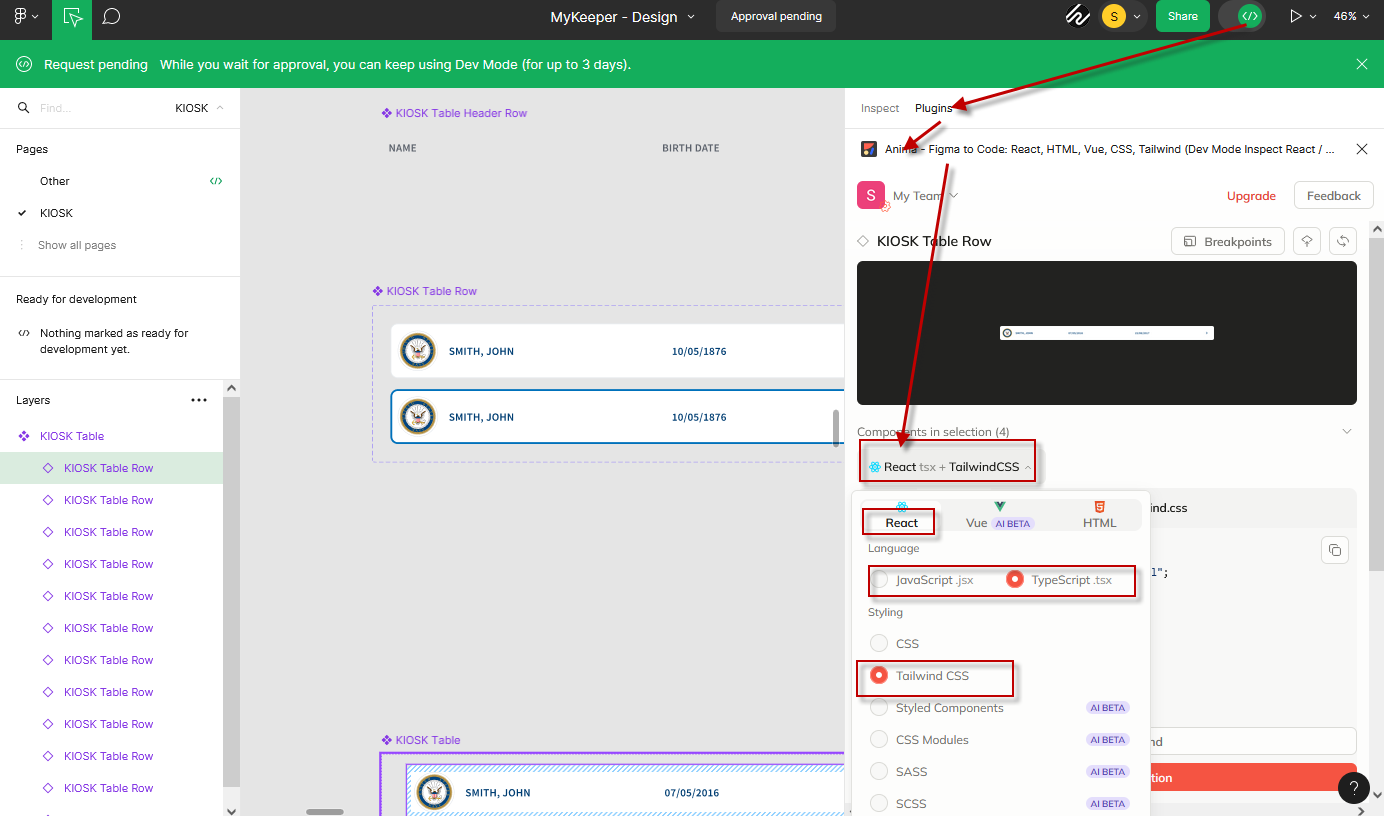
**Convert Figma element to React/Nextjs code with Tailwind css.**

(Convert 1 simple page may take 1 – 2 hours)

Steps:

1. On Figma, go to Dev Mode, install and open plugin Anima. The Anima will auto generate React code with tailwind css from selected Figma layer.

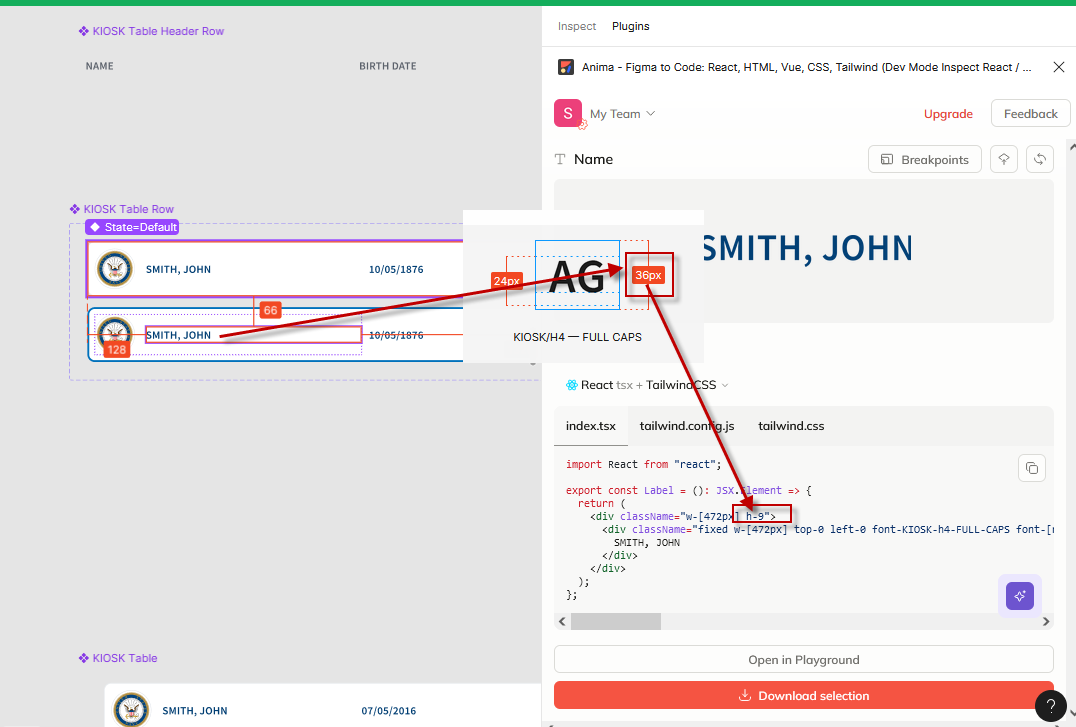


2. Convert Figma objectstyle to tailwind classes.

Case A: Convert to Built-in tainwind class,

when style Property value match tailwind class property value.

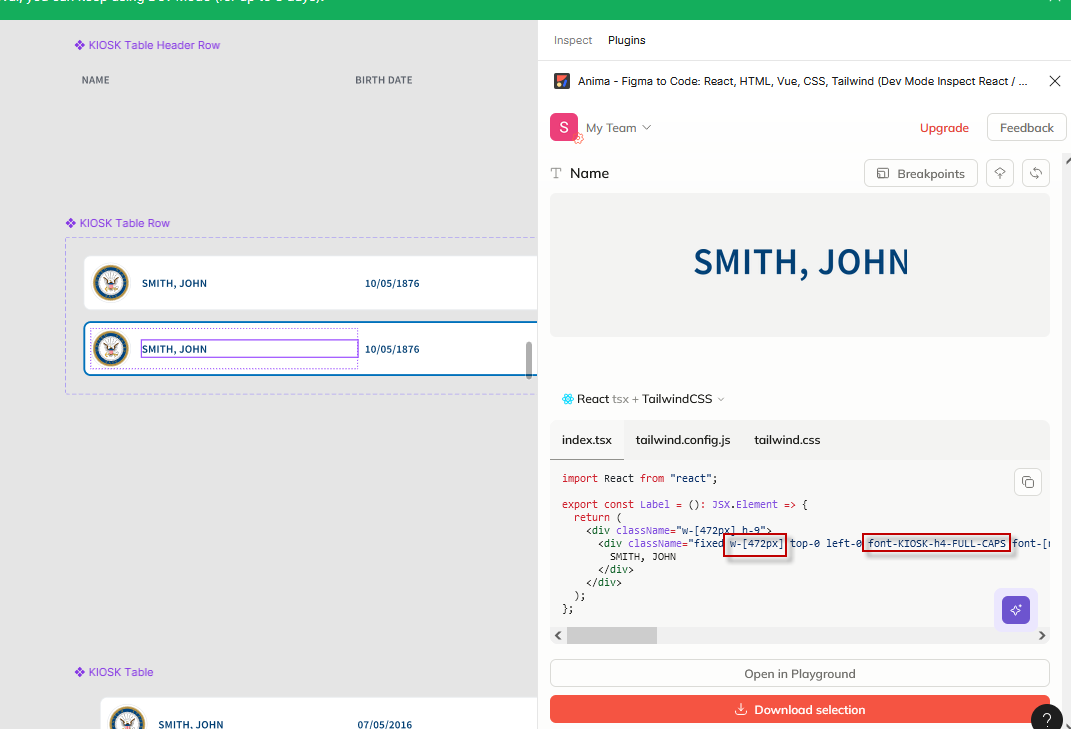
If Anima detect a style property value match a tailwind class property value, it will auto convert it to related tailwind class. For example, if the object height is set to 36px on figma, it will be converted to tailwind class “h-9” as below.



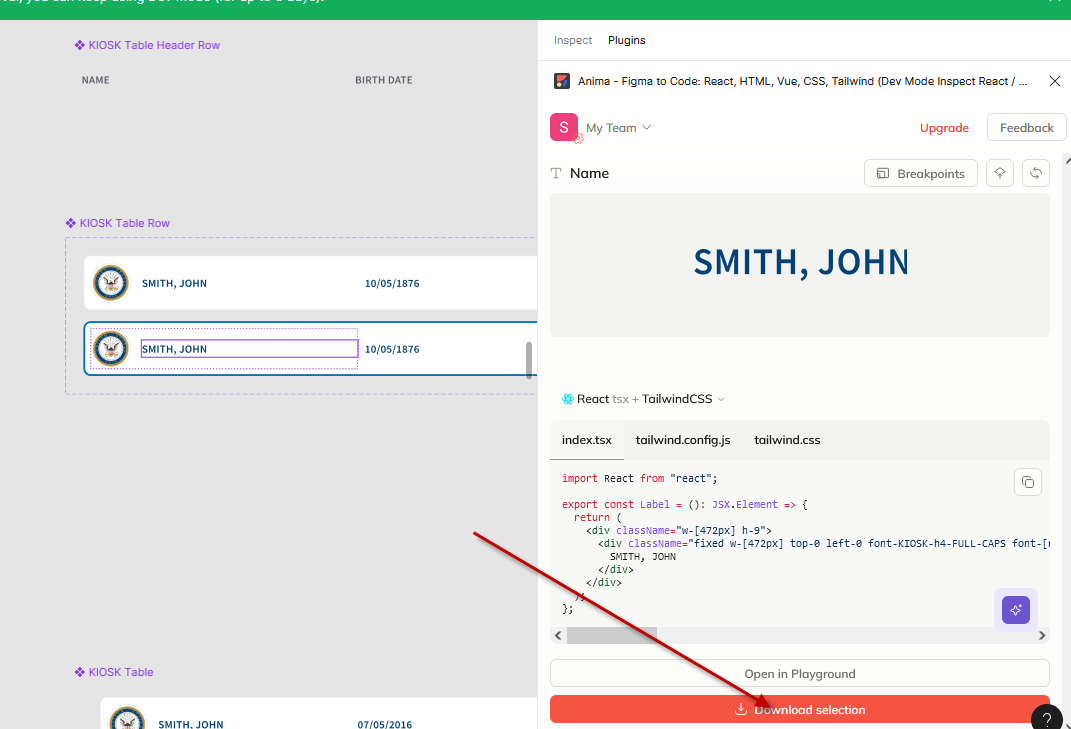
Case B: Generate to new user defined tailwind valuables and classes ,

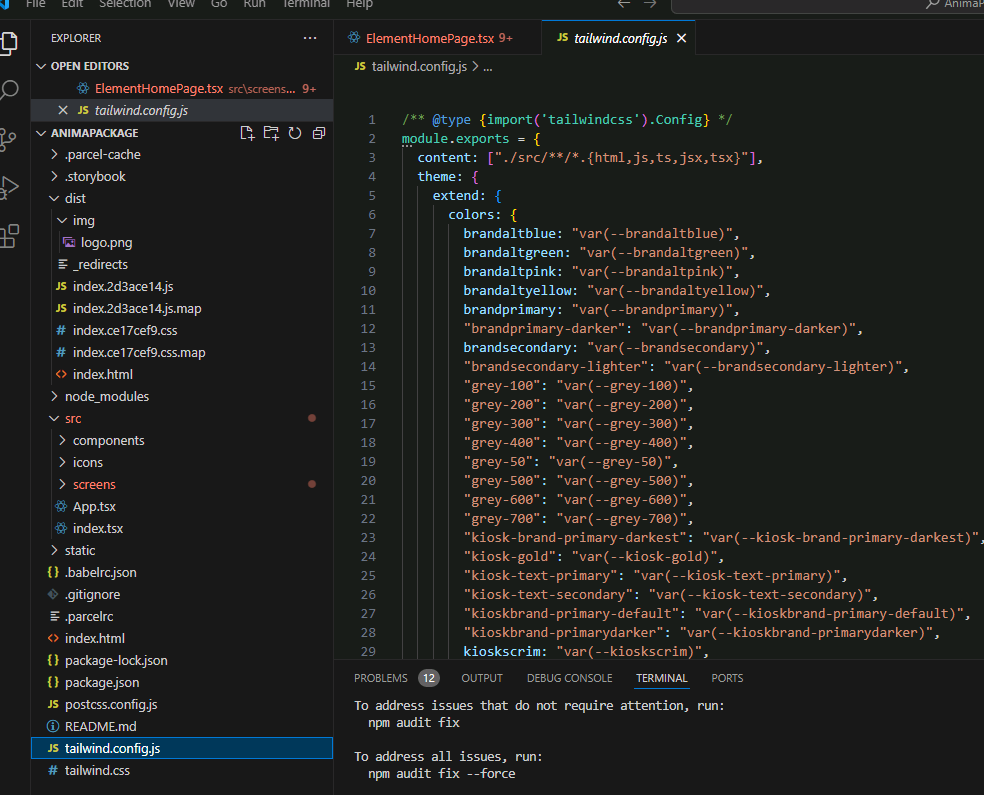
when there is no matching built-in tailwind class found, or a figma valuable is used.

The user defined tailwind classes and config will be auto generated on tailwind.config.js and tailwind.css



3. Download and open the generated react+tailwind app project.





4. Use the code as reference, create related page/component in our own nextjs app.

4.1 Merge the generated tailwind.config.js to nextjs tawilwind.config.js

4.2 Merge the generated tailwind.css, which contains all the generate valuables/parameters to nextjs app main style file.

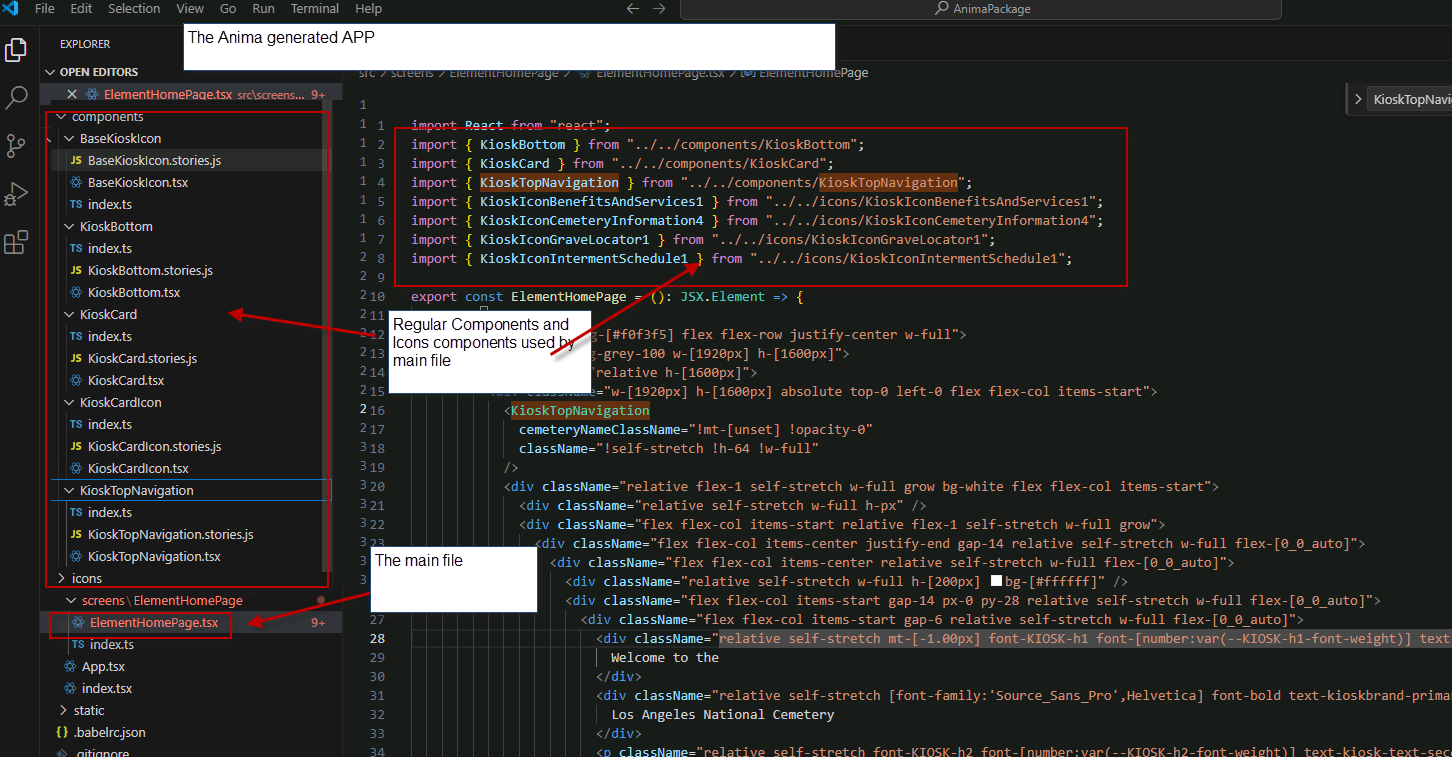
4.3 Merge generated images in /img/ folder to nextjs app

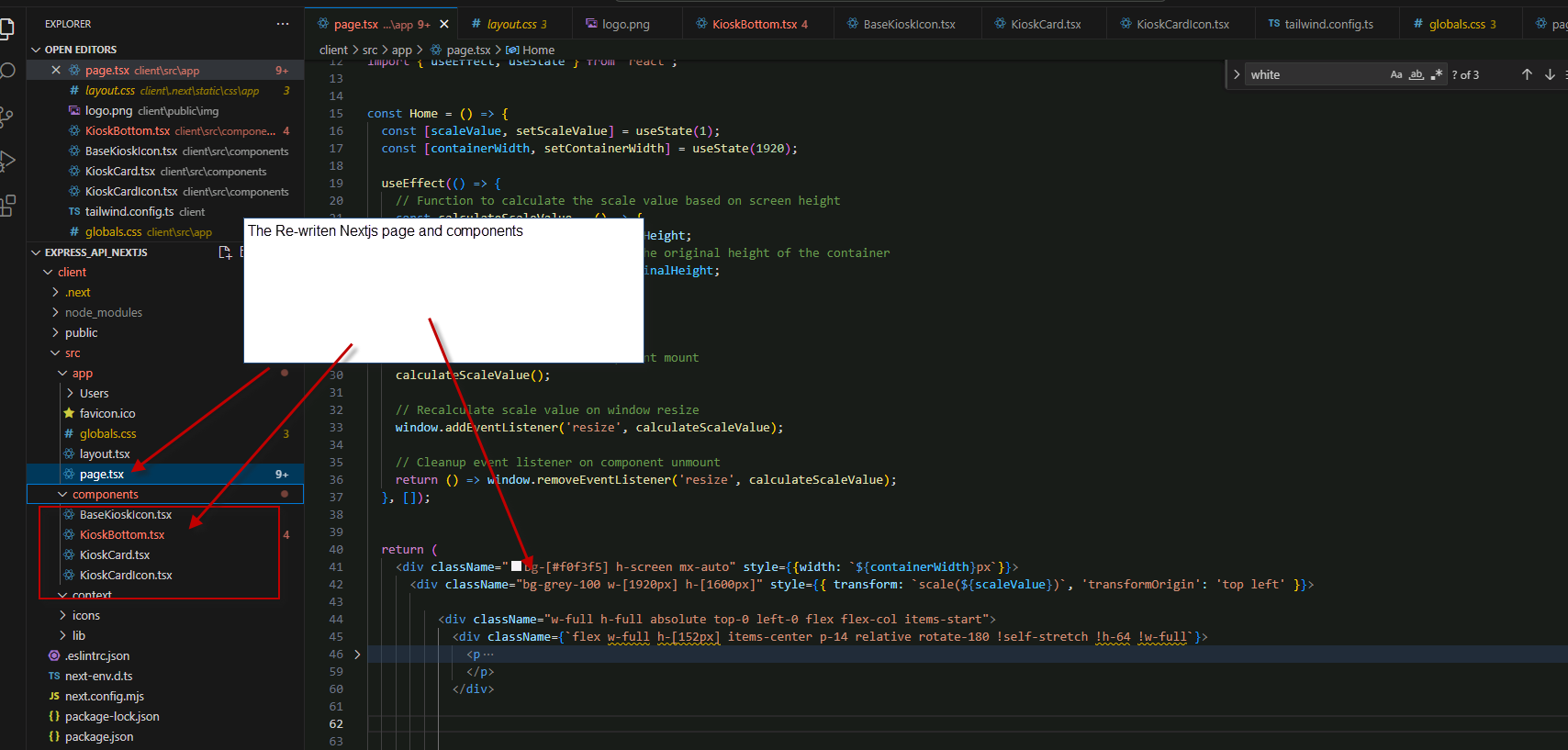
4.4 Create NextJs page or component reference to the generated react codes.

It’s most time consuming part, need to read the original long codes of pages and components, rewrite the codes to make it simpler with nextjs, make responsible, and etc.

The generated codes is much more “long” than human written code, and it’s not responsive. Need to reduce the code and make it responsive while keep the layout base position, size color, and font.

The generated components can also be reduced. (It generated a lot of unnecessary components).





**5. Different cases of original Figma files**

**5.1what if the Figma designer doest know tailwind**

If the designer does not know tailwind, the generated code will have a lot of new user-defined tailwind valuables/parameters and classnames.

It will be a lot of work to read and maintain the code.

For example:

<divclassName="relative self-stretch mt-[-1.00px] font-KIOSK-h1 font-[number:var(--KIOSK-h1-font-weight)] text-kiosk-text-secondary text-[length:var(--KIOSK-h1-font-size)] text-center tracking-[var(--KIOSK-h1-letter-spacing)] leading-[var(--KIOSK-h1-line-height)] [font-style:var(--KIOSK-h1-font-style)]">

Developers need to either rewrite the above code manually with built-in tailwind class according to the generated tailwind parameters,

or leave it as is but hard to read and modify, (Example: it’s hard to ke the above div responsive by edit the classNames).

**5.2 what if the Figma designer knows tailwind**

When using Figma in conjunction with the Anima plugin to generate React + Tailwind CSS code, it's essential for the Figma designer

to follow certain best practices to ensure the generated code is clean, efficient, and aligns well with Tailwind CSS principles.

### Setting Up Tailwind's Design Tokens in Figma

**A. Tailwind's Spacing Scale**

Tailwind uses a predefined set of spacing values. You can set these up in Figma as a style guide:

1. **Create a Spacing System in Figma:**
   * Open your Figma file.
   * Go to the **Design System** page (create one if not already present).
   * Create a new Frame or Group and name it "Spacing."
   * Add Text layers representing each of Tailwind's spacing units.

Here's a breakdown of some Tailwind spacing units and their equivalent pixel values:

| **Tailwind Class** | **Pixel Value** |
| --- | --- |
| p-0.5 | 2px |
| p-1 | 4px |
| p-2 | 8px |
| p-3 | 12px |
| p-4 | 16px |
| p-5 | 20px |
| p-6 | 24px |
| p-7 | 28px |
| p-8 | 32px |
| p-9 | 36px |
| p-10 | 40px |
| ... | ... |

1. **Create Text Labels:**
   * For each spacing unit, add a label with the Tailwind class name, such as p-1, p-2, etc.
   * Set the text's position next to its corresponding visual representation to indicate the value.
2. **Visual Representation:**
   * Use rectangles to represent each spacing unit. For example, draw a rectangle with a width of 8px and height 8px for p-2.
   * Align rectangles and labels side by side for easy comparison.
3. **Add Annotations:**
   * Include annotations that explain each spacing value's purpose. For instance, "Use for small padding" or "Common for container margins."

**B. Tailwind's Color Scale**

Tailwind provides a comprehensive color palette with shades for each color. Here’s how to set up a similar color system in Figma:

1. **Create a Color Style Guide:**
   * Open the **Design System** page.
   * Create a Frame named "Colors."
   * Add rectangles for each color and shade provided by Tailwind.
2. **Define Tailwind's Colors:**

Tailwind's color classes have names like bg-blue-500, text-gray-800, etc. Here’s a sample color setup:

* + **Gray:**
    - gray-50: #F9FAFB
    - gray-100: #F3F4F6
    - gray-200: #E5E7EB
    - ...
    - gray-900: #111827
  + **Blue:**
    - blue-50: #EFF6FF
    - blue-100: #DBEAFE
    - blue-200: #BFDBFE
    - ...
    - blue-900: #1E3A8A
  + **Primary, Secondary Colors:** Add other primary colors like red, green, yellow with different shades.

1. **Create Figma Color Styles:**
   * Select each rectangle and create a color style by clicking the + icon under the Fill section.
   * Name each style using Tailwind's naming convention, like blue-500, gray-800, etc.
2. **Use Color Tokens:**
   * Use these styles consistently throughout your Figma designs.
   * Apply these color styles to backgrounds, text, and borders where applicable.

**C. Tailwind's Typography Scale**

Tailwind has a defined typography scale for font sizes, weights, and line heights. You can replicate this in Figma by creating text styles.

1. **Set Up Text Styles in Figma:**
   * Go to the **Design System** page.
   * Create a Frame named "Typography."
2. **Define Text Styles:**
   * Create text layers for each typography class:

| **Tailwind Class** | **Font Size** | **Line Height** |
| --- | --- | --- |
| text-xs | 12px | 16px |
| text-sm | 14px | 20px |
| text-base | 16px | 24px |
| text-lg | 18px | 28px |
| text-xl | 20px | 28px |
| text-2xl | 24px | 32px |
| ... | ... | ... |

1. **Create Figma Text Styles:**
   * Select each text layer, then create a text style by clicking the + icon under the Text section.
   * Name the style using Tailwind's naming convention, like text-base, text-lg, etc.
2. **Typography Weights:**
   * Add font weights such as font-normal, font-semibold, font-bold, etc.
   * Apply these styles to headings, paragraphs, and other text elements throughout your design.

**Using Tailwind Tokens in Your Designs**

Once you have set up these tokens in Figma, you can start using them in your design process:

* **Apply Consistent Spacing:** Use the spacing scale for padding and margins on frames, elements, and components.
* **Consistent Colors:** Apply color styles to backgrounds, text, borders, and shadows to ensure consistency with Tailwind.
* **Typography:** Use text styles for all text elements to maintain a consistent typographic scale.

**Exporting and Syncing with Anima**

When using the Anima plugin to generate code, ensure the following:

* **Name Layers and Styles Clearly:** Use descriptive names so that the exported code retains clarity.
* **Check Anima Settings:** Ensure Anima is set up to respect Figma styles and exports them as Tailwind classes.
* **Review Generated Code:** After exporting, review the generated code to ensure Tailwind's utility classes are applied correctly.

**Tips for Figma Designers**

* **Keep Updated:** Tailwind CSS regularly updates its design tokens. Make sure your Figma tokens are current with the latest Tailwind version.
* **Design Responsively:** Use Figma's responsive layout features to ensure designs look good on all screen sizes, mirroring Tailwind's responsive utilities.
* **Collaborate with Developers:** Work closely with developers to ensure designs are feasible and aligned with the codebase.