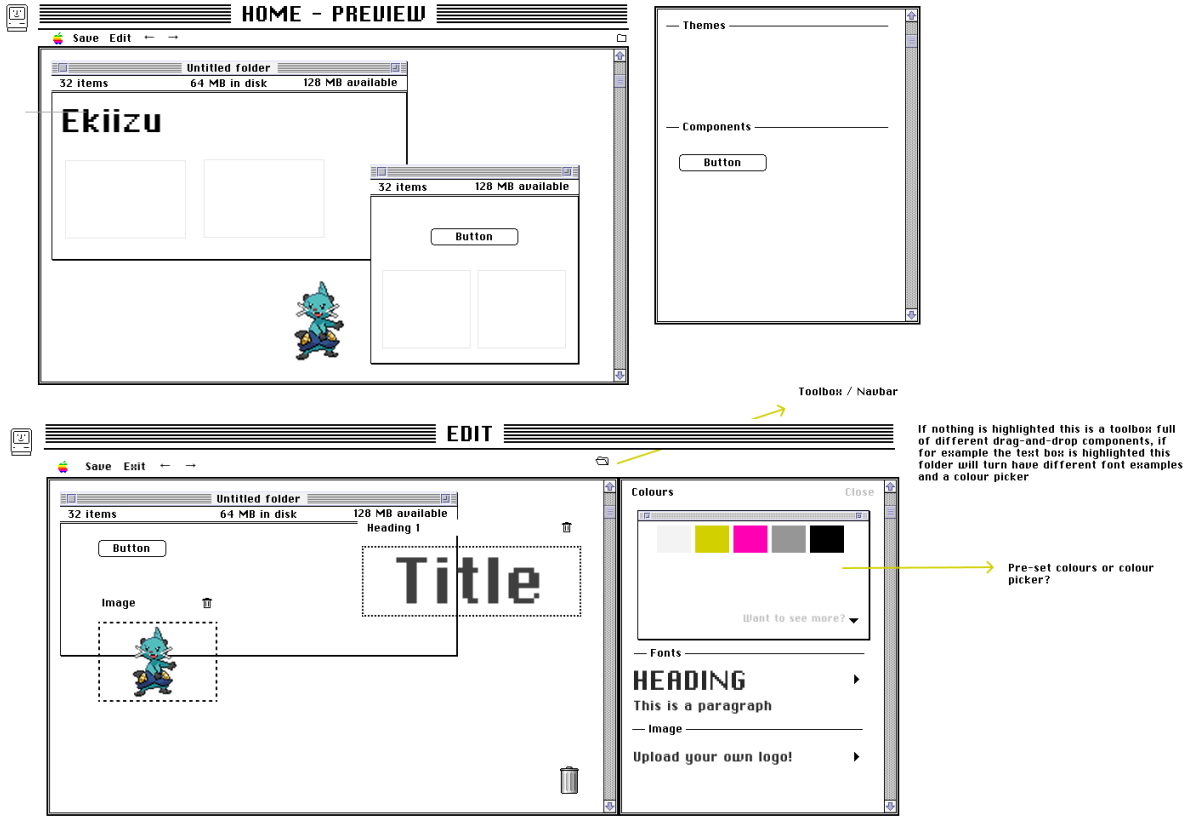


Figure 28 Figma inspired Mac 1984 theme designed by me.

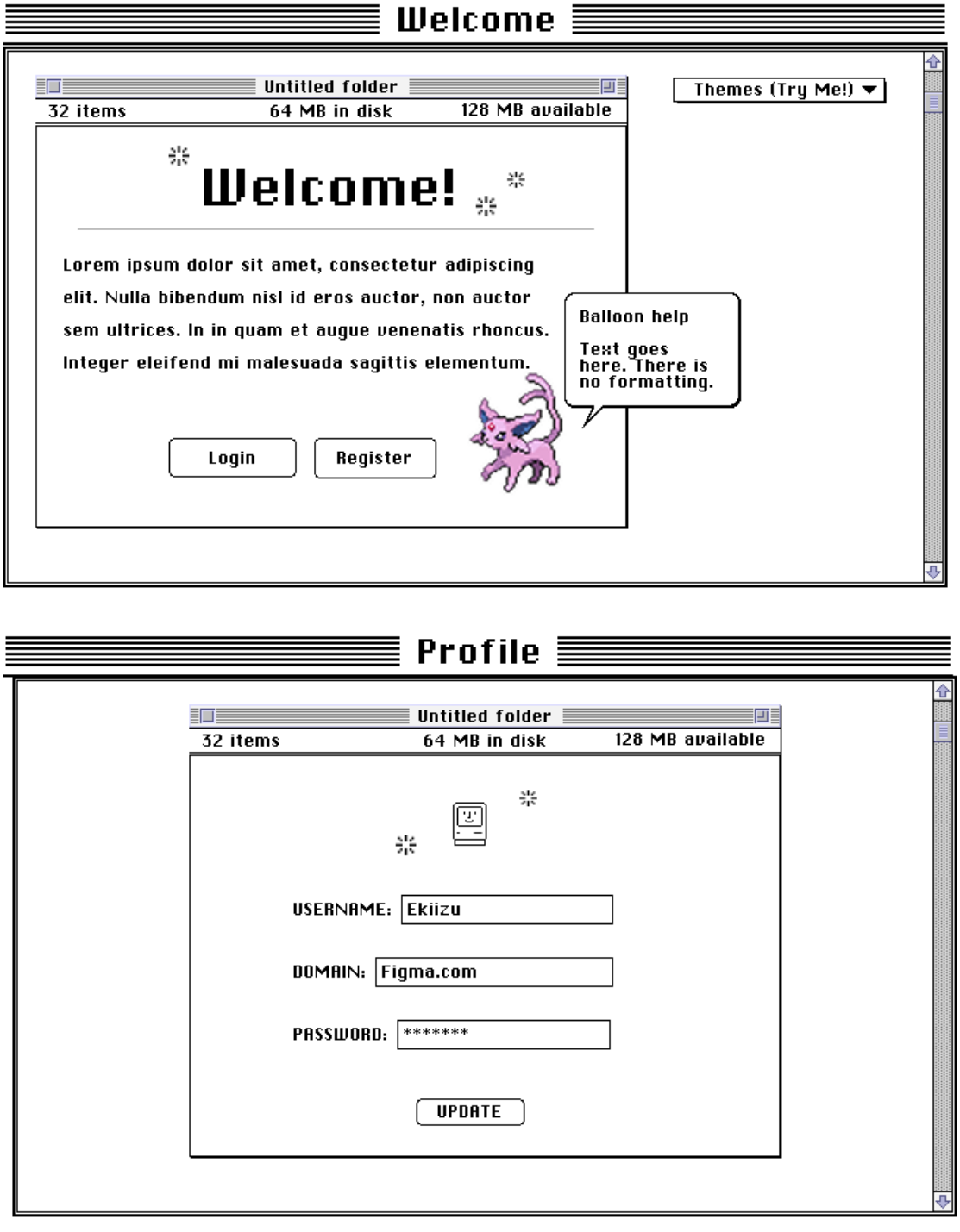


Figure 29 Figma inspired Mac 1984 theme designed by me.

Further research using Figma Make can be seen within the AI log in Miro.

4.2.6 High Fidelity Prototypes

Figure 30 Initial Homepage Design in Figma.

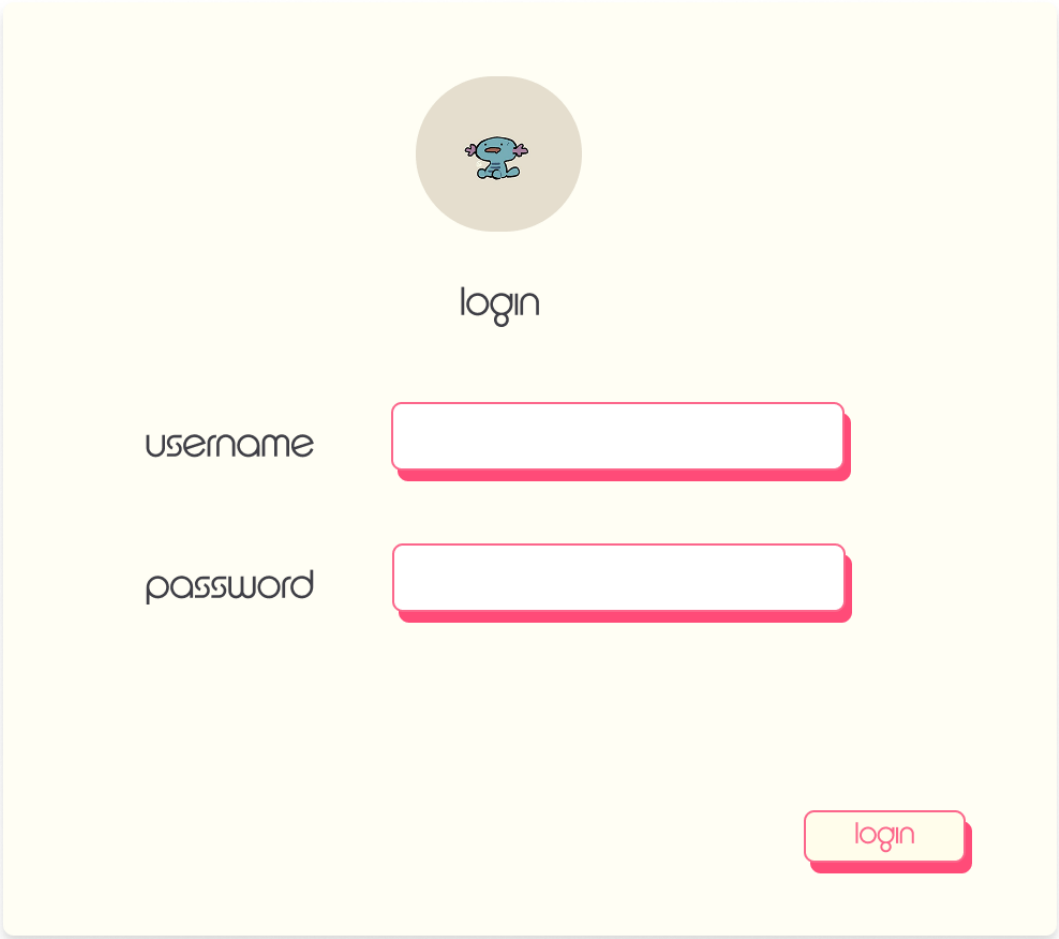


Figure 31 Initial Login Design in Figma.

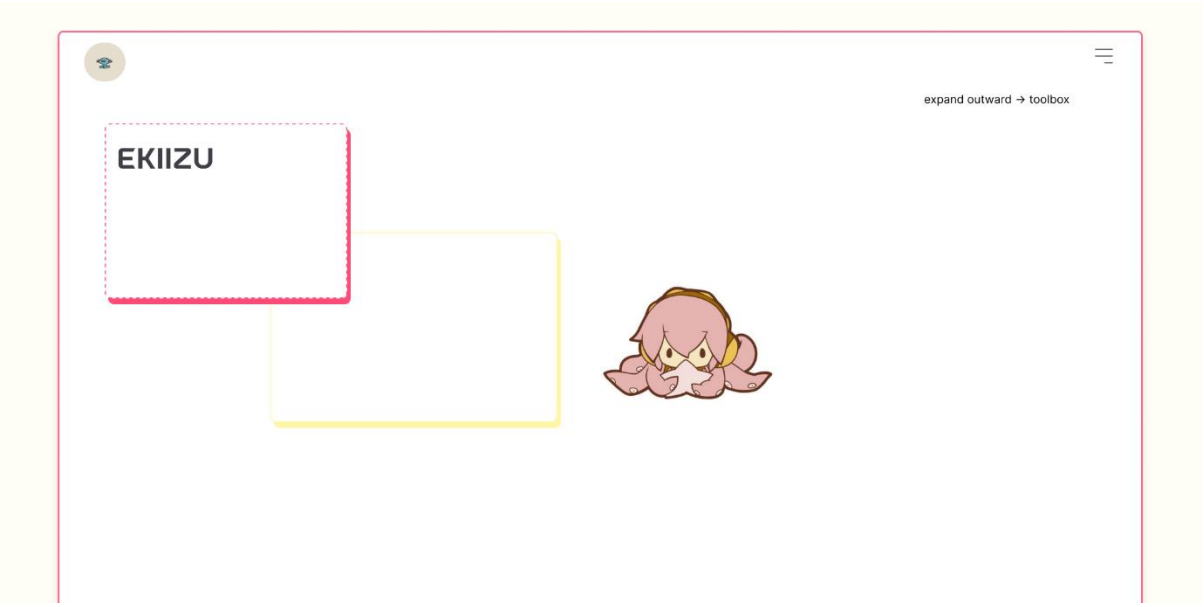


Figure 32 Initial Canvas Design in Figma.

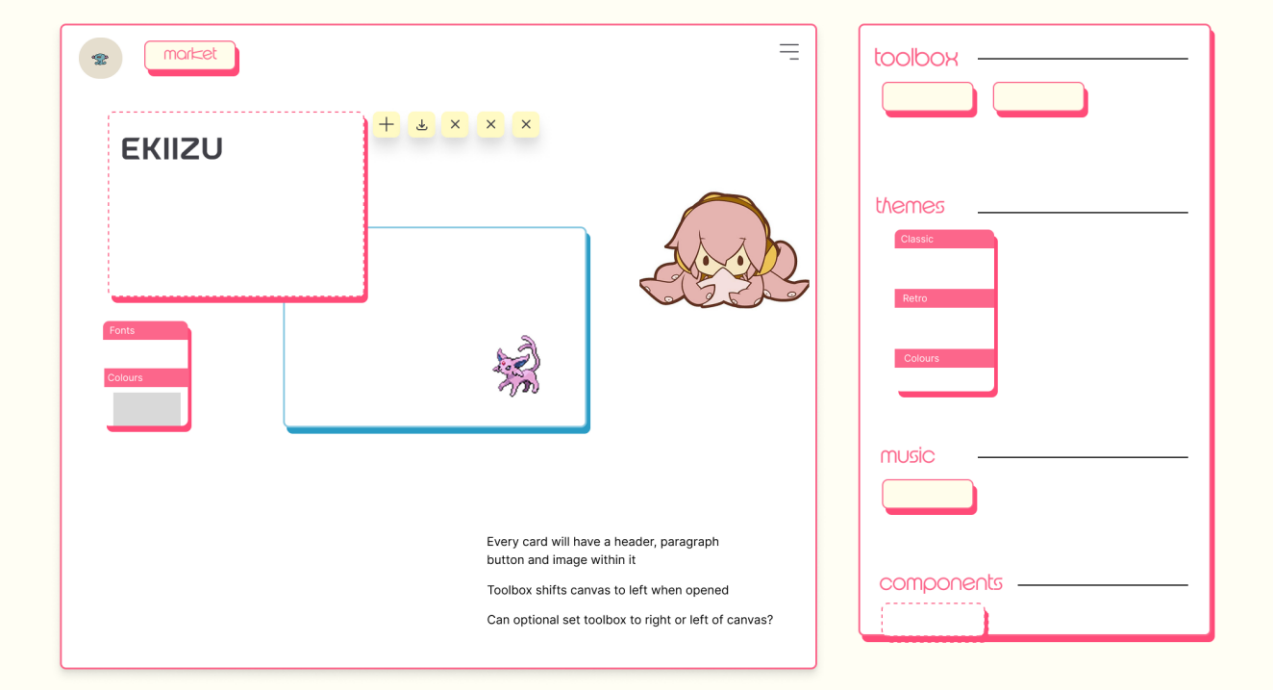


Figure 33 Initial Canvas Design with Toolbox in Figma.

4.3 Process Design:

4.3.1 Frameworks, Libraries and System Logic

The system was built using React and TypeScript, with React responsible for rendering the user interface and managing component state. This was suitable for Orbit because the application is made up of many reusable interface components, such as canvas nodes, toolbars, modals, theme controls and marketplace cards.

Orbit uses Supabase as its backend platform. Supabase is an open-source Backend-as-a-Service platform that provides a hosted PostgreSQL database, authentication, file storage, API generation and Row Level Security. This made it suitable for the project because it allowed the application to store user accounts, profiles, canvas data, guestbook entries and marketplace content without requiring a custom backend server.

The drag-and-drop canvas system was implemented using *dnd-kit*. This library was used to make toolbox items draggable, allow existing nodes to be repositioned, and provide drag overlays for visual feedback. The canvas logic distinguishes between new components dragged from the toolbox and existing nodes already placed on the canvas.

html2canvas was used for marketplace preview generation. When a user submits a canvas or card to the marketplace, the relevant HTML element is captured as an image, converted into a PNG blob and uploaded to Supabase Storage. This allows marketplace items to show a visual preview as well as storing the reusable template data.

The application also uses custom utilities and hooks to keep repeated logic organised. These include helpers for theme application, signed image URLs, sound feedback, authentication state, embed URL conversion and canvas boundary clamping. This helped keep the main canvas logic more manageable and made repeated behaviours easier to reuse across the system.

4.4 Database Design:

4.4.1 Entity-Relationship Diagrams (ERDs)

The database design went through several changes during early development. Initially, the ERD separated each canvas component into its own table, with different tables planned for cards, images, links and other component types. However, this became unsuitable as the canvas system developed because each node needed to share common properties such as position, size, z-index and ownership. Creating a separate table for every component type would have made the system harder to maintain and extend.

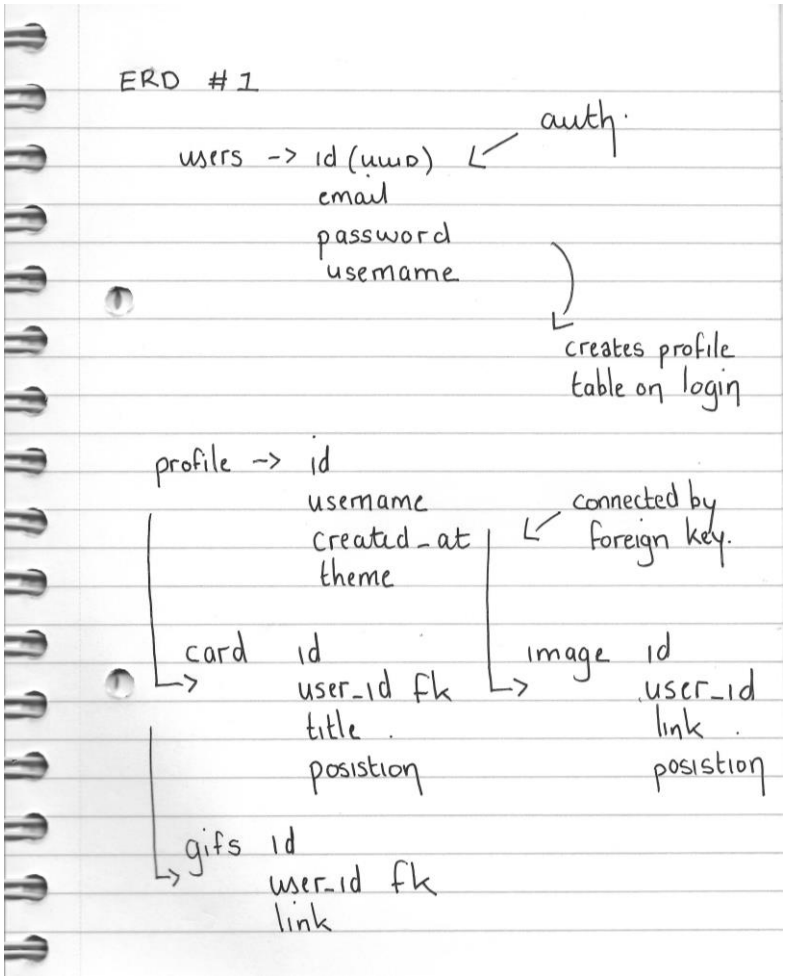


Figure 34 ERD #1

The final design uses a more flexible structure centred around a *canvas_nodes* table. Each node stores shared data such as id, user_id, type, x, y, w, h, z and props. The type field identifies which component should be rendered, while the props field uses JSONB to store flexible node-specific data. This was useful because different node types require different attributes. For example, a text node may only need text content, while an image node needs a file path and name, and a card node may contain a title, description, buttons and embedded blocks.

Core relationships such as users, profiles, canvas nodes, guestbook entries and marketplace items are modelled relationally, while flexible component content is stored inside JSONB fields.

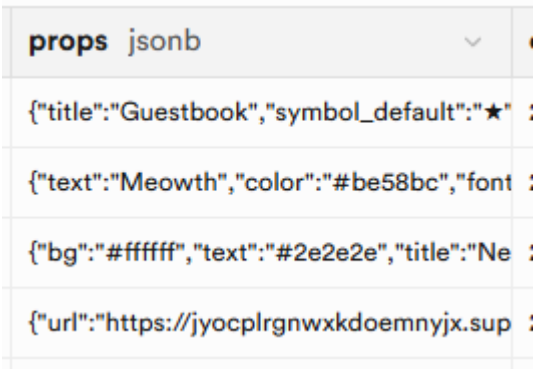


Figure 35 How props work in the database.

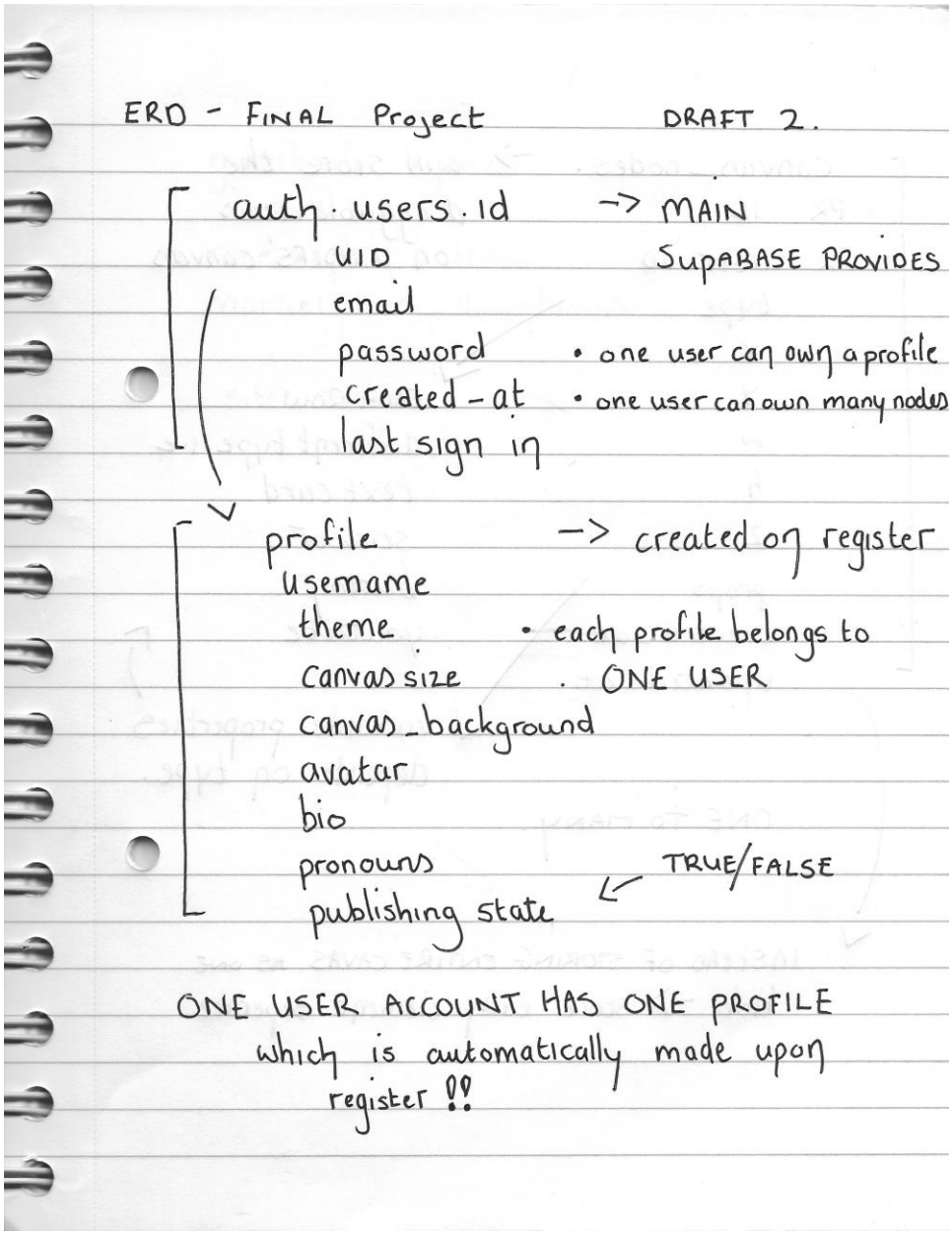


Figure 36 ERD #2

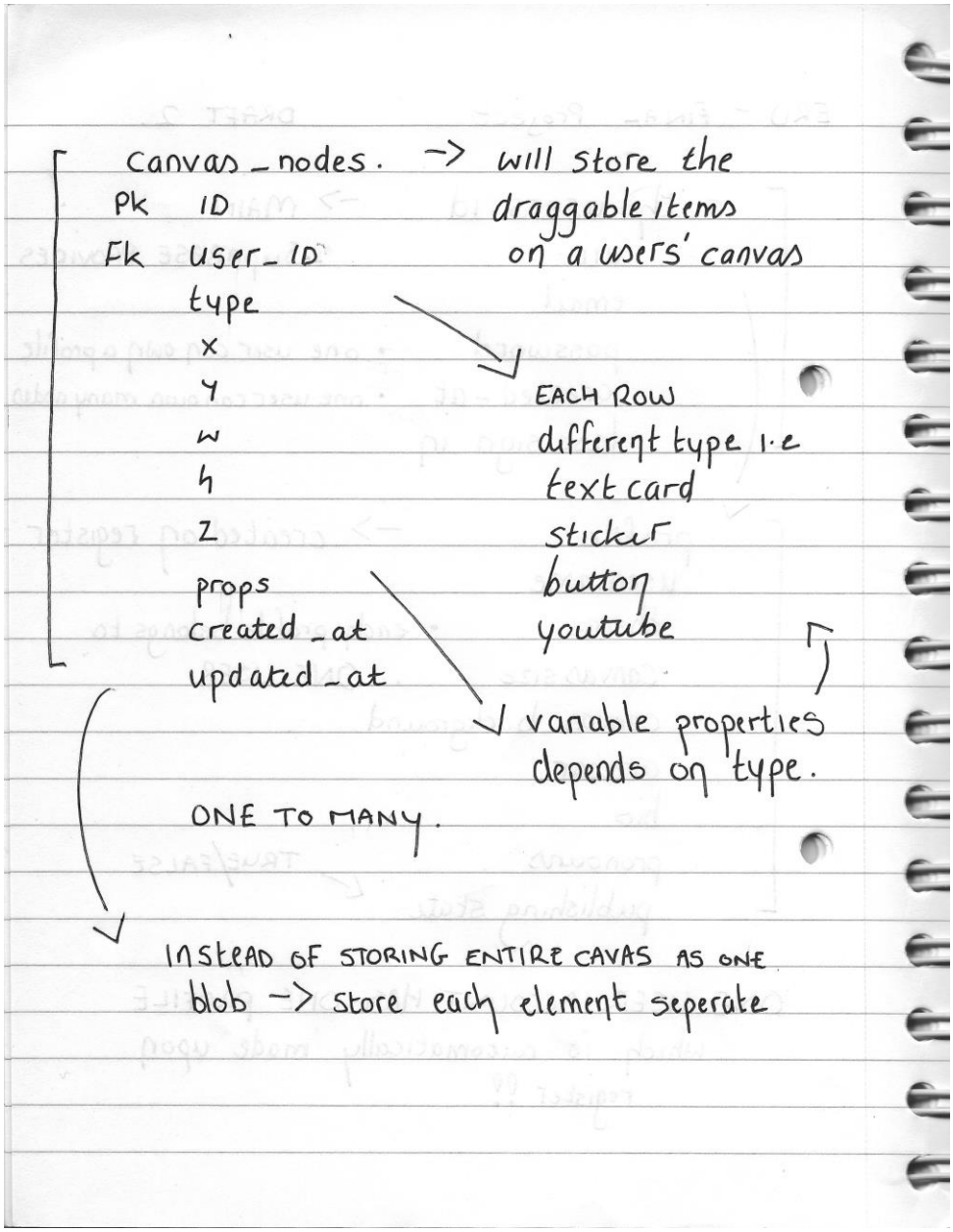


Figure 37 ERD Draft #2 continued

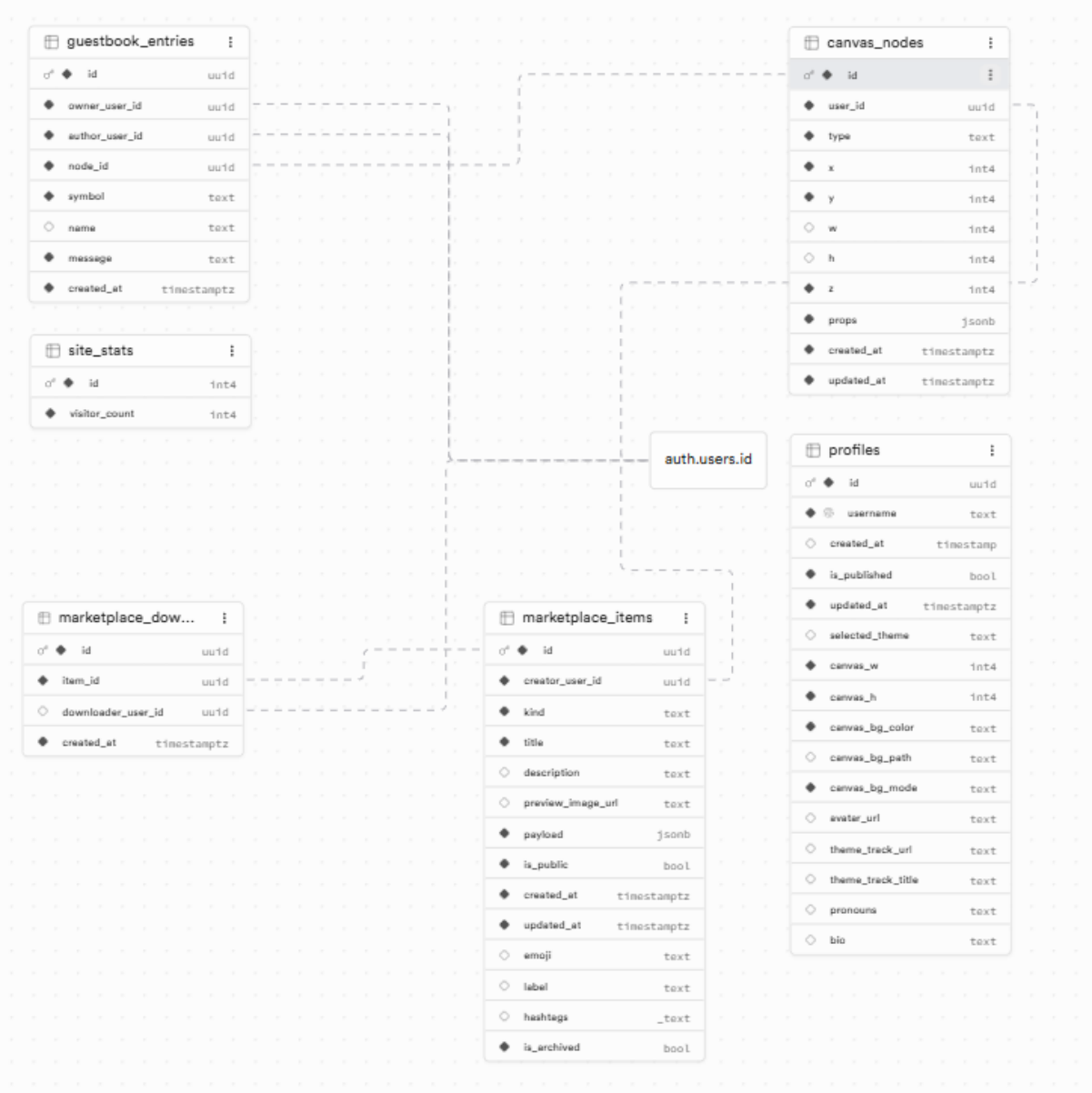


Figure 38 Final Database in Supabase

4.4.2 Database Design Rationale

A relational database was selected because the application relies on clear relationships between users, profiles, canvas nodes, guestbook entries and marketplace items. Each user owns a profile, each profile can have many canvas nodes, and marketplace or guestbook data must be linked back to the correct user. Using PostgreSQL through Supabase allowed these relationships to be modelled clearly and securely.

Foreign keys were used to protect data consistency. For example, canvas nodes and profile data are linked to the authenticated user ID, meaning all user generated content must belong to a real account. This reduced the risk of invalid data being stored in the system.

Supabase Authentication also simplified user management. Instead of manually storing passwords or creating a custom user system, Orbit uses Supabase's built-in auth.users table. The application then references each user through their unique UUID. This improved security because password handling, session management and authentication were managed by Supabase rather than being built manually.

SQL tables also made querying and persistence easier. Canvas nodes could be loaded by filtering by user_id, public profiles could be found using a unique username, and marketplace items could be queried separately from user canvas data. This structure made it easier to save, load, update and delete content consistently across the application.

5. Implementation:

5.1 Development Methodology and Process

The system was developed using an iterative, sprint-based methodology across two-week development cycles. Each sprint had a goal, planned features, and a review of technical challenges. This allowed the project to develop gradually from research and design into implementation, testing and then adding the final touches.

Progress was documented through GitHub commits, screenshots, videos, Figma files and Miro notes.

Sprint 1: 23rd January – 6th February

Goal

This initial sprint focused on establishing how the application would be structured. This involved the visuals of the application, the supporting research to justify this project and the technical foundation.

Key features implemented

- Initial wireframes and user flows were designed on paper.
- Similar applications were researched to inform the project requirements.
- User survey findings were reviewed to support design decisions
- Visual themes and layout ideas were explored in Figma and Figma Make
- React, Typescript and Vite were selected for the front-end.
- Usability heuristics were researched to guide interface decisions.
- Susan Kare's interface and icon design work was researched to support the project's visual direction.
- A small test prototype was created to explore how *dnd-kit* handled draggable interaction.
- The GitHub repository was configured for version control.
- Personas created to define the target users and user needs.

Relevance

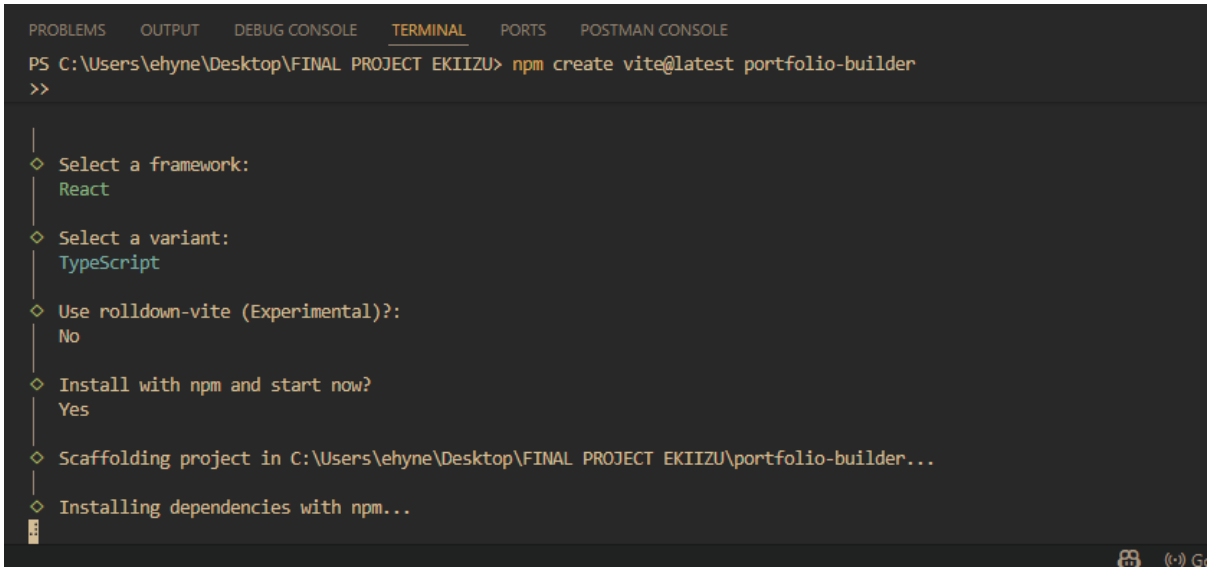


Figure 39 Installing React with Typescript using Vite

Sprint 1 was primarily a research and design sprint rather than a coding sprint. The main output was the visual and structural foundation of the application, including Figma wireframes, user flows, personas, theme exploration, and research into similar applications. This work is discussed in more detail in the previous chapters, but it is included here because it directly influenced later implementation decisions, including

the use of a draggable canvas, reusable node components, and the theme based interface.

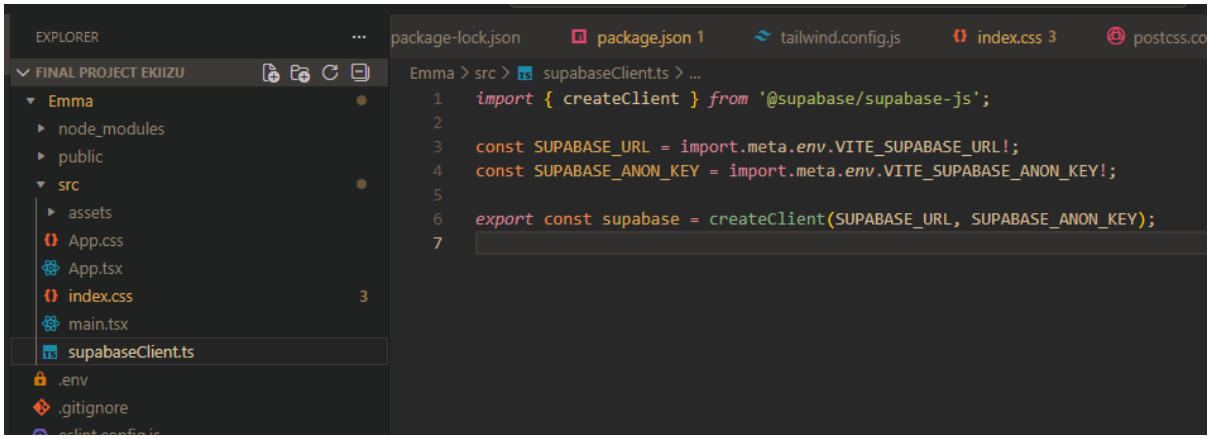


Figure 40 Supabase connection.

The initial React and TypeScript scaffold was created using Vite, including the early component structure and development environment. The database and front-end were also connected at this stage, establishing the foundation for persistent user data later in development.

Technical Challenges and Solutions

The project initially became delayed because a clear design direction had not yet been solidified. The challenge was creating an interface that felt expressive and customisable while still being realistic to the project's timeframe. This was addressed through further design exploration in Figma, Figma Make and Miro.

Research into UI designers such as Susan Kare, similar applications, and Indie Web examples helped guide the visual direction of the project. Figma Make was also used to explore how a drag-and-drop system could work visually and how it could remain user-friendly. Figma was then used to test different theme options on the application structure, including themes suggested through survey feedback. This can all be seen in previous chapters.

Once the design direction became clearer, a small React prototype was created to test how *dnd-kit* handled draggable interaction. This helped confirm that the chosen technical approach was feasible before the full canvas system was implemented.

Figure 41 React prototype to test dnd-kit, no back-end connection at this stage.

Additional screenshots documenting this development process can be found in Miro.

Outcome

By the end of the sprint, the project had a clear research base, personas, initial user flows, wireframes, selected visual themes, a chosen technical stack, a working React/TypeScript/Vite scaffold and an early *dnd-kit* test prototype.

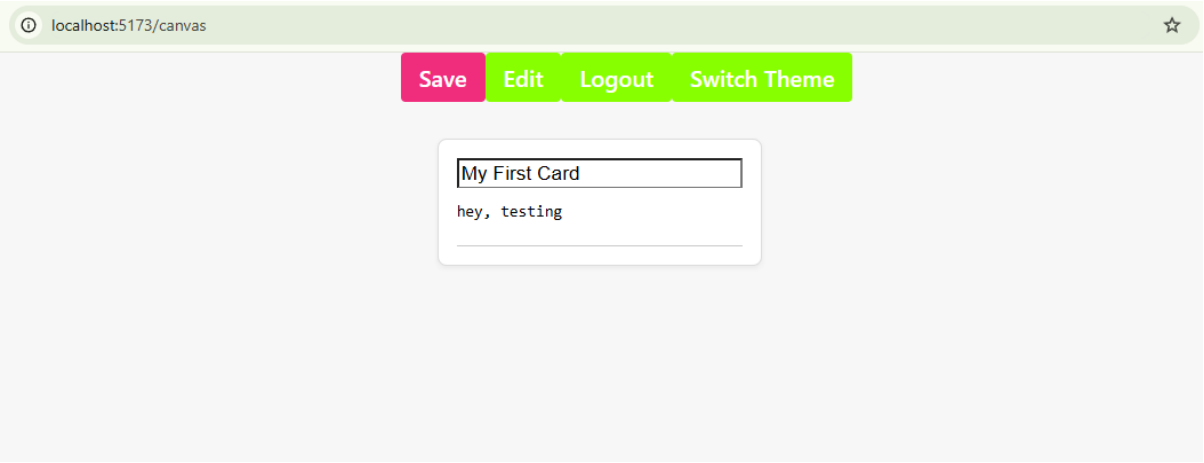


Figure 42 First working test of the application.

Sprint 2: 6th Feb – 20th February

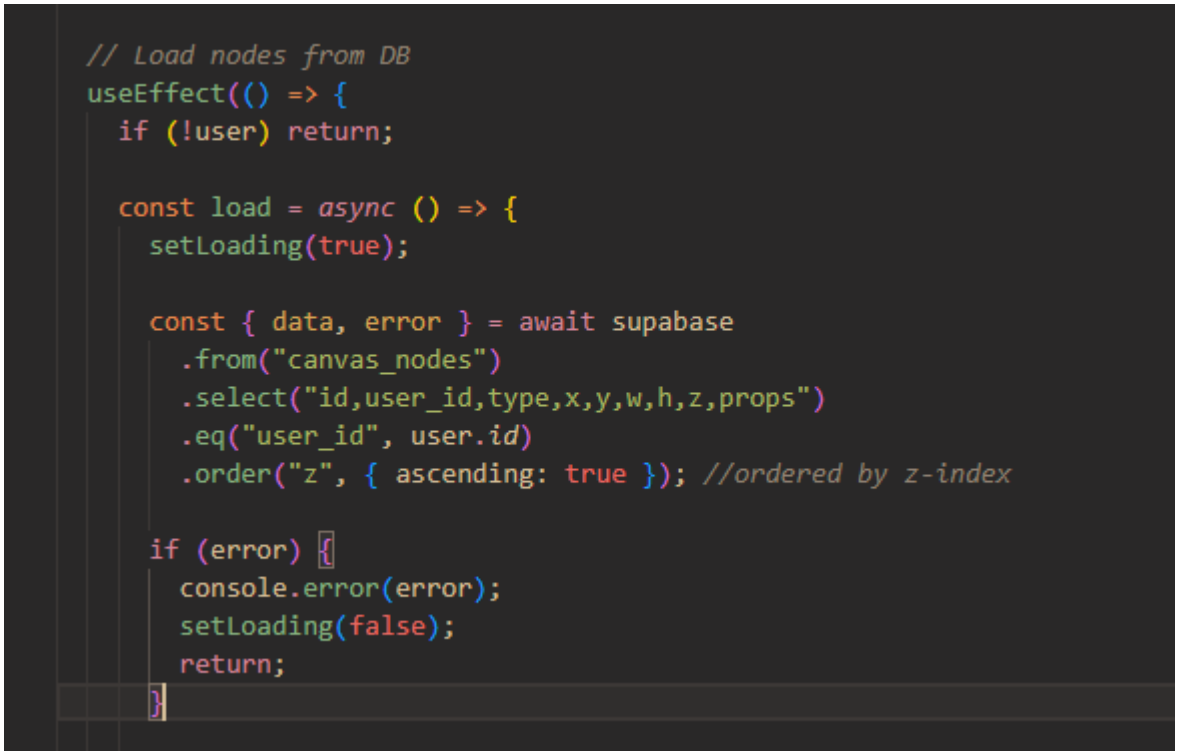
Goal

Design and implement the initial back-end structure required to support user accounts, canvas ownership and persistent data. This was extremely important as each user's canvas would need to remember what they added to the canvas and where they were placed.

Key Features implemented

- Refine ERDs and planned the database structure.
- Set up Supabase authentication, database tables and Row Level Security policies.
- Replaced the original *single cards* table idea with a stronger relational structure.
- Implemented the *profiles* and *canvas_nodes* tables.
- Connected user-generated data to Supabase *auth.users.id*.
- Ensure user authentication works as intended
- Added early drag-and-drop persistence for canvas cards.
- Implemented username-based public profile routes instead of ID-based routes.
- Added database-level username uniqueness.
- Began implementing theme switching, persistence and public viewing.
- Canvas components persisting in database

Code



```
// Load nodes from DB
useEffect(() => {
  if (!user) return;

  const load = async () => {
    setLoading(true);

    const { data, error } = await supabase
      .from("canvas_nodes")
      .select("id,user_id,type,x,y,w,h,z,props")
      .eq("user_id", user.id)
      .order("z", { ascending: true }); //ordered by z-index

    if (error) {
      console.error(error);
      setLoading(false);
      return;
    }
```

Figure 43 How nodes load from the database.

The main code focus was connecting front-end user actions to persistent Supabase data. Canvas elements were stored in the canvas_nodes table, with fields for the node type, position, size, rotation, layering, and props. When a user loaded their canvas, the application queried the nodes associated with their account and reconstructed the layout on the front end.

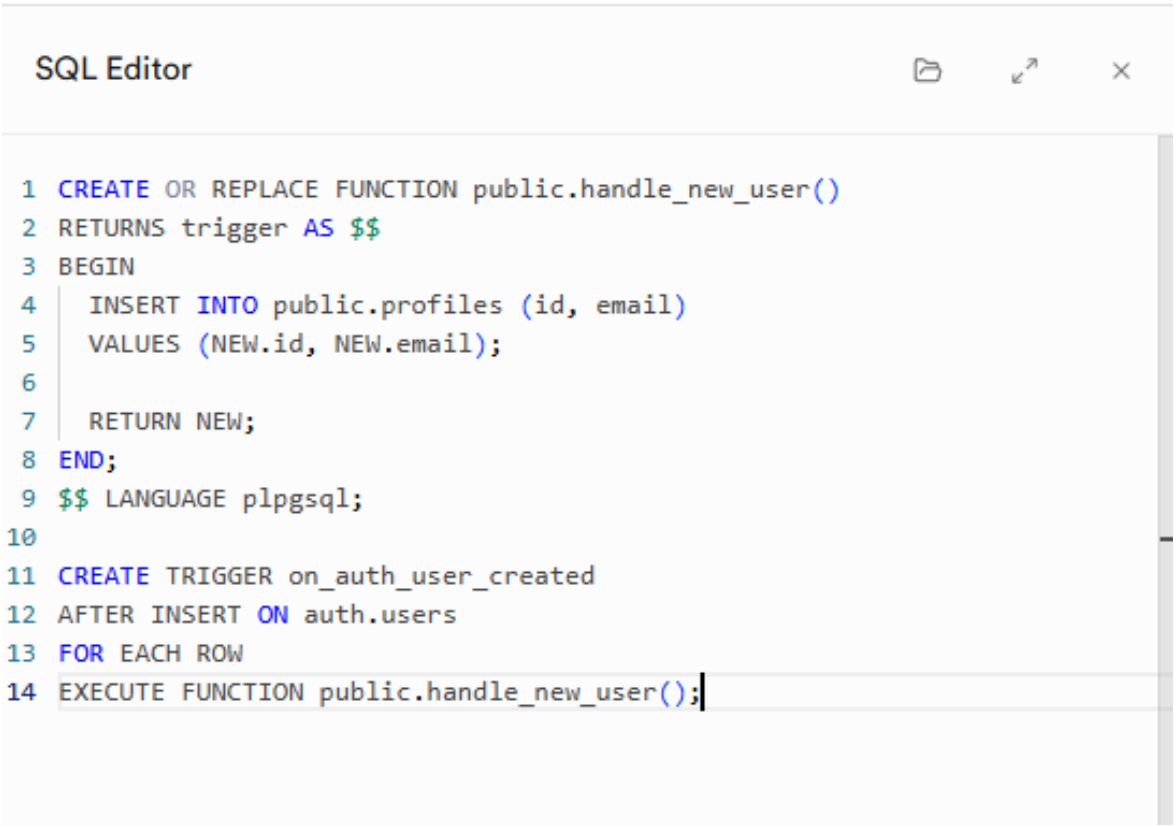


Figure 44 PostgresSQL trigger for unique usernames.

User profiles were also connected to authentication. Instead of manually creating a separate users table, the application uses Supabase's built-in auth.users table. A PostgreSQL trigger was added so that every new authenticated user automatically received a matching profile record. This helped fix issues where the application could crash after registration if a profile did not yet exist.

```css
:root {
  color-scheme: light dark;
  font-family: MyCustomFont, apple-system, Segoe UI, Roboto, Arial, sans-serif;
  line-height: 1.4;

  --bg-primary: #ffffff;
  --panel-bg: #fffbea;
  --bg-card: #ffffff;
  --text-primary: #2e2e2e;
  --text-secondary: #94cc4b;

  --accent-neon: #ff4f7a;
  --accent-secondary: #94cc4b;
  --border-soft: #f3edb3;

  --shadow-offset: 8px;
  --shadow-color: rgba(0, 0, 0, 0.06);

  --workspace-bg: #ffffff;
  --workspace-radius: 18px;
  --workspace-shadow: 0 18px 35px rgba(0, 0, 0, 0.12);
  --border: rgba(0, 0, 0, 0.08);

  --control-bg: #fffdeb;
  --control-hover: #ff4f7a;
  --control-text: #2e2e2e;
}
```

Figure 45 CSS Variables for default theme.

```css
.auth-page {
  min-height: 100vh;
  display: grid;
  place-items: center;
  padding: 48px 20px;
  background: var(--bg-primary);
  color: var(--text-primary);
}
```

Figure 46 Example of CSS variables in use.

A second important code focus was the theme system. The ThemeProvider stored the active theme in React state, while the theme selector allowed users to change it. The selected theme was saved both locally and in Supabase, then applied to the root HTML element using a data-theme attribute. This allowed CSS variables in the :root theme blocks to restyle the interface without hardcoding colours into every component. For example, components can use var(--bg-primary), meaning the UI is not fixed to one colour set and instead reads from the reusable theme tokens.

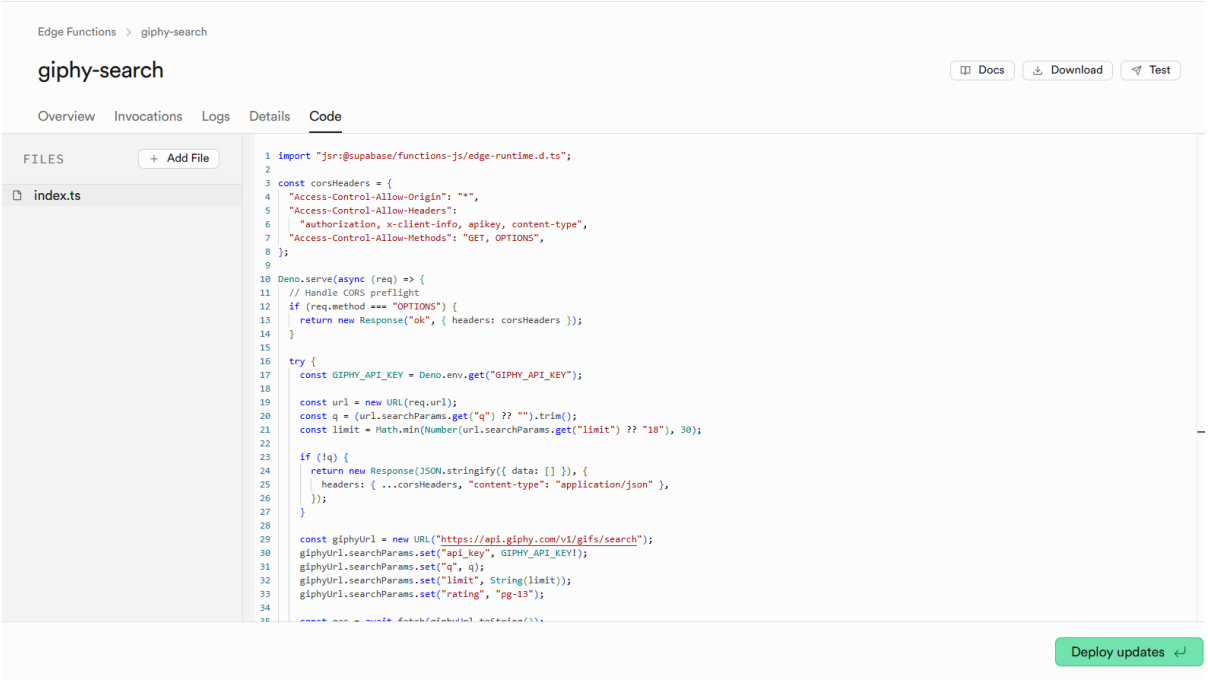


Figure 47 Giphy feature in Supabase Edge Function

After card persistence was working, the first media features were introduced. The Giphy search feature was implemented using a Supabase Edge Function. Instead of calling the Giphy API directly from the React front end, the user's search query was sent to the Edge Function. The Edge Function accessed the Giphy API key securely through Supabase environment variables, requested matching GIF results from Giphy, and returned the results to the front end in real time. This allowed users to search for GIFs while preventing the API key from being exposed. The selected GIF URL was then stored as part of the canvas node data so it could persist when the canvas was reloaded.

```
      setLoading(false);
      return;
    }

    const hydrated = await hydrateImageUrls((data ?? []) as CanvasNode[]);

    setNodes(hydrated);

    setLoading(false);
  };

  load();
}, [user]);
```

Figure 48 HydratedImageUrl which converts Supabase storage paths into usable images URLs.

User image uploads were implemented using Supabase Storage. When a user uploaded an image, the file was stored in a Supabase storage bucket, while a stable file path was saved in the database as part of the relevant canvas node or profile. This separated the uploaded file itself from the database metadata and avoided storing large image files directly in the database.

Hydrated image URLs were then used when loading the canvas or profile. Instead of relying only on a saved URL, the application used the stored Supabase file path to generate a fresh displayable image URL. When canvas nodes are loaded from Supabase, the returned node data is passed through hydrateImageUrls() before being stored in React state. This function checks for image nodes and converts their saved Supabase Storage paths into usable image URLs. The hydrated nodes are then saved with setNodes(hydrated), allowing uploaded images to appear correctly after the page is refreshed. This approach also helped avoid issues with temporary or expired image links, as the permanent value stored in the database was the file path rather than the generated display URL.

Technical Challenges and Solutions

The first challenge was the original database design. The initial plan used a single card-based table with JSON fields for media, buttons, and layout data. Although this was flexible, it made the system harder to query, scale, and secure. This was solved by moving to a more relational structure using separate tables for profiles, canvas nodes, and later interaction-based features. This is discussed in more detail in Chapter 4.

One issue was that the selected user theme did not persist correctly after the canvas was saved. This happened because the theme initially only existed in local state, localStorage, or the DOM. When another user visited the public profile route, the app did not know which theme belonged to that profile, so it fell back to the default styling. This was solved by saving the selected theme to Supabase as part of the user's profile data, allowing the public view to load and apply the correct theme.

Another challenge was ensuring that user data was secure. Row Level Security was enabled so users could only access or modify their own canvas data. This meant security was enforced directly at database level rather than relying only on front-end checks.

Figure 49 UNIQUE constraint for username field.

A further issue was public profile routing. The application originally used user IDs in public URLs, but this was changed to usernames to make profiles easier to access and share. To prevent duplicate usernames causing routing conflicts, a UNIQUE constraint was added to the username field in the profiles table. Enforcing this constraint at the database level ensured that each user had a distinct public identifier and improved data integrity.

Hydrated image URLs were also added because uploaded image nodes were failing to display after the canvas was refreshed. This happened because the database stored the stable Supabase Storage path rather than a browser ready image URL. Hydration solved this by converting stored file paths into fresh displayable URLs when the canvas loaded. This avoided issues with expired signed URLs and made image persistence more reliable.

Outcome

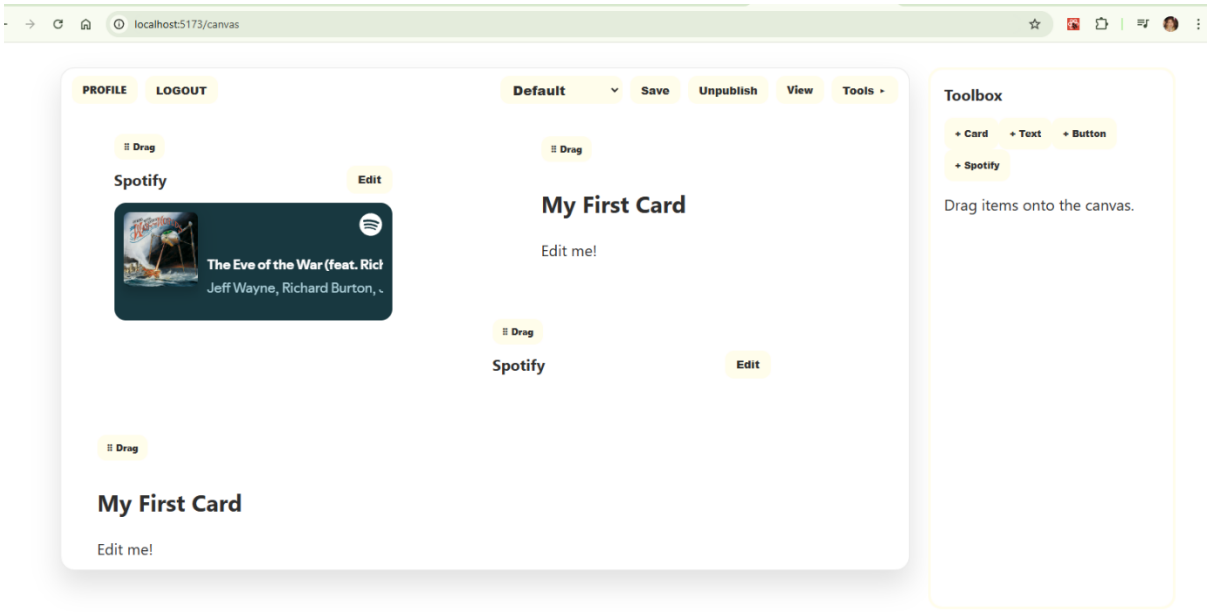


Figure 50 Canvas at the end of Sprint 2.

By the end of this sprint, the application had a secure Supabase back-end, automatic profile creation, username based public profile routing, early canvas persistence, embedded media support, GIF support through the Giphy API, and a theme system that could persist user selected themes across both the canvas edit and public views.

Sprint 3: 20th Feb – 6th March

Goals

The main aim of this sprint was to get the core functionality working, such as the components being drag and drop. At this point, cards, gifs, and images were saving and persisting in the database as node types. A marketplace system was also implemented.

As basic features were working, Figma designs were revisited. The design off this application was a large part of its identity so it was crucial that it was consistent and strong

Key Features Implemented

- Implemented the core draggable canvas system using dnd-kit.
- Added YouTube embed nodes.
- Added guestbook nodes, including a guestbook table in the backend.
- Expanded canvas customisation with canvas backgrounds.
- Added user profile pictures, with image data persisting in the database.
- Improved the editing system for canvas components.
- Added more card customisation, including extra font options and components inside cards.
- Added a transparent label component.
- Added full screen canvas functionality.
- Added a more detailed home view, toggled through a slider.
- Added background particles
- Reworked the main visual themes and improved font colour contrast where text or images were difficult to see.
- Began implementing the marketplace feature, allowing users to upload a full canvas template or a single card design.
- Identified image issues in the marketplace feature, meaning it remained incomplete at this stage.
- Planned further work on the marketplace and user profile section.
- Fixed React Router SPA routing issues for deployment.
- Hosted the project on Render and tested the production build.
- Improved mobile view after hosting and testing the app in a production environment.

Code focus

```
<DndContext
  onDragStart={onDragStart}
  onDragEnd={onDragEnd}
  onDragCancel={onDragCancel}
>
```

Figure 60 DndContext for dnd-kit.

The main code focus of this sprint was the drag-and-drop movement system using dnd-kit. The DndContext acted as the wrapper for the draggable area and provided access to the drag event lifecycle, including when a drag started and ended. During dragging, the node's actual x and y values were not immediately changed in the database. Instead, CSS transform was used to move the node visually, which made dragging feel smoother and more responsive. When the drag ended, the final position was calculated and saved to Supabase.

```
<div
  ref={setNodeRef}
```

Figure 61 Node ID.

```
const style: React.CSSProperties = {
  position: "absolute",
  left: node.x,
  top: node.y,
  zIndex: node.z,
  transform: transform
    ? `translate3d(${transform.x}px, ${transform.y}px, 0)`
    : undefined,
  opacity: isDragging ? 0.85 : 1,
  touchAction: "none",
};
```

Figure 62 CSS Transform during node movement.

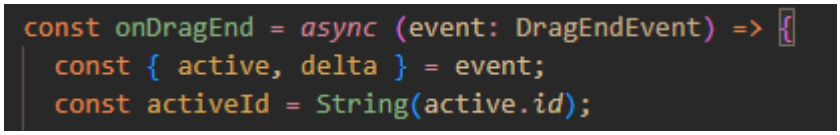


Figure 63 DragEnd after node movement.

Each node in the database follows a structure similar to:

{

id,

type,

x,

y,

w,

h,

z,

props

}

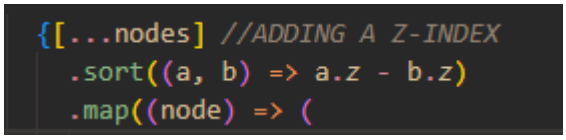


Figure 64 Node rendering

Figure 65 Different node types and how they look when they render on the canvas.

The type field controls which component is rendered on the canvas. For example, when node.type === "card", the application renders the CardNode component. All nodes are rendered within the canvas container, which keeps interaction limited to the workspace area. This helped prevent components from being placed outside the canvas and made the editing experience more controlled.

SQL Editor

```sql
create table if not exists public.guestbook_entries (
  id uuid primary key default gen_random_uuid(),

  owner_user_id uuid not null references auth.users(id) on delete cascade,
  author_user_id uuid not null references auth.users(id) on delete cascade,

  node_id uuid not null references public.canvas_nodes(id) on delete cascade,

  symbol text not null default '★',
  name text,
  message text not null,

  created_at timestamptz not null default now()
);

create index if not exists guestbook_entries_owner_idx
  on public.guestbook_entries(owner_user_id);

create index if not exists guestbook_entries_node_idx
  on public.guestbook_entries(node_id);

create index if not exists guestbook_entries_created_idx
  on public.guestbook_entries(created_at desc);
```

Figure 67 Guestbook entries SQL and RLS Policies creation in Supabase.

The guestbook required a separate backend structure because guestbook entries were not simply visual canvas nodes. A guestbook node exists on the canvas, but each message needed to store additional information, including the message content, author, owner with profile connection, and timestamp. This allowed messages to be linked to the correct guestbook and displayed again when the profile was viewed.

The marketplace feature was also planned and implemented during this stage. Due to being unsure how to implement the marketplace without breaking the existing database structure, ChatGPT was used to help reason through the database design. A description of my existing tables was provided along with a description of what the marketplace was expected to do. Provided code was then taken and changed to suit. This process is documented on Miro in an AI log.

The marketplace allows users to share either a full canvas template or an individual card design. When a user chooses to send an item to the marketplace, the application saves the item into the *marketplace_items* table in Supabase. Each marketplace item stores information such as the creator's user ID, item type, title, description, preview image URL, public status, and payload data.

The payload is important because it contains the reusable structure of the shared item. For a full canvas, the payload stores the canvas settings, including canvas width, height, background colour, background image path, and selected theme, as well as an array of canvas nodes. For a card template, the payload stores the card node's type, size, and props. This means the marketplace does not only save a screenshot of the design, it also saves the data needed to recreate the item later.

```
const sanitizeNodeForMarketplace = (n: CanvasNode) => {
  const props = { ...(n.props ?? {}) };

  if (n.type === "images") {
    // keep path + metadata; drop url
    if ("url" in props) delete props.url;
  }

  return {
    type: n.type,
    x: n.x,
    y: n.y,
    w: n.w,
    h: n.h,
    z: n.z,
    props,
  };
}; //prepares node data before it goes to the marketplace, for
```

Figure 68 SanitizeNodeForMarketplace which removes temporary values before sending to database.

Before saving a canvas to the marketplace, each node is cleaned using *sanitizeNodeForMarketplace()*. This removes temporary values, such as signed image URLs, while keeping useful information such as the image path and node metadata. This was necessary because signed URLs can expire, so saving them permanently would cause images to break later. Instead, the stored path can be used to generate a new usable URL when needed.

Technical Challenges and Solutions

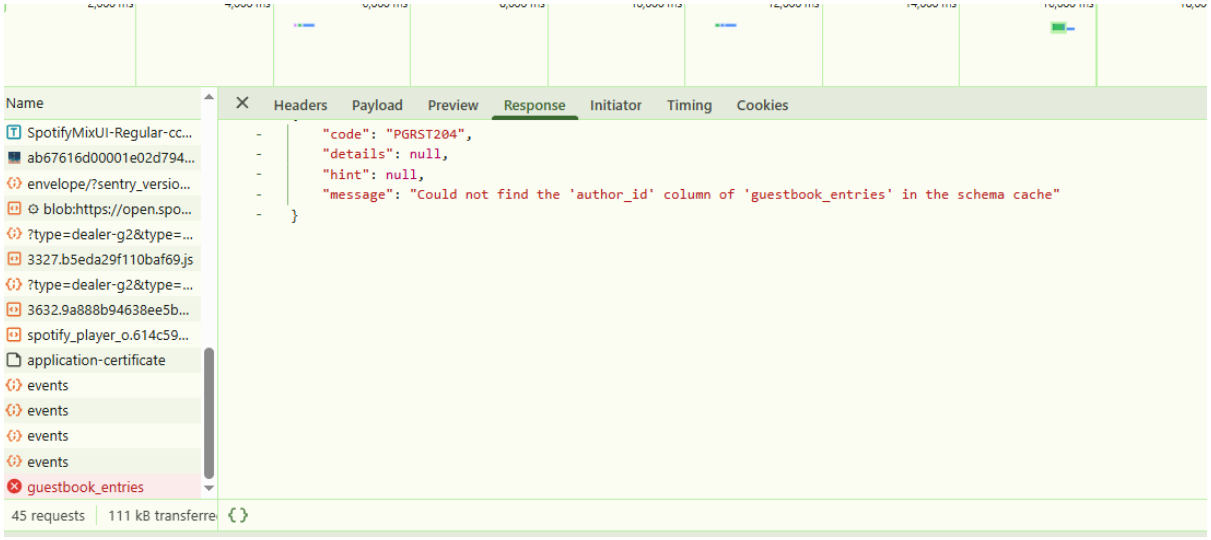


Figure 69

During development, an issue occurred with the guestbook feature where Supabase returned an error stating that the *author_id* column could not be found in the *guestbook_entries* schema cache. This happened because the frontend query expected a column that did not match the current database structure. The issue was resolved by checking the Supabase table schema which used *author_user_id*. This was a small issue that took far too long to resolve and was just an oversight due to forgetting what columns were named in Supabase.

Figure 71 Background image error.

An issue also developed when saving uploaded canvas background images. The image file was successfully uploaded to Supabase Storage, but the background path was not being correctly saved to the user's profile. The issue was traced to the Supabase update query, which needed to target the correct profile row using the authenticated user's ID. Adding the *.eq("id", user.id)* condition ensured that only the current user's profile was updated with the new *canvas_bg_path*. This fixed the connection between uploaded background images and persistent user profile data.

Routing issues were also encountered when hosting on Render. The application worked locally, but refreshing a nested route such as a public profile or marketplace page

caused errors because the server tried to find that route directly. This was caused by the way single page React applications handle routing. The issue was solved by configuring Render to redirect all routes back to the main *index.html* file, allowing React Router to handle the route on the client side.

Outcome

Overall, this sprint expanded the application from a basic canvas editor into a more complete system with persistent movement, guestbook interaction, marketplace sharing, and improved visual design.

By the end of sprint three this is how my database looked:

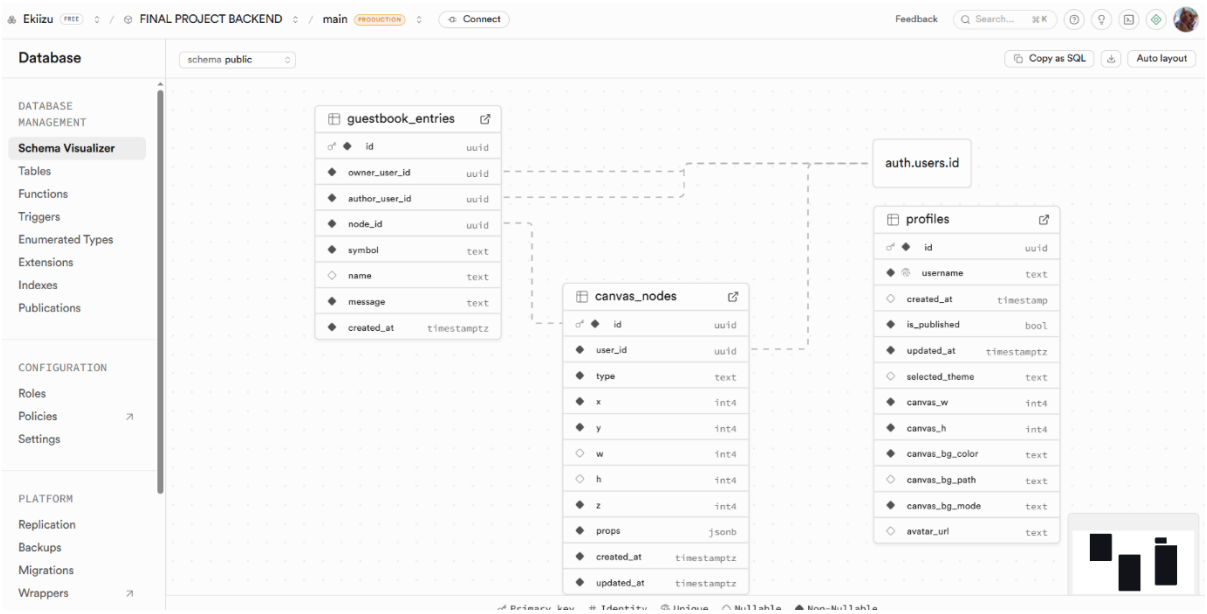


Figure 72 Database with Guestbook added.

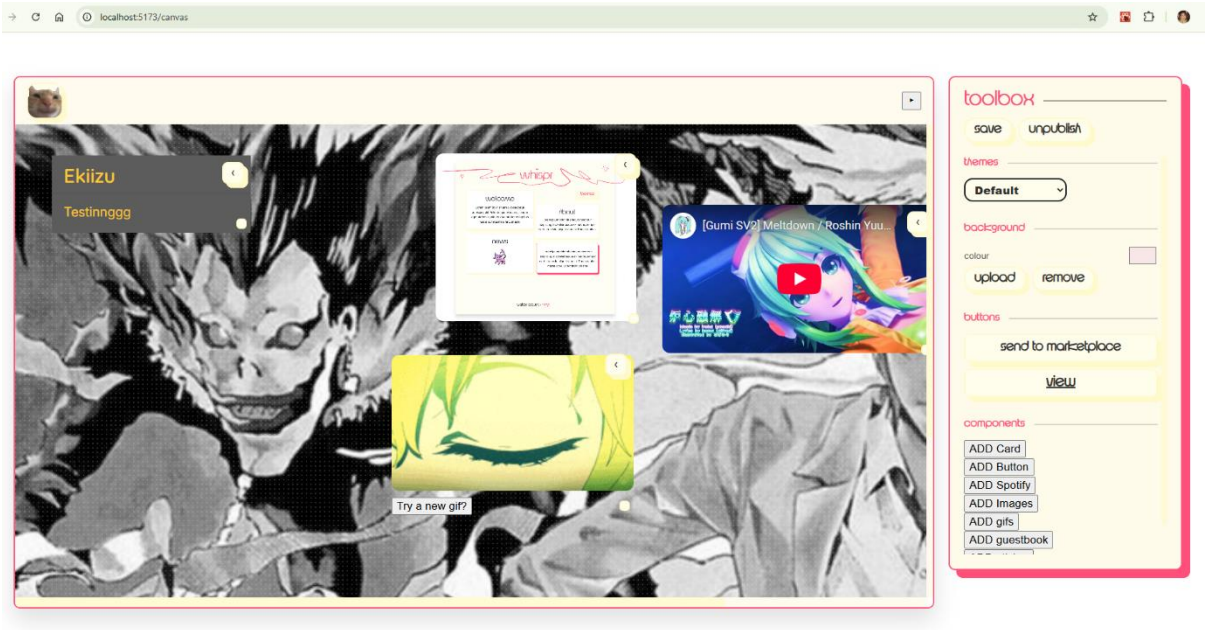


Figure 73 Canvas at the end of sprint 3.

Sprint 4: 6th March - 20th March

Goals

The main goal of this sprint was to expand the marketplace, improve user interactions through modals, allow users to add MP3 audio to their profiles, and implement z-index layering for canvas components.

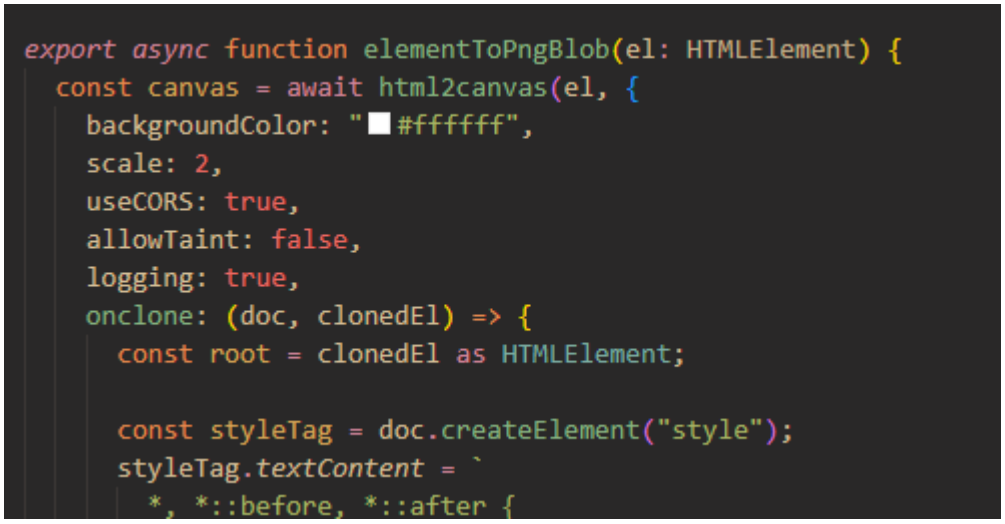
Key Features Implemented

- Expanded the marketplace so users could upload full canvas templates and individual card designs.
- Added a creator page for users to display their shared canvas and cards.
- Used *html2canvas* to generate preview images of canvas templates and cards.
- Saved marketplace preview image references to the database.

- Fixed marketplace image previews so templates and cards could be viewed more clearly.
- Prevented components from being dragged outside of the canvas area by adding a clamp
- Added z-index support so users could layer components correctly.
- Added modal components for improved user interactions.
- Added an MP3 audio bucket in Supabase for storing audio files.
- Fixed an issue where uploaded images were being cut off if they were not resized correctly.
- Added support for logging in using the Enter key.
- Edited the theme button and fixed the theme dropdown in the toolbox.
- Cleaned up the profile page.
- Improved the detailed home page.
- Fixed CSS issues across the app.
- Improved mobile view styling, especially on the home page.

Code

The marketplace preview system was generated using *html2canvas*. When a user shares a card or canvas, the application captures the selected HTML element and converts it into a PNG image blob. This blob is then uploaded to the *market-previews* Supabase Storage bucket. After upload, Supabase returns a public URL for the preview image, which is saved in the marketplace item record. This allows users browsing the marketplace to see a visual preview before deciding whether to use a template.



```
export async function elementToPngBlob(el: HTMLElement) {
  const canvas = await html2canvas(el, {
    backgroundColor: "#ffffff",
    scale: 2,
    useCORS: true,
    allowTaint: false,
    logging: true,
    onclone: (doc, clonedEl) => {
      const root = clonedEl as HTMLElement;

      const styleTag = doc.createElement("style");
      styleTag.textContent = `
        *, *::before, *::after {
```

Figure 74 Function to turn image data into a binary file object.

In the code, elementToPngBlob() handles the conversion from an HTML element to an image. It uses html2canvas with a higher scale value to improve image quality. During the capture process, styles such as animations, shadows, filters, videos, and iframes are disabled. This makes the preview more stable and avoids issues with unsupported or external embedded content. The resulting canvas is then converted into a PNG blob and uploaded using uploadMarketPreview().

A PNG blob is image data stored as a binary file object. In Orbit, this allowed uploaded or generated image content to be handled by the browser and then uploaded to Supabase Storage as a PNG file.

For a full canvas, the application captures the canvas element, uploads the preview image, and then inserts a marketplace record with kind: "canvas". For a single card, the application captures a hidden preview version of that card, uploads the preview image, and then inserts a marketplace record with kind: "node". This allows the same marketplace system to support both full canvas templates and smaller reusable card components.

```
// Load nodes from DB
useEffect(() => {
  if (!user) return;

  const load = async () => {
    setLoading(true);

    const { data, error } = await supabase
      .from("canvas_nodes")
      .select("id,user_id,type,x,y,w,h,z,props")
      .eq("user_id", user.id)
      .order("z", { ascending: true }); //z-index bug

    if (error) {
      console.error(error);
      setLoading(false);
      return;
    }
```

Figure 75 Nodes loading from DB changed to include z-index.

Z-index support was also implemented during this sprint. In the original node database structure, each component already had a z value, but it was not yet being used as a front-end feature. Nodes with smaller z values are rendered first, while nodes with larger z values are rendered later. In the browser, later elements appear above earlier

elements when they overlap. Before adding interface controls for this, the z value was manually changed in the database to confirm that the layering system worked.

```
const nextZ = nodes.length ? Math.max(...nodes.map((n) => n.z)) + 5 : 0; // needs increasing value
```

Figure 76 Nodes increase by 5 if brought forward.

New components were also given a higher z value by default so that they appear on top of existing components when added to the canvas.

```
const moveNodeZ = async (
  nodeId: string,
  direction: "backward" | "forward",
) => {
  if (!user) return;

  const previousNodes = [...nodes];
  const targetNode = nodes.find((n) => n.id === nodeId);
  if (!targetNode) return;

  const minZ = Math.min(...nodes.map((n) => n.z));
  const maxZ = Math.max(...nodes.map((n) => n.z));
```

Figure 77 Layer controls.

Layer controls were then added so users could move a selected component forward or backward. When this happens, the selected node is assigned a temporary higher or lower z value, and the full node list is resorted. These updated values are then saved to Supabase. This fixed an issue where layering changes visually worked during editing but reset after refresh because the original database z-index values were being reloaded. By persisting the updated z values, the canvas keeps the correct layering across sessions.

```
export function clamp(value: number, min: number, max: number) {
  return Math.max(min, Math.min(max, value));
}
```

Figure 78 Clamp function that limits a value so it cannot go below a min or above a max value.

```
const clampedX = Math.round(clamp(x, 0, canvasW - defaultW));
const clampedY = Math.round(clamp(y, 0, canvasH - safeH));
```

Figure 79 Restricts nodes x and y position so it cannot go out of the canvas.

A clamp function was used to keep canvas nodes within the canvas area. When a node is created or dragged, its x and y coordinates are checked against the canvas width and height. If the node would move outside the canvas, the value is limited to the nearest valid position. This prevents nodes from being lost off-screen and improves the usability of the canvas interaction system.

```
    updatedX = Math.round(clamp(n.x + delta.x, 0, canvasW - nodeW));
    updatedY = Math.round(clamp(n.y + delta.y, 0, canvasH - nodeH));

    return { ...n, x: updatedX, y: updatedY };
  }),
```

Figure 80 When a user drags a node, Orbit calculates the new position

This was also added to dnd-kit's drag end logic for existing components. *delta.x* and *delta.y* represent how far the user dragged the node. The code adds this movement to the current position and then clamps the result before saving it. This means nodes remain within the canvas even after being moved.

```
//receives the audio file URL through props, c
type Mp3PlayerProps = {
  url: string | null;
  title?: string | null;
  isOwner?: boolean;
  onReplace?: () => void;
  onRemove?: () => void;
  variant?: "default" | "topbar";
};
```

Figure 81 MP3 props

An MP3 player component was also added so users could attach music to their profile and make their digital space feel more personal. The component was designed to be reusable, allowing it to appear in both the canvas editing view and the public profile view.

The audio URL is used as the source for the track, while the title is displayed in the player interface. The isOwner prop controls whether the upload and delete buttons are

shown. This ensures that only the profile owner can replace or remove the track, while visitors are limited to playing or pausing it. Uploaded audio files are stored in Supabase Storage and linked to the user's profile data.

Other UI changes were also made during this sprint, including updates to themes, buttons, page layouts, and mobile styling. These changes are documented in more detail on Miro.

Technical Challenges and Solutions

The marketplace preview feature initially caused the error "Attempting to parse an unsupported colour function 'colour'." This happened because html2canvas could not interpret some of the modern CSS colour values used by the application themes. To solve this, the *elementToPngBlob()* helper was updated to clean the cloned HTML before the screenshot was generated. Then, inside the callback, unsupported *color()* values were detected and replaced with standard hex or RGBA colours. Gradients, masks, filters, animations, shadows, and iframes were also removed from the cloned preview. This made the generated marketplace previews more reliable and prevented the canvas image capture from failing.

Figure 82 Menu issues on nodes due to z-index being added.

Another challenge appeared after z-index layering was added. The side menu and contextual controls were sometimes hidden behind canvas components when the z-index was changed. This happened because the selected component and its menu rendered at the same time. As a result, when a node was moved forward, it could visually cover parts of the editing controls. This would be solved by separating the floating tray/menu from the normal node order and giving the interface controls their own higher layer. This would allow the menu to stay visible and usable even when canvas components overlapped.

Outcome

By the end of this sprint, the marketplace had become a more complete and usable feature. Users could share both full canvas templates and individual card designs, with preview images generated and stored through Supabase. This made the marketplace clearer and more useful because users could see what they were importing before choosing a template.

The canvas editing system was also improved through z-index layering and clamp boundaries. Users could move components forward or backward, and these layering changes were saved to the database, so they remained after refresh. The clamp function also made the canvas more reliable by preventing components from being dragged or dropped outside the visible workspace.

Further UI improvements were made with Orbit slowly coming more in line with Figma prototypes

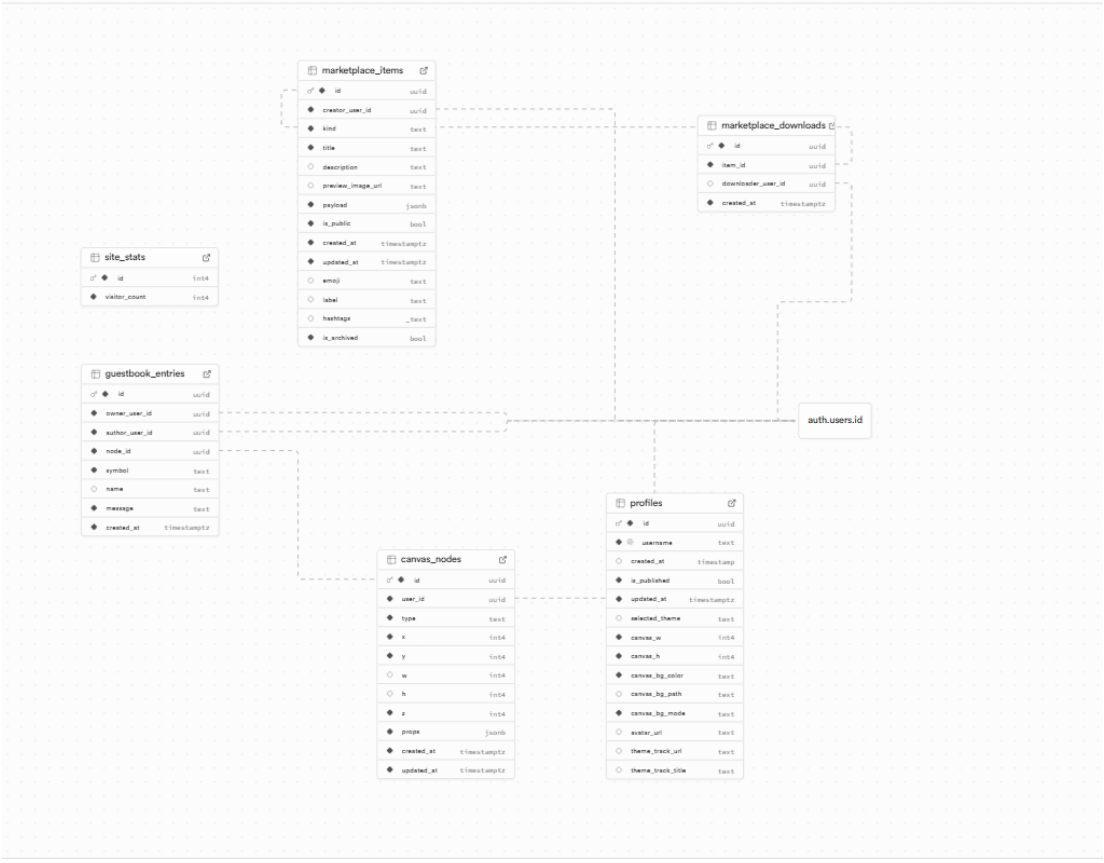


Figure 83 Database at the end of sprint 4.

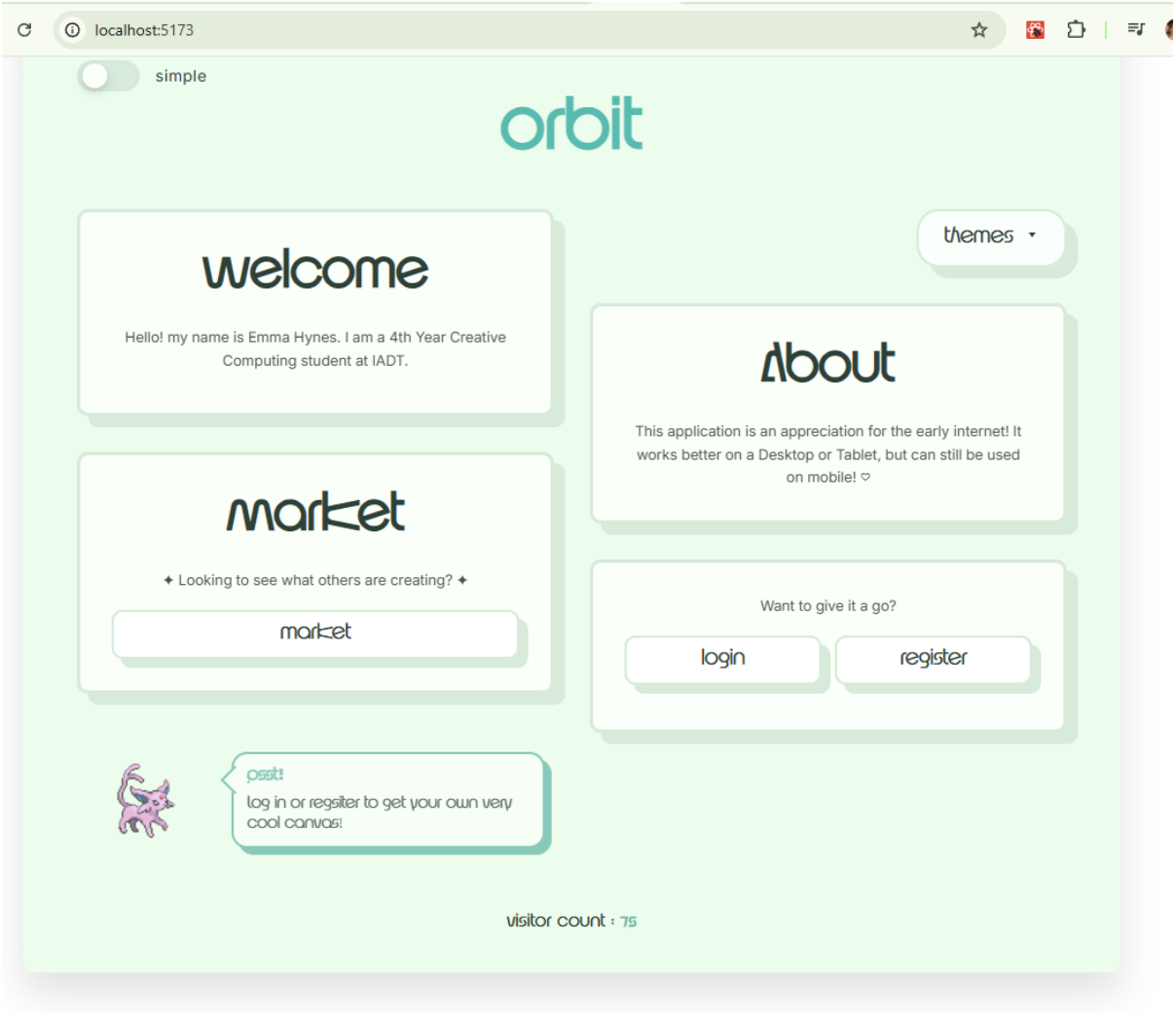


Figure 84 Home design at the end of sprint 4

Sprint 5: 20th March – 3rd April

Goals

The main goal of this sprint was to make the interface as functional and stable as possible before user testing. A key focus was reworking the node menu system. After z-index layering was added, the menu could sometimes appear underneath canvas components because it was being affected by the same stacking order as the nodes.

This sprint focused on separating the menu from the nodes, improving editing controls, and fixing issues with node deletion and editing.

Key Features Implemented

- Renamed the project/root directory from “Emma” to “Orbit”.
- Reworked the profile page structure and layout.
- Added a hover card on the public view page to display user bio and pronouns.
- Fixed editing functionality for image nodes.
- Fixed editing functionality for label nodes.
- Fixed editing functionality for button nodes.
- Tidied and refined the overall UI.
- Added hover behaviour to iframe based embeds.
- Improved the component button tray.
- Changed the component tray so it only appears when a component is selected.
- Updated the component tray so it opens downward.
- Added a background to the component tray for better visibility.
- Made the component tray flip sides depending on the node’s position on the canvas.
- Changed rendering so the component tray always appears above other canvas elements.

Code

Figure 85 Separate rendering for contextual menus when nodes load in.

Figure 86 FloatingTray and how it renders.

The contextual menu was changed to render separately from the canvas nodes. Instead of placing the controls inside each DraggableNode, the canvas first renders all nodes from the nodes array, then renders FloatingTrayOverlay as a separate overlay inside the canvas surface.

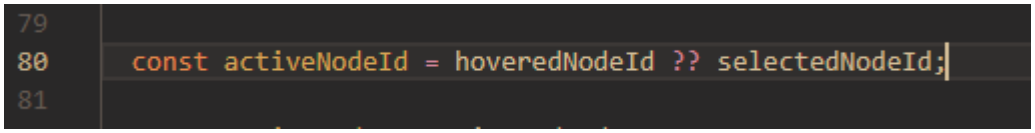


Figure 87 How a node is deemed active

The currently selected or hovered node is calculated using selectedNodeId or hoveredNodeId and passed into the overlay as activeNode. The overlay then uses the active node's x and y position, width, and canvas size to position itself beside the selected component.

This ensured the menu remained visible above the selected node, even when canvas components overlapped.

Other UI changes were made to the tray, including clearer iconography, a stronger background, and a downward-opening layout. These changes made the editing controls easier to understand and use.

For iframe-based embeds, the interaction behaviour was adjusted. Unlike normal nodes, iframe nodes could be selected through hover. This was added because clicking directly on an iframe can trigger the embedded content, such as playing a video or interacting with a music embed. Hover selection made it easier to access the tray controls without accidentally starting media.

```
const nodeWidth = node.w ?? 260;
const nodeCenterX = node.x + nodeWidth / 2;
const flipLeft = nodeCenterX > canvasW / 2; // menu position based on the nodes position and width

const trayLeft = flipLeft ? node.x - 64 : node.x + nodeWidth + 10; //if node is on the right side of the canvas, flipleft becomes true
```

Figure 88 Flips FloatingTray left or right based on position.

The tray flip behaviour works by checking whether the selected node is on the left or right side of the canvas. First, the tray gets the node width, defaulting to 260px if no saved width exists. It then calculates the centre point of the node. If the node's centre point is greater than half of the canvas width, the tray flips to the left side of the node. For example, if the canvas is 1800px wide, the halfway point is 900px. If the selected node's centre is greater than 900px, flipLeft becomes true, and the tray appears on the left. This prevents the tray from going off-screen or becoming awkward to use.

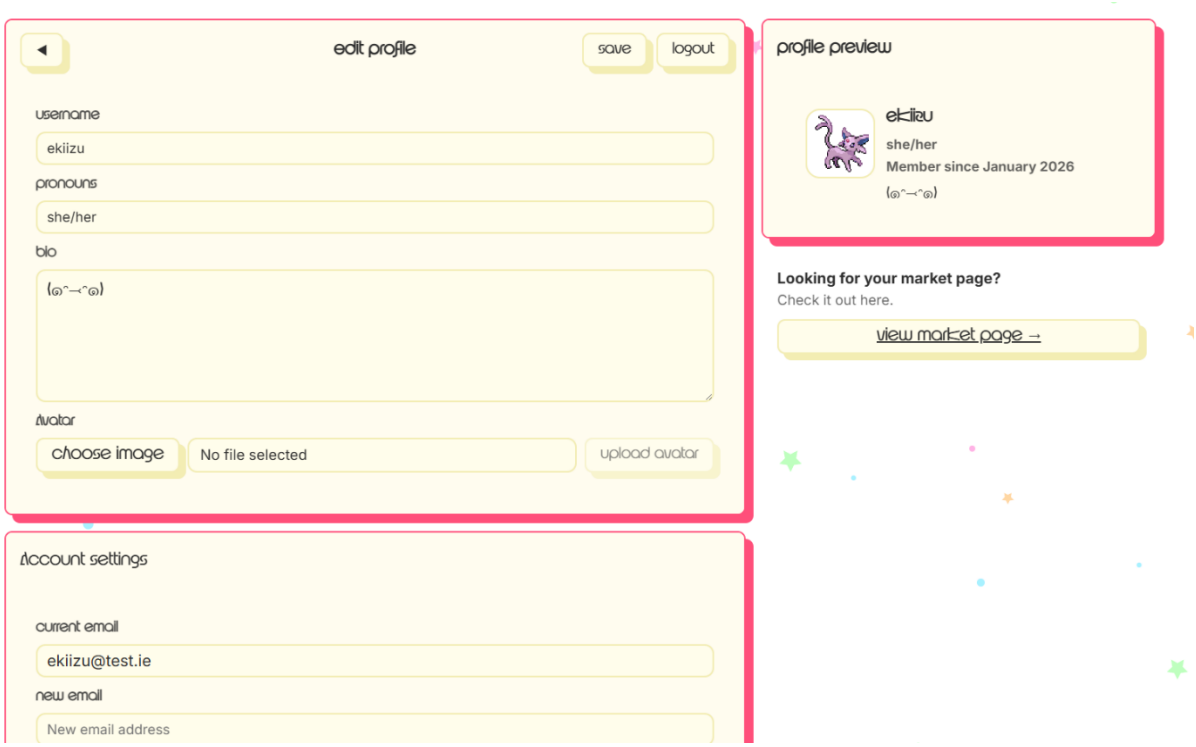


Figure 89 Profile Edit UI update

Other UI changes were also made at this stage such as improving the look of the profile edit

Technical Challenges and Solutions

There were no major challenges during this sprint, as many changes were UI based.

Outcome

By the end of this sprint, the interface was more stable, clearer, and ready for user testing. The component tray worked more reliably, remained visible above canvas elements, and adapted its position depending on where the selected node was placed. Editing issues with image, label, and button nodes were fixed, and iframe embeds became easier to manage.

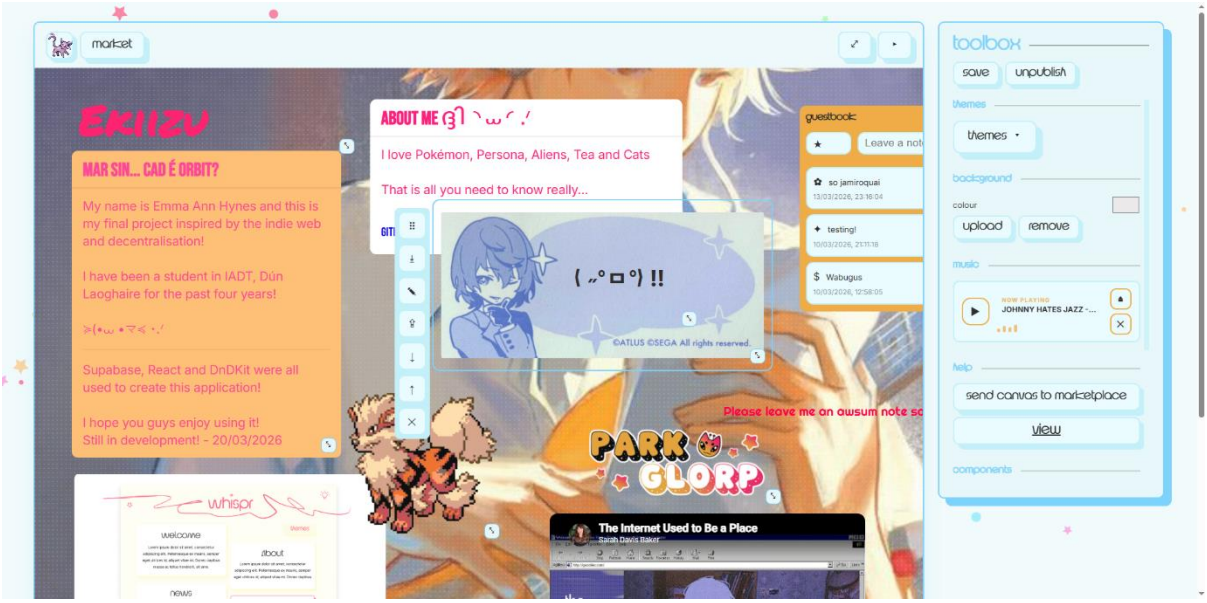


Figure 90 Canvas at the end of sprint 5.

Sprint 6: 3rd April – 17th April

Goals

The main goal of this sprint was to gain feedback from user testing and implement the feedback.

Key Features and Implementation

- User testing and used feedback to guide UI changes
- Added animations using Motion to improve interface feedback and transitions.
- Split the large Canvas file into smaller files/components to improve code organisation and maintainability.
- Fixed TypeScript errors that were causing deployment issues.
- Fixed canvas height issues.
- Added a new sticker node type.
- Added a delete button for the canvas.
- Fixed a z-index bug where component layering reset after refresh.

- Updated z-index persistence so layering changes were saved correctly instead of reverting to the original database value.
- Changed and refined the visual themes.
- Added a new background style.
- Removed the star background from selected areas/themes.
- Improved marketplace readability.
- Fixed marketplace mobile view issues.
- Fixed general mobile view issues across the application.
- Updated the toolbox layout based on user testing feedback.
- Improved the contextual/component menu based on user testing feedback.
- Moved drag and resize controls based on user testing feedback.
- Made the component controls move to the top or bottom depending on canvas height/position.
- Added delete functionality for guestbook entries.

Code

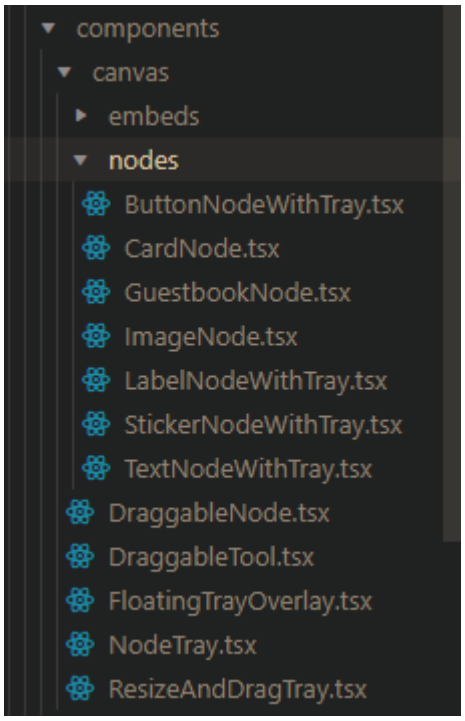


Figure 91 Updated organisation.

A major code focus of this sprint was improving maintainability. The original Canvas file had become too large because it contained canvas rendering, node movement, toolbox logic, uploads, marketplace actions, and editing controls in one place. To make the

code easier to manage, parts of this logic were separated into smaller components such as the draggable node component, floating tray overlay, resize and drag tray, and tool preview components. This made the codebase easier to understand and reduced the risk of breaking unrelated features when making changes.

```
function AnimatedRoutes() {
  const location = useLocation();

  return (
    <AnimatePresence mode="wait">
      <motion.div
        key={location.pathname}
        className="route-transition"
        initial={{ opacity: 0, }}
        animate={{ opacity: 1, }}
        exit={{ opacity: 0, }}
        transition={{ duration: 0.22, ease: "easeOut" }}
      >
        <Routes location={location}>
          <Route path="/" element={<Home />} />
```

Figure 92 Function for motion animation.

Motion was used to add small animations and improve user feedback. These animations helped the interface feel more responsive and made transitions between the pages feel clearer without changing the core functionality of the app.

Figure 93 Updated UI for contextual menu.

A further improvement was the contextual sidebar, canvas colours, theme colours. The aim was to make the interface feel more visually consistent with the selected theme while also improving the overall vibe of the application.

The background system was also changed. Originally, the application used a TypeScript based star background, but it was changed due to some visual inconsistencies and loading issues. ChatGPT was used to help generate and refine a reusable gradient background component based on the grainient effect on reactbits. This can be seen in AI logs in Miro.

The final AnimatedBg component reads the current CSS theme variables and uses them to create a layered radial gradient. This meant the background could automatically adapt to each selected theme rather than being hardcoded as one fixed design.

```
  const obs = new MutationObserver(update); //detects when theme changes by watching HTML element
  obs.observe(document.documentElement, {
    attributes: true,
    attributeFilter: ["data-theme"],
  });

  return () => obs.disconnect();
```

Figure 94 MutationObserver which detects when a theme changes.

ChatGPT was useful in helping structure this component, particularly with converting theme colours into transparent RGBA values and using a MutationObserver to detect when the theme changed. The code was then adapted to fit Orbit's existing theme system and styling. This improved the visual consistency of the application and made the background feel more intentional than the original star background.

Technical Challenges and Solutions

Previously, layering changes could appear to work visually during the editing session but reset after the page was refreshed. This happened because the updated z-index values were not being saved consistently to Supabase. The solution was to update the selected node's z value, reorder the full node list, normalise the z-index values, and save the updated values back to the database. This meant that when the canvas was reloaded, the node layers appeared in the correct order.

Outcome

By the end of this sprint, the contextual sidebar, canvas colours, and theme backgrounds had been improved. The new animated background helped make each theme feel more complete and visually distinct, while still using the same reusable CSS variables as the rest of the interface. Overall, at this point Orbit had all features working as intended.

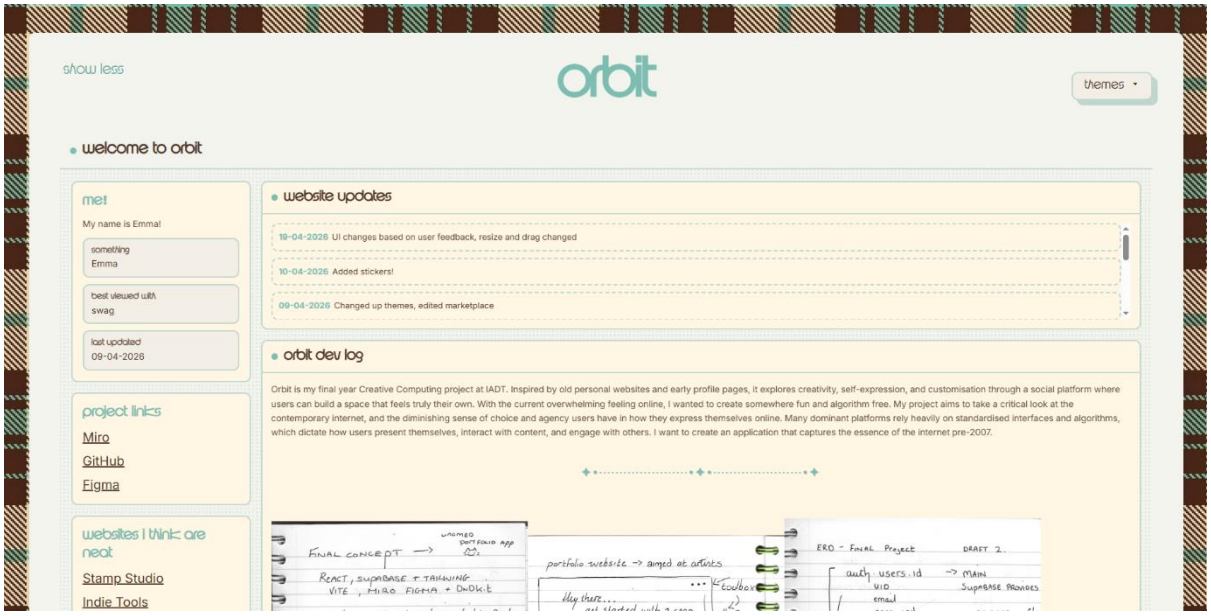


Figure 95 Detailed Home

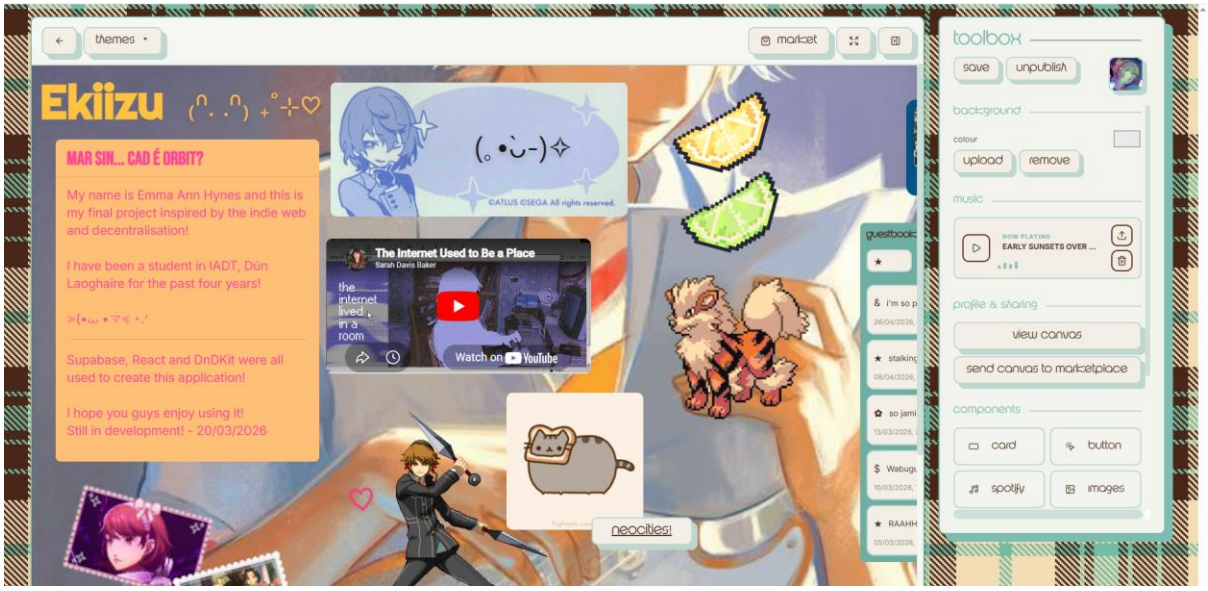


Figure 96 Canvas as of sprint 6

Sprint 7: 17th April – 1st May

Goals

The goal of this sprint was to ensure that the application was working as intended, carry out light performance testing, add comments, tidy the code, and make final UI improvements before submission.

Key Features and Implementation

- Added custom validation messages for clearer user feedback.
- Refined themes so they matched more consistently across the full application.
- Added scroll behaviour inside card nodes so longer content could be viewed properly.
- Added a visual drag-onto-canvas interaction.
- Removed click-to-add behaviour for components, so users now drag components onto the canvas instead.
- Added navigation from the marketplace to the creator page.
- Added delete functionality for guestbook entries.
- Separated resize and drag controls into a dedicated tray.
- Refined typography across the interface.
- Changed the toolbox layout.
- Added code comments for clarity and maintainability.
- Added rotation in the database for components and implemented functionality

Code

```
const deleteEntry = async (entryId: string, authorUserId: string) => {
  if (!meUserId) return;

  const canDelete = meUserId === authorUserId || meUserId === node.user_id;
  if (!canDelete) return;

  const { error } = await supabase
    .from("guestbook_entries")
    .delete()
    .eq("id", entryId);

  if (error) {
    console.error(error);
    return;
  }

  await loadEntries();
};
```

Figure 97 Guestbook entry delete

Guestbook deletion was implemented inside the GuestbookNode component. Each guestbook entry is stored in the guestbook_entries Supabase table with a unique ID and an author_user_id. When the delete button is clicked, the application checks whether the logged-in user is either the author of the entry or the owner of the guestbook node. If the user has permission, Supabase deletes the matching row using the entry ID. The guestbook entries are then reloaded so the deleted message is removed from the interface. This was implemented so users could remove unwanted or inappropriate messages from their profile.

SQL Editor

```
1 alter table canvas_nodes
2 add column if not exists rotation numeric default 0;
```

Figure 98 SQL to edit canvas_nodes table and add rotation.

```
                }}
onRotateLeft={() => {
  if (activeNode) {
    updateNode(activeNode.id, {
      rotation: (activeNode.rotation ?? 0) - 15,
    });
  }
}}
onRotateRight={() => {
  if (activeNode) {
    updateNode(activeNode.id, {
      rotation: (activeNode.rotation ?? 0) + 15,
    });
  }
}}
onRotateReset={() => {
  if (activeNode) {
    updateNode(activeNode.id, {
      rotation: 0,
    });
  }
}}
//contextual menu
```

Figure 99 Contextual menu buttons for rotation

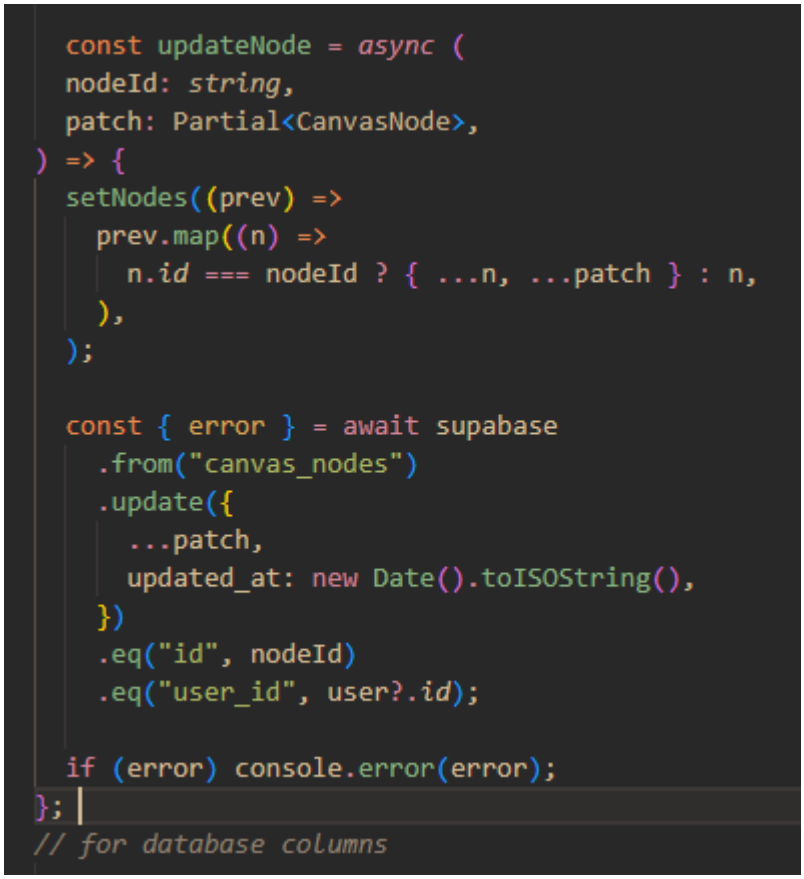
Rotation was implemented in a similar way to the layering system. A rotation field was added to the *canvas_nodes* table so that each component could store its own rotation value. Rotation functions were then passed into the FloatingTrayOverlay. This means that when a user clicks the rotate buttons in the contextual menu, the overlay calls the relevant rotation function.

```
const activeNodeId = hoveredNodeId ?? selectedNodeId;

const activeNode = activeNodeId
  ? (nodes.find((n) => n.id === activeNodeId) ?? null)
  : null;
```

Figure 100 Ensures the correct component is rotated.

The selected node is stored as *activeNode*, which is calculated using the currently selected or hovered node ID. This ensures that the correct component is updated when the user rotates it.



```
  const updateNode = async (
  nodeId: string,
  patch: Partial<CanvasNode>,
) => {
  setNodes((prev) =>
    prev.map((n) =>
      n.id === nodeId ? { ...n, ...patch } : n,
    ),
  );

  const { error } = await supabase
    .from("canvas_nodes")
    .update({
      ...patch,
      updated_at: new Date().toISOString(),
    })
    .eq("id", nodeId)
    .eq("user_id", user?.id);

  if (error) console.error(error);
};
// for database columns
```

Figure 101 Node updates in database.

When the user clicks rotate left, the application takes the current rotation value and subtracts 15 degrees. When the user clicks rotate right, it adds 15 degrees. The updated rotation value is then passed to *updateNode()*, which updates the local state and saves the new value to Supabase. This means the rotation appears immediately on the canvas and remains saved after refresh.

```
const signUp = async (email: string, password: string, username: string) => {
  const cleanUsername = username.trim().toLowerCase();
  const cleanEmail = email.trim().toLowerCase();

  const { data: existingUser, error: checkError } = await supabase
    .from("profiles")
    .select("id")
    .eq("username", cleanUsername)
    .maybeSingle();

  if (checkError) {
    return { error: { message: "Could not verify username. Please try again!" } };
  }

  if (existingUser) {
    return { error: { message: "That username is already taken." } };
  }

  const { data, error } = await supabase.auth.signUp({
    email: cleanEmail,
    password,
    options: { data: { username: cleanUsername } },
  });

  if (error) {
    if (error.message.toLowerCase().includes("user already registered")) {
      return { error: { message: "That email is already in use. Please use something else!" } };
    }
```

Figure 102 Updated error messages for registering.

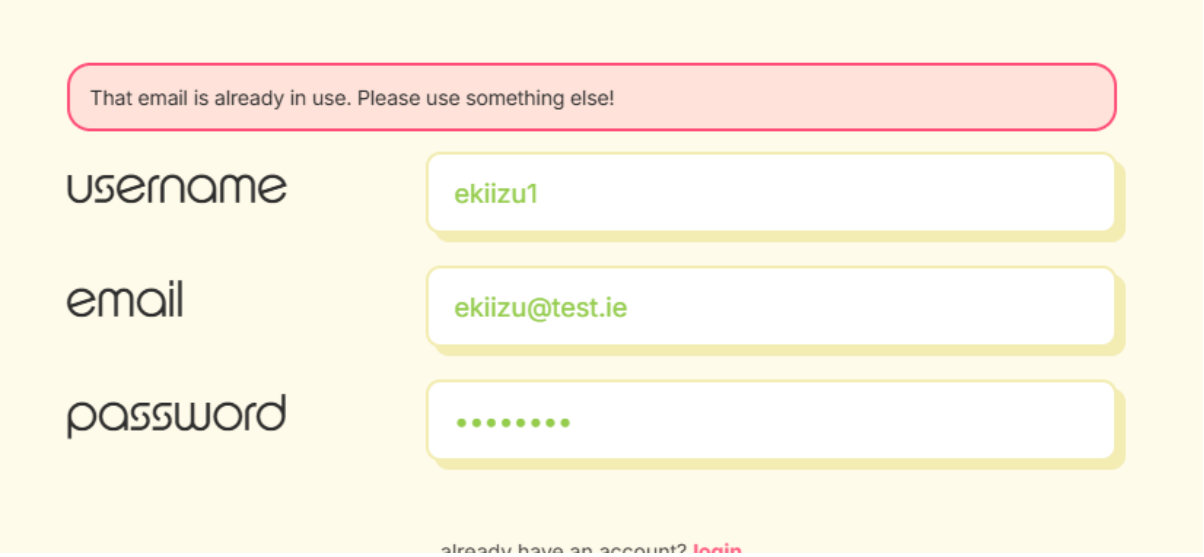


Figure 103 Updated error messages in use.

Due to some user feedback error messages and validations errors were improved throughout the application also.

Technical Challenges and Solutions

One challenge with implementing rotation was that older nodes did not originally have a rotation value. To prevent these older nodes from breaking the application, the rotation logic uses a default value of 0 when no rotation exists. This means every node can safely be rotated, even if it was created before the rotation feature was added.

Another challenge was ensuring that drag, resize, layer, and rotate controls did not interfere with each other. This was improved by separating the resize and drag controls into their own dedicated tray like splitting up the menu originally while keeping rotation and other actions in the contextual overlay.

Outcome

By the end of this sprint, Orbit had reached a complete and presentable state for submission. The final sprint focused on polishing the application, improving user feedback, refining the interface, adding code comments, and fixing remaining issues.

Final testing was carried out across the application, including authentication, canvas saving, node editing, resizing, deleting, rotation, marketplace uploads, responsive behaviour, and performance. Bugs found during testing were fixed where possible, and any remaining limitations were documented for the future.

5.2 Development Environment

The application was developed using Visual Studio Code. GitHub was used for version control, while Supabase was used for authentication, database storage and file storage. The project was deployed and tested using Render.

Figma and Figma Make were used during the design and prototyping stage to explore layouts, themes and interaction ideas. Miro was used to document research, planning and sprint progress.

AI assistance was used as a development aid for certain sections and to support debugging, but all code that was generated was reviewed, tested and adapted to fit the project. It was also used to clean up CSS files at the end of development.

This can be seen in the AI log sections on Miro.

5.3 Conclusion

Overall, the implementation stage transformed Orbit from a design concept into a functional web application. The development process and sprints with different thoughts and comments can be seen in more detail on Miro.

Development progress was recorded in a YouTube Playlist which is linked in Appendix E.

6. Testing and Evaluation:

6.1 Test Plan

This chapter evaluates the final system through a combination of functional testing, usability testing and performance testing. A test plan was used to assess whether the core features of the application operated as intended, while user feedback and responsive testing were used to evaluate the overall usability of the system across desktop and mobile devices.

6.2 Functional and Feature Testing

This section presents the functional and non-functional tests carried out on the application. Functional testing was used to check whether the main features worked as expected, while non-functional testing evaluated usability, responsiveness, security, performance and persistence.

6.2.1 Functional Requirements

#	Description	Input	Expected Output	Actual Output
1	The system shall allow users to register, log in, and log out securely.	User enters valid email, password and username, then logs in/out.	Account is created, user can log in successfully, and logout returns user to login page.	Passed
2	The system shall automatically create a profile record for each newly registered user.	New user registers through the registration form.	A matching record is created in the profiles table using the Supabase auth user ID.	Passed
3	The system shall require each user to have a unique username to support public profile routing.	User attempts to register with an existing username.	Registration is blocked and a message such as “That username is already taken” is shown.	Passed
4	The system shall allow visitors to view a user’s published public canvas through a username-based URL.	Visitor opens /profile/username for a published user.	The correct public canvas loads using the username route.	Passed
5	The system shall prevent unpublished canvases from being publicly viewable.	Visitor opens a username-based URL for an unpublished canvas.	The canvas is not displayed, or the visitor is shown an /not published message.	Passed

6	The system shall allow authenticated users to create and edit a personal canvas, with all changes saved to the database.	Logged-in user edits canvas content and refreshes the page.	Changes persist after refresh and are loaded from Supabase.	Partially Passed. Refresh Issues
7	The system shall allow users to add, move, resize, layer, and delete canvas nodes.	User drags a component onto the canvas, moves it, resizes it, changes layer order, then deletes it.	Node is created, updated, layered correctly and removed from the canvas/database when deleted.	Passed
8	The system shall support multiple node types including text, cards, images, GIFs, Spotify embeds, YouTube embeds, stickers, labels, and guestbook components.	User adds each available node type from the toolbox.	Each node type appears correctly and stores its relevant data.	Passed
9	The system shall allow users to select and apply themes to their profile and canvas.	User selects a theme from the theme dropdown.	Theme updates immediately and persists after refresh	Partially Passed- Refresh Issues
10	Allowing users to upload a profile audio track	Users upload a selected MP3 track.	Song uploads and can be played/paused	Passed
11	The system shall allow users to submit canvas content to the marketplace.	User submits a card or canvas to the marketplace.	Marketplace item is created successfully.	Passed
12	The system shall allow users to browse shared marketplace content.	User opens the marketplace page.	Shared marketplace content is displayed.	Passed

13	The system shall allow users to leave a guestbook message on another user's profile.	Logged-in user submits a guestbook message.	Message is saved and displayed on the canvas.	Passed
14	The system shall support z-index ordering so nodes can appear above or below one another.	User brings a node forward or sends it backward using the contextual menu	Node layer order changes and persists after refresh.	Partially Passed – Refresh Issues
15	The system shall prevent draggable components from being moved outside the permitted canvas area.	User drags or resizes a component beyond the canvas boundary.	Component remains within the canvas area.	Passed
16	The system shall adapt toolbox and interface controls based on screen size and canvas position.	User views app on different screen sizes or selects nodes near canvas edges.	Toolbox and controls remain accessible and usable.	Failed
17	Providing validation and user feedback	User submits empty fields, invalid links, or duplicate usernames.	Error messages are shown and invalid data is not saved.	Passed

6.2.2 Non-Functional Requirements

#	Description	Input	Expected Output	Actual Output
1	The system shall provide an interface that is intuitive enough for users to create and customise a canvas without needing coding knowledge.	A user attempts to create and customise a canvas.	User can add, edit and arrange components without needing code or technical knowledge.	Passed

2	The system shall allow first-time users to understand the main editing interactions, such as dragging, resizing, and editing nodes, with minimal instruction.	First-time user interacts with the canvas	User can understand the basic editing workflow after brief use.	Passed
3	The system shall support personalisation and creative freedom without requiring advanced skills.	User change's themes, adds media, edits cards and arranges nodes.	User can create a visually personalised canvas using built in controls.	Passed
4	The system shall respond smoothly to drag-and-drop interactions and render canvas updates without noticeable lag during normal use.	User drags, resizes or edits nodes on the canvas.	Interactions feel responsive and updates appear quickly during normal use.	Partially Passed – Slow image loading
5	Published canvases shall be viewable by visitors without requiring login.	Visitor opens a published username-based profile URL.	Published canvas loads without requiring authentication.	Passed
6	The system shall adapt appropriately to different screen sizes, including mobile layouts.	User opens the application on desktop, tablet or mobile.	Layout adjusts appropriately and key controls remain usable.	Failed

6.2.3 Analysis of Results

Functional testing showed that the core features worked as intended. Users could register, log in, create and edit canvas nodes, upload media, change themes, publish profiles, and view public canvases. The main success was persistence, as nodes, themes, backgrounds, and profile data were restored correctly after refresh or login.

Testing also confirmed that the node system supported multiple content types, including cards, images, GIFs, embeds, labels, stickers, and guestbook components. Z-index layering and clamping worked correctly, allowing nodes to be layered while preventing them from moving outside the canvas. Marketplace features also worked as intended and users were able to update their user information easily.

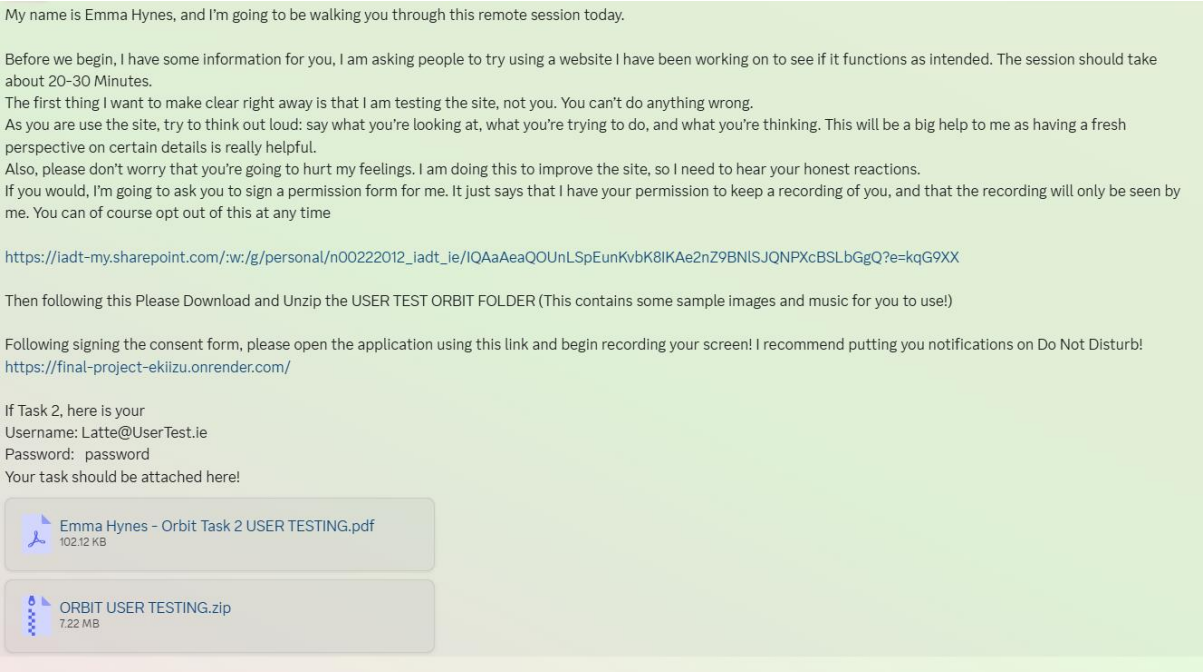
Non-functional testing showed that Orbit was usable and responsive on desktop. However, mobile canvas editing was more limited because of smaller screen space and less precise touch interaction. Overall, the application met most functional and non-functional requirements, with mobile editing remaining the main limitation.

6.3 Usability and User Testing

The purpose of usability testing is to identify any issues within the design. This was a key area for the project. User testing was carried out using two tasks designed around the core features of the website.

Six participants took part in the testing process remotely. The participants were split into two groups, with three users completing each task. This allowed feedback to be gathered on two key areas of the system without making the testing session too long.

Participants were sent a message containing all information they needed including a consent form, a link to the website and a sample folder containing sample images. They were asked to follow the task they were given.


My name is Emma Hynes, and I'm going to be walking you through this remote session today.

Before we begin, I have some information for you, I am asking people to try using a website I have been working on to see if it functions as intended. The session should take about 20-30 Minutes.
The first thing I want to make clear right away is that I am testing the site, not you. You can't do anything wrong.
As you are use the site, try to think out loud: say what you're looking at, what you're trying to do, and what you're thinking. This will be a big help to me as having a fresh perspective on certain details is really helpful.
Also, please don't worry that you're going to hurt my feelings. I am doing this to improve the site, so I need to hear your honest reactions.
If you would, I'm going to ask you to sign a permission form for me. It just says that I have your permission to keep a recording of you, and that the recording will only be seen by me. You can of course opt out of this at any time

https://iadt-my.sharepoint.com/:w:/g/personal/n00222012_iadt_ie/IQAaAeaQOUnLSpEunKvbK8IKAe2nZ9BNlSJQNPXcBSLbGgQ?e=kqG9XX

Then following this Please Download and Unzip the USER TEST ORBIT FOLDER (This contains some sample images and music for you to use!)

Following signing the consent form, please open the application using this link and begin recording your screen! I recommend putting you notifications on Do Not Disturb!
https://final-project-ekiizu.onrender.com/

If Task 2, here is your
Username: Latte@UserTest.ie
Password: password
Your task should be attached here!

Emma Hynes - Orbit Task 2 USER TESTING.pdf
102.12 KB

ORBIT USER TESTING.zip
7.22 MB

Figure 104 Message sent out to participants.

6.3.1 User Tasks

Task 1 – Registration and Initial Canvas Creation

Participants were asked to create an account, access their personal canvas, add components and begin customising the canvas. This task was used to test whether the onboarding process was clear and whether first-time users could understand how to start building their canvas.

Participant	Outcome	Notes
USER #1	Completed	Registered, customised canvas successfully. Completed Task
USER #2	Completed with minor difficulty	Struggled with getting image nodes to work correctly, did not know it was a drag. Completed Task
USER #3	Completed	Registered, customised the canvas successfully. Mentioned difficult to read font. Completed Task

Task 2 – Login and Market Upload

Participants were asked to log in to an existing account, interact with their canvas and upload content to the marketplace. This tested whether returning users could access their account, understand the editing tools and complete a sharing action.

Participant	Outcome	Notes
USER #4	Completed	Logged in and uploaded to the marketplace successfully. Completed Task
USER #5	Completed	Logged in, commented on nice reactivity and colours. Completed Task
USER #6	Completed with some difficulty	Logged in, uploaded to marketplace after struggling with canvas having no progress indicators. Picked it up

		gradually, however. Completed Task

6.3.2 User Feedback

The survey results showed that users responded positively to the creative direction of Orbit, particularly the freedom to customise a personal canvas. However, the feedback also highlighted areas for improvement, including clearer drag-and-drop guidance for components, more consistent UI styling, better font readability and progress indicators when images or marketplace uploads are loading.

These findings were used to inform the finishing touches and features that could be added in the future.

The full survey responses are included in Appendix A.

6.4 Performance and Responsive Testing

Performance and responsive testing were carried out to check that Orbit worked reliably across different screen sizes and performed efficiently in the browser. Google Lighthouse was used to test the deployed website and review areas such as performance, accessibility, best practices and SEO.

The Lighthouse results aimed to help identify issues that could affect the user experience, including loading speed, image optimisation, accessibility warnings and layout behaviour on smaller screens.

Responsive testing was also carried out manually using browser developer tools to check how key pages, such as the canvas editor, profile page and marketplace, behaved on different viewport sizes.

Performance

Test Area	Input	Result
Page load speed	Open the hosted app and inspect loading in Chrome DevTools	Loaded successfully

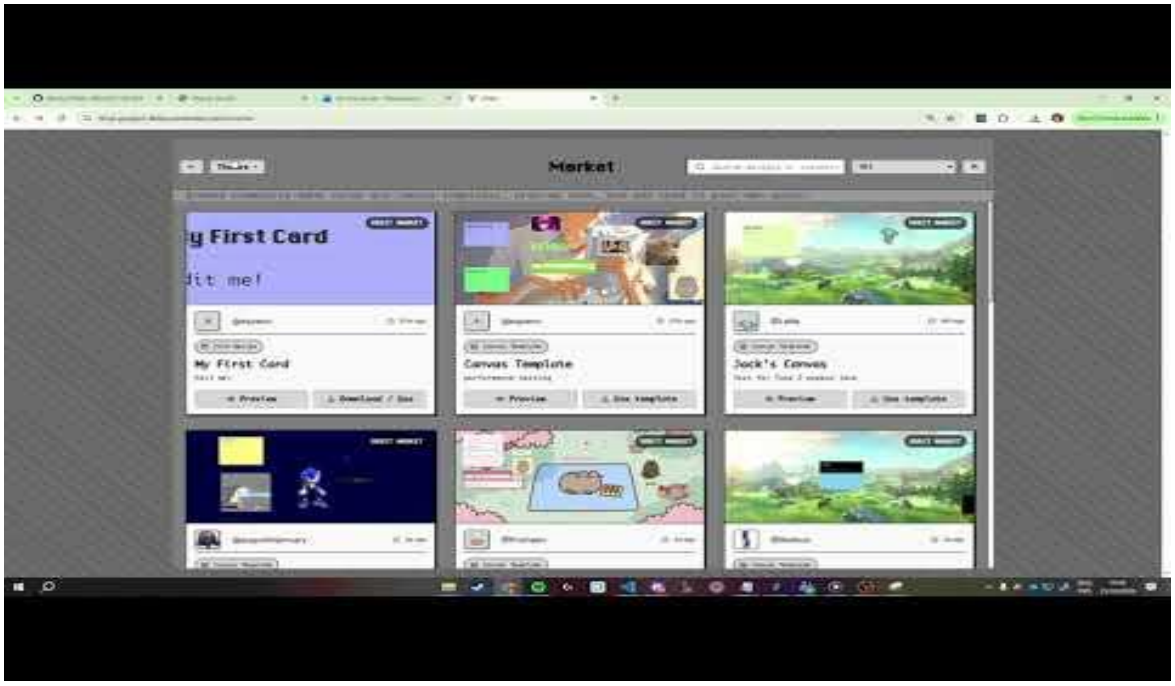
Canvas loading	Log in and load a saved canvas with multiple nodes	Saved canvas restored correctly
Drag-and-drop	Drag nodes around the canvas repeatedly	Nodes move smoothly on desktop
Node creation	Drag several components onto the canvas	Components added successfully
Resize performance	Resize card multiple times	Resize worked correctly
Media loading	Load images, gifs, Spotify/Youtube	Media loaded correctly
Marketplace previews	Send a card/canvas to marketplace	Preview generated successfully
Mobile perfomance	Test canvas on mobile phone	Responsive, poor editing experience

Response

Page tested	Requests	Transferred	DomContent	Load	Notes
Home Page	11	2.8kb	138ms	221ms	Excellent
Canvas	76	185 kb	636 ms	656 ms	Good, Under 1 second
Marketplace	50	2.1 mb	330 ms	415 ms	Good – Larger Transfer size

The results suggest that the application loads efficiently during normal use, although future optimisation could focus on reducing marketplace image sizes and lazy loading media content.

https://www.youtube.com/watch?v=FnwwYre5ptQ



Further performance test results can be seen in Appendix B.

6.5 Error Handling and Debugging

During user feedback, participants identified areas where Orbit needed clearer communication and clearer error handling. Although Supabase provides built-in error responses for authentication, these were revisited after user testing and adapted to better suit Orbit's interface and user experience.

Some users were unsure whether actions such as saving, uploading, or submitting forms had worked successfully. In response, clearer feedback messages, validation checks, and loading states were added to make the system easier to understand. For example, registration was improved so users receive a clear message when a username is already taken or on the marketplace users now must be logged in to use the download button. Key actions, such as importing templates, also uses disabled button states while loading to prevent duplicate submissions.

Debugging was carried out using browser developer tools, console logs, Supabase error messages, and repeated manual testing. This helped identify issues with the authentication, uploads, marketplace actions, and canvas updates.

7. Project Management

7.1 Agile Methodology and Sprint Structure

The project followed an Agile approach, with development divided into short 2-week sprints. This allowed the project to be planned, tested, reviewed, and adapted throughout the semester. Each sprint focused on a specific stage of the project, from research and design to implementation, testing, refinement, and final presentation.

Sprint	Dates	Focus
Sprint 1	23rd Jan – 6th Feb	Figma, research, early design
Sprint 2	6th Feb – 20th Feb	Back-end and setup
Sprint 3	20th Feb – 6th March	React Development
Sprint 4	6th Mar –20th Mar	React Development and UI
Sprint 5	20th Mar – 3rd April	Development
Sprint 6	3rd April – 17th April	User testing, report, development
Sprint 7	17th April – 1st May	Report, UI and final polish
Sprint 8	1st May – 6th May	Presentation

Miro and a notebook were also used to track design decisions, sprint progress, technical issues, and feedback throughout development.

7.2 Tools for Project Planning and Management

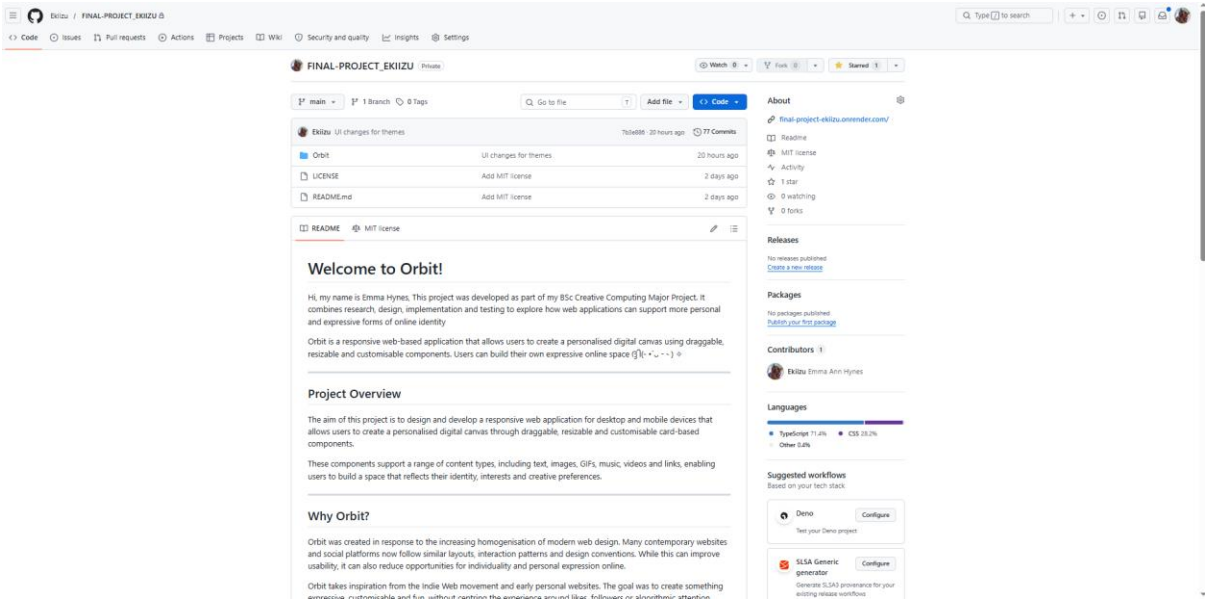


Figure 105 GitHub connected to project

GitHub was used to store and manage the project code throughout development. It allowed changes to be tracked using commits and provided a backup of the project repository. GitHub was also used as the final submission location for the application code.

Figure 106 Miro for planning and management

Miro was used for visual planning and project documentation. It helped organise the research, design ideas, user flows, wireframes, sprint planning, and screenshots of development progress. This made it easier to track and see how the project changed over time.

A physical notebook was used for planning, sketching ideas, and recording thoughts during development. This can be seen on Miro.

8. Conclusion and Future Work:

8.1 Summary of Outcomes

Overall, I believe this project successfully did what I created it to do. Instead of prioritising scrolling, recommendations, or addictive interaction patterns, Orbit focuses on slower and more intentional creation. Users are encouraged to make, arrange, customise, and share a small space that feels personal to them. By taking strong inspiration from the early internet this project aimed to demonstrate that websites can be personal and visually distinctive if designed thoughtfully and intentionally from the start.

The final working application works as a digital scrapbook or pinboard, allowing users to freely add and arrange content such as cards, images, links, GIFs and stickers. a small guestbook to allow users who come and go to leave a message was also added directly taking inspiration from the old internet. A major outcome of this project was my improved understanding of the effort that goes into developing a functional application of this scale from start to finish. It was difficult to keep everything organised and stay on track at points, but I have gained a good insight into the development cycle and will carry this onwards with future projects.

From a technical perspective, I developed a full stack application using React, TypeScript, CSS and SQL with Supabase. The final application has features such as user authentication, full customizable canvases, customizable card components, image uploading, persistent theme customization, a community-based marketplace feature and persistent storage through Supabase. These features all align with my initial plans for the functional requirements of Orbit.

The testing that was done was also key to improving the application and helped me understand just how important user feedback was when designing something with such a strong emphasis on an enjoyable and strong user experience.

Overall, I am extremely satisfied with the outcome of Orbit. I feel that this project allowed me to demonstrate the skills I gained while in IADT and allowed me to show of the passion I have for UI and web design. It has strengthened my confidence in designing and developing full stack applications even further in the future.

8.2 Critical reflection on Process and Learning

Orbit began due to my interest in the lack of individuality on the current internet and my disinterest in the current state of the internet and websites I use every day. Being in a computing course that often asked me to use systems such as bootstrap and other UI Frameworks due to it being the industry standard pushed me to try to make something myself with just CSS even if it was harder and took longer to design. I'd always felt restricted in that and while of course functionality is priority, design was never as much of a focus.

Coupled with this, as mentioned in my research, many websites lack customisation and the ones that do require people to have coding knowledge. Following the release of the iPhone we have drifted further away from making websites that are visually distinctive as they need to operate on mobile phones and with the current rise of AI also likely to begin influencing how new websites are designed, I became interested in questioning what the future of online spaces might look like and whether users will have even fewer opportunities to shape their own digital environments.

I never intended to create a website that rejected usability or attempted to redesign the entire system. Instead, Orbit was designed to explore how expressive design could exist alongside good UI practice. The application still uses clear feedback messages, loading states, validation, consistent buttons, readable layouts, and responsive design. The aim was not to make a confusing or overly experimental interface, but to show that a website can be functional and usable while still allowing more room for personality and creativity.

I learnt throughout making this application that it is incredibly hard to make something that will suit everyone. That there is a reason why websites have become so similar to one another. It was difficult to design an interface that allowed users to place things freely, as ensuring that the UI such as the contextual menus all functioned similarly with each component despite being so different put into perspective the effort required to make something like this.

Making Orbit allowed me to explore a topic and area I was genuinely passionate about while also improving my technical understanding and putting me in the shoes of developers and users alike.

8.3 Limitations and Future Improvements

Although Orbit successfully met its core aims, it still has limitations and areas that I could improve on with further development.

One limitation is that users only have one canvas they can customise. I would like to allow users to build on their single canvas and allow them to add an about page or contact info page if they so wished. This would require figuring out how to make the difference between each page users are customising visually distinct and clear. I would also like to allow users to make their canvases have an adjustable height if they so choose.

Accessibility could also be improved even further. While Orbit as it currently is has strived to include clear feedback messages and validation, further improvements could be made by adding in a WCAG checker. This would aim to help user understand whether their chosen colours, text sizes and layouts were accessible. It would not aim to stop users from doing what they wanted but would ensure users were aware if something was going to be difficult to read for example.

Another future improvement that could be made would be to add more node types. Future versions of the application could include calendar nodes, polls and data visualisations nodes for example. The aim would be to give people as many options as possible in creating their canvas.

I would also like to add a form of admin moderation. While I would not control how people designed their canvases, currently I am unable to delete things from the marketplace. Canvases and cards can only be removed if owned by the user. I would ensure that admin accounts would have more privileges to delete anything unsavoury from the marketplace.

A final improvement I would like to make is the market design. The market currently works functionally but overall, I feel like the design is weak in comparison to other pages on the app. I would like to revisit the design along with other pages and make sure that the design feels consistent and the flow between pages is consistent.

8.4 Final Conclusion

Orbit did not aim to solve the issue of web homogenisation completely. Instead, I designed it to question, why do we use platforms the way we do? What makes a

platform enjoyable, interesting, or visually distinct? How much freedom should users have over the spaces they create online?

Through this project, I wanted to show that the imperfect, human side of the web can still exist alongside modern technologies. Although the earlier web was often chaotic, inconsistent, and frankly ugly, it also gave people freedom to experiment, create, and express themselves.

I came across this quote that strongly connects to this idea:

> **“**We’re in danger of losing what’s made the Internet the most important medium in history – a decentralized platform where the people at the edges of the networks – that would be you and me – don’t need permission to communicate, create and innovate.” - Dan Gillmor

This quote reflects the concern at the centre of Orbit. As the web becomes more centralised, standardised, and controlled by large platforms, users are often given fewer opportunities to shape their own online spaces.

On a personal level, this project feels like a reflection of my time in college. I have never considered myself the strongest programmer, but my interest in technology, design, and creativity is visible throughout Orbit. At the beginning of the project, I was nervous about whether I could build something of this scale by myself. However, completing Orbit has improved my confidence and shown me how much I have developed since I began college in first year.

I am excited to continue working on Orbit following the end of this project and I have many ideas on how to further improve it.

One of the most rewarding parts of the project was seeing my friends create and share their own canvases. Everyone used the same system differently, which reinforced the original aim of the project.

I have included examples of some orbit user’s pages here, with their consent:

Figure 107 User 'Sadb'

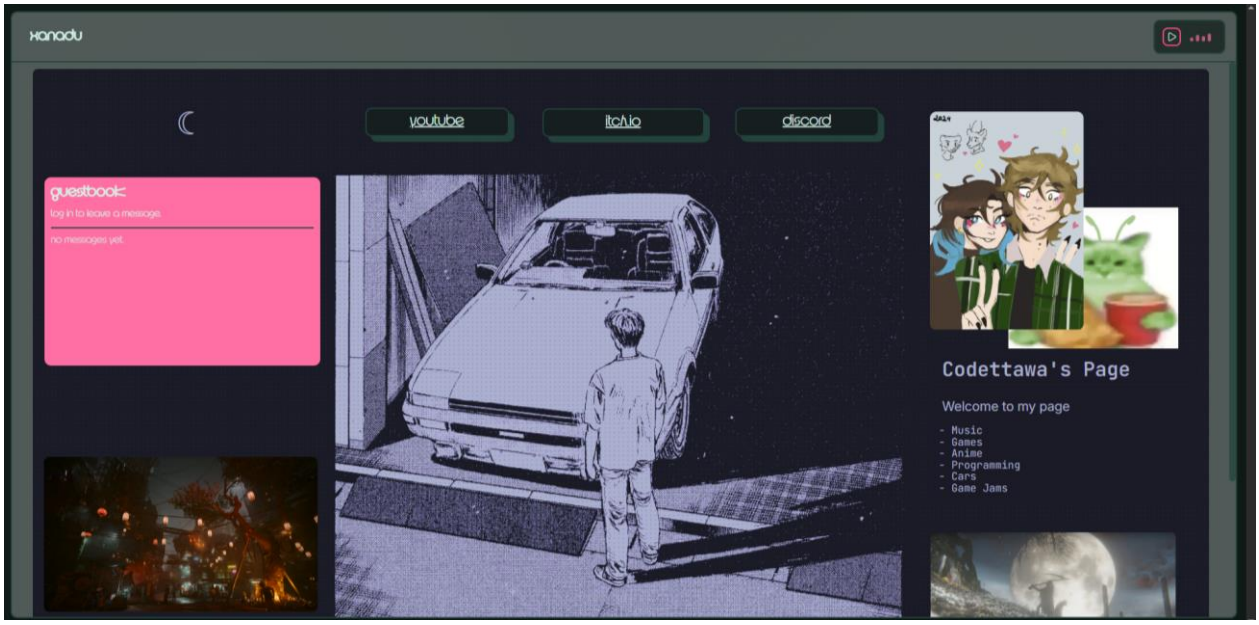


Figure 108 User 'Xanadu'

Thank you so much for reading!

9. References

Abrams, Z. (2021, December 2). *How can we minimize Instagram's harmful effects?* American Psychological Association. https://www.apa.org/monitor/2022/03/feature-minimize-instagram-effects

Atlassian. (2025). *What Is a UML Diagram: Types, Examples, and Uses | Atlassian*. **Atlassian. https://www.atlassian.com/work-management/project-management/uml-diagram**

Baxter, A. (n.d.). *The profound impact of human touch in design Vs artificial intelligence*. JUMP. https://www.wesayhowhigh.com/blog/article/profound-impact-human-touch-design-vs-artificial-intelligence

Bell, A. (2024, April 2). *How we're approaching theming with modern CSS*. Piccalilli. https://piccalil.li/blog/how-were-approaching-theming-with-modern-css/

Cesarato, A. (2020, April 21). *Susan Kare: an Iconic Career*. Domestika; DOMESTIKA. https://www.domestika.org/en/blog/3320-susan-kare-an-iconic-career?exp_set=1

Chayka, K. (2024). *Filterworld*. Heligo Books.

Davis, N. (2025, September). *Who Owns the Web Now? Centralization vs. Decentralization in the Age of AI*. Web Designer Depot. https://webdesignerdepot.com/who-owns-the-web-now-centralization-vs-decentralization-in-the-age-of-ai/

Deriba, F., Saqr, M., & Tukiainen, M. (2025). *Exploring the role of AI in enhancing accessibility in user interface design: A qualitative study of student experiences* (p. pp 437–447). https://doi.org/10.1007/978-981-96-5658-5_44

Douglas, C. (2023, October 29). *What is a User flow in UX Design? - Visily*. Visily. https://www.visily.ai/blog/what-is-a-user-flow-in-ux-design/

Dykes, T. (2015, February 16). *Personas Make Users Memorable*. Nielsen Norman Group. https://www.nngroup.com/articles/persona/

Gillmor, D. (2026). *Why the Indie Web movement is so important*. Dangillmor.com. https://dangillmor.com/2014/04/25/indie-web-important/

Goree, S., Doosti, B., Crandall, D., & Su, N. M. (2021). Investigating the Homogenization of Web Design: A Mixed-Methods Approach. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3411764.3445156

Jen, C., & Gibbons, S. (2025, May 30). *The Template Trap: How Template Culture Is Dumbing Down UX*. Nielsen Norman Group. https://www.nngroup.com/articles/template-trap/

Kahle, B. (2015). *Locking the Web Open: A Call for a Decentralized Web | Brewster Kahle's Blog*. Kahle.org. https://brewster.kahle.org/2015/08/11/locking-the-web-open-a-call-for-a-distributed-web-2/

Kohlstedt, K. (2018, July 6). *Designed with Kare: Influential Graphics of Apple's Early Macintosh Computers*. 99% Invisible. https://99percentinvisible.org/article/designed-with-kare-influential-macintosh-graphics-of-early-apple-computers/

Lindahl, N. (2025, September 20). *The Indie Web is leading a quiet rebellion against big tech*. Medium. https://medium.com/@Nathans_Tweets/the-indie-web-is-leading-a-quiet-rebellion-against-big-tech-f53e32ad11a5

Qiu, Y. (2024). Social Comparison on Social Media Platforms: A media and communication Perspective. *SHS Web of Conferences*, *185*(185), 03008–03008. https://doi.org/10.1051/shsconf/202418503008

Reynolds, C., & Hallinan, B. (2020). The haunting of GeoCities and the politics of access control on the early Web. *New Media & Society*, *23*(11), 146144482095160. https://doi.org/10.1177/1461444820951609

Sanjay Kushwah. (2025, August 7). *What is a Wireframe?* Medium. https://medium.com/@sanjay-kushwah/what-is-a-wireframe-761760dd1029

Sarah Davis Baker. (2025, April 14). *The Internet Used to Be a Place*. YouTube. https://www.youtube.com/watch?v=oYlcUbLAFmw

Veera Harish Muthazhagu, & Surendiran B. (2024). *Exploring the Role of AI in Web Design and Development: A Voyage through Automated Code Generation*. https://doi.org/10.1109/iitcee59897.2024.10467409

10. Appendix

Appendix A: Visual Research

Initial Notes:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764670165254144&cot=14

Kinopio Visual Research:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669787109374&cot=14

Neocities Visual Research:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669787003533&cot=14

Appendix B: Survey Materials and Results

Initial Survey Conducted before Development:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669444436698&cot=14

User Testing Survey:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669598147428&cot=14

Performance Testing:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669785778759&cot=14

Appendix C: Design Evidence

Wireframes and User Flows:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669787237455&cot=14

Personas:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669446261506&cot=14

Figma Designs:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764670165547831&cot=14

Appendix D: Technical Evidence

Database and ERD:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669448097674&cot=14

Use Case Diagrams:

https://miro.com/app/board/uXjVJ5az30k=/?moveToWidget=3458764669518492574&cot=14

Appendix E: Supporting Documentation

GitHub:

https://github.com/Ekiizu/FINAL-PROJECT_EKIIZU

Miro:

https://miro.com/app/board/uXjVJ5az30k=/?share_link_id=48004344737

Playlist of videos demonstrating development:

https://youtube.com/playlist?list=PL7vsJcL6kFNOPqdmX28egPymQvZ_2xuOP&si=IXIlhlYa2VLmAolb

Website:

https://final-project-ekiizu.onrender.com/