# From the ground up: Implementing a re-usable and efficient web app code base

Submitted By: Abigayle Hickey

HYKE

100 Signal Hill Road
St. John’s, NL, Canada
A1A 1B3

T 1-709-769-1986
**hello@hykeup.com**

Dec 17th, 2021
Claire Avery

Co-operative Education Office
Faculty of Engineering and Applied Science
Memorial University of Newfoundland
St. John’s, NL
A1B 3X5

Dear Ms. Avery,

During this work term (Engineering 005W) I was employed with HYKE as an undergraduate co-operative student under the software development team. This is my second work term with HYKE, and I was supervised by Jillian Breau.

HYKE is a local start-up that aims to eliminate unnecessary coupon-inspired or accumulative point made purchases by offering personalized offers for their users, given the type of item they want to buy and how much they want to spend. HYKE will generate a unique offer with any of their linked stores that can be used in a given time frame.

This company has allowed me to contribute my own designs and code to both their front-end and backend code. The enclosed presentation package titled *From the ground up: Implementing a re-usable and efficient web app code base* outlines my experience aiding in the implementation of a web app, from the very beginning.

If there are any questions concerning this report, I would be pleased to discuss them further with you.

Sincerely,

Abigayle Hickey

Abigayle Hickey

# From the ground up: Implementing a re-usable and efficient web app code base

Work Term Five Presentation Package

HYKE

**Submitted To:**

Ms. Claire Avery

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

Dec 17th, 2021

**Created By;**

Abigayle Hickey

201732641

## Table of Contents

## Presentation Overview:

## 1.0 – Introduction

### 1.1– Who is HYKE

HYKE is a local start up, based through the Genesis Enterprise program. HYKE has developed an app aimed at eliminating unnecessary purchases inspired by coupons or accumulative points by offering consumers a discount for the store of their choice at their desired price point. The Hyke app will generate a unique coupon for the user to redeem within a set time frame [1].

Hyke has future plans of mapping out users spending habits, and through AI generating offers at specific times that the user will most likely choose to redeem. In time, user inputs should decrease as the AI learns what the user is most likely to buy at specific times. Through this process, Hyke is aiming to learn what percentage discounts combined with the time frame of the coupon expiry will best drive people to redeem the offer. Hyke also aims to be completely transparent with this information by making it accessible by each user to better understand their own spending habits [1].

### 1.2– My project

This is my second work term with Hyke, and through my first experience I aided in the completion of the frontend web app design, and front and backend unit testing. Since the beginning of this work term, the web app underwent a completely new design and as such required to be implemented from the very beginning.

I was able to contribute to both front and backend implementation and aided in the planning and execution of this implementation. Through trail and error of this process, I have summarized the best steps to take to collaboratively implement a design.

## 2.0– Implementing a web app

As the implementation of a web app is a team effort, the design should be thoroughly assessed to distribute tasks in the most efficient way. It is important to outline the design theme and repetitive components for the team to collaboratively utilize.

### 2.1– Creating a themes and tailwind configuration file

The overall styling of the web app should be first created inside a themes file. This file will contain all the default styling information the application will use, such as font style, size, color, etc. This styling file, or CSS file, eliminates the risk of the team constantly re-writing the same styling code that will be utilized by the whole project.

Tailwind is a CSS framework that is useful for improving the efficiency of styling. Through tailwind, styling code does not require its own CSS file, and rather can be incorporated into the elements directly, eliminating half the files that the project will need. CSS files also require elements to be tracked using a “class name”, which is then defined inside the CSS file. Class names must be unique to their element, otherwise will be overwritten and their style will be lost. This can cause problems for large teams of programmers individually making elements, thus tailwind removes this by not requiring class names, and rather writing the styling directly within the element.

A tailwind configuration file holds all the recurring styles of the design. This includes the theme colors, font sizes, etc. It is best practice to define this configuration file first, by accessing the design and seeing what styles will be needed. Given this file, all programmers will be using the exact same styling, and there will be no variation within the code.

### 2.2– Creating recurring components

In addition to thematic styles, it is also important to take note of the components that repeat throughout the design. A problem our team experienced was code blockages due to multiple programmers requiring the same component. Instead of both implementing the same component, one would be blocked waiting for the others to be finished and used.

To eliminate this problem we faced, it would be more efficient to first outline the components that repeat throughout the web app, and assign these to be implemented first, individually. Components can be created so that they intake certain values, such as titles or colors, to make their design unique to them, but similar to the overall component design. In this way, when all components are created, the team can utilize the pool of resources and implement their code without code blockages.

### 2.3– Using utils files for components

Components often require functions to allow them to succeed the expected interactive outcome. It's best practice to keep these functions in a separate utils file, to ensure each file has minimal purpose for best readability. This also ensures that if a component requires the same function but is found on two separate pages and thus different files, instead of repeating that code, the component can access the function inside the same file from two different places.

As well variable names components use should be stored in their utils files. This reduces repetitive code and makes the re-designability of the application easier as every function and variable name can be found in the same place. Given a re-design, the one utils file would be updated compared to updating every file that would have used those functions or variables.

## 3.0– Conclusion and Reflection

This work term was my first experience aiding in the implementation of a web app from the very beginning. As such, our team learned a lot of the best measures to take through trials and errors. For best efficiency, extra planning should be taken for mapping out the styles and components that every programmer will require to use. After creating the themes and component files, the team will all have access to the same pool of resources, which will reduce code blockages, increase code consistency, and make the overall implementation very efficient.

As well, using utils files for each component will make their respective functions and variables most consistent and non repetitive. Each programmer will be able to access the required information from the same file. In this way, if the design changes, only the one file will require updates compared to multiple files all using the same information.

From this work term I learned a lot about working within a team and coordinating tasks efficiently. Given this work term again, I would take more precautions to carefully create an implementation plan, by assessing the styles and components individually, before implementing the web app further. Through this way, the web app will be able to be most efficiently developed, eliminating code blockages and keeping the code as consistent and readable.

## 4.0 – References

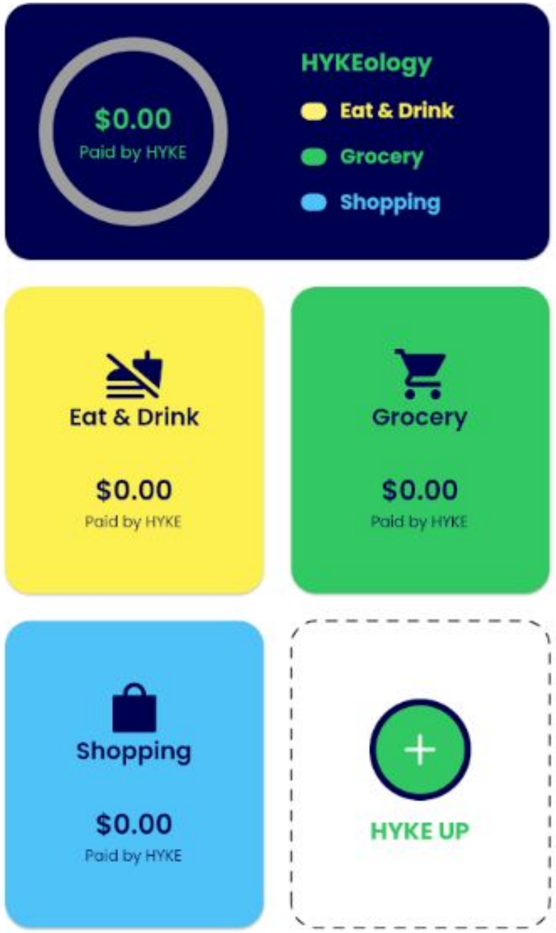
[1] "HYKE," Hyke Technologies Inc, 2021. [Online]. Available: https://hykeup.com/. [Accessed November 2021].

[2] "tailwindcss," [Online]. Available: https://tailwindcss.com/. [Accessed November 2021].

HYKE

# IMPLEMENTING A REUSABLE AND EFFICIENT WEB APP CODE BASE

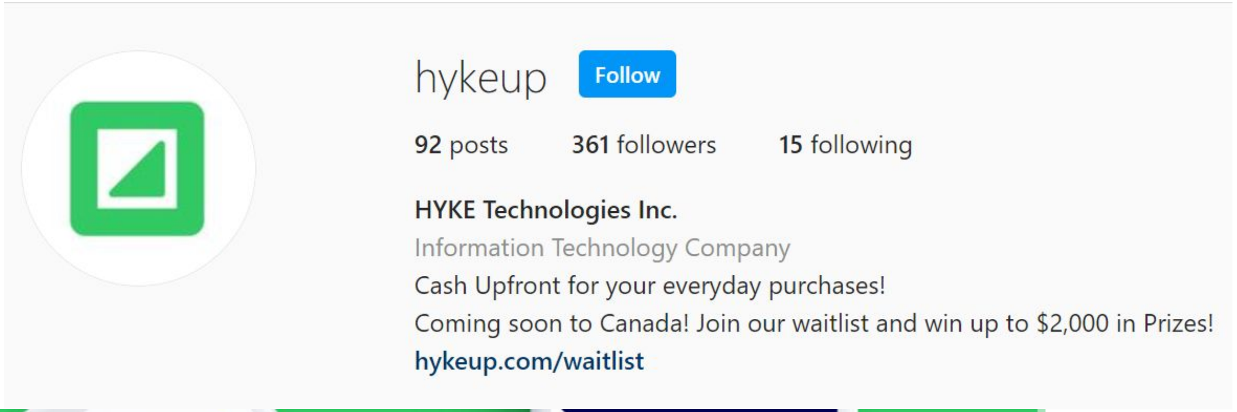
By: Abigayle Hickey



## WHO IS HYKE?

- Local Genesis startup
- Designed to alleviate unnecessary purchases made through coupons or point saving
- The Hyke app generates a personalized offer instantly based on where and how much the user wants to spend
- Future plans to patternize users spending style and generate better offers based on the users history

# FOLLOW US FOR LAUNCH UPDATES!!



hykeup Follow

92 posts 361 followers 15 following

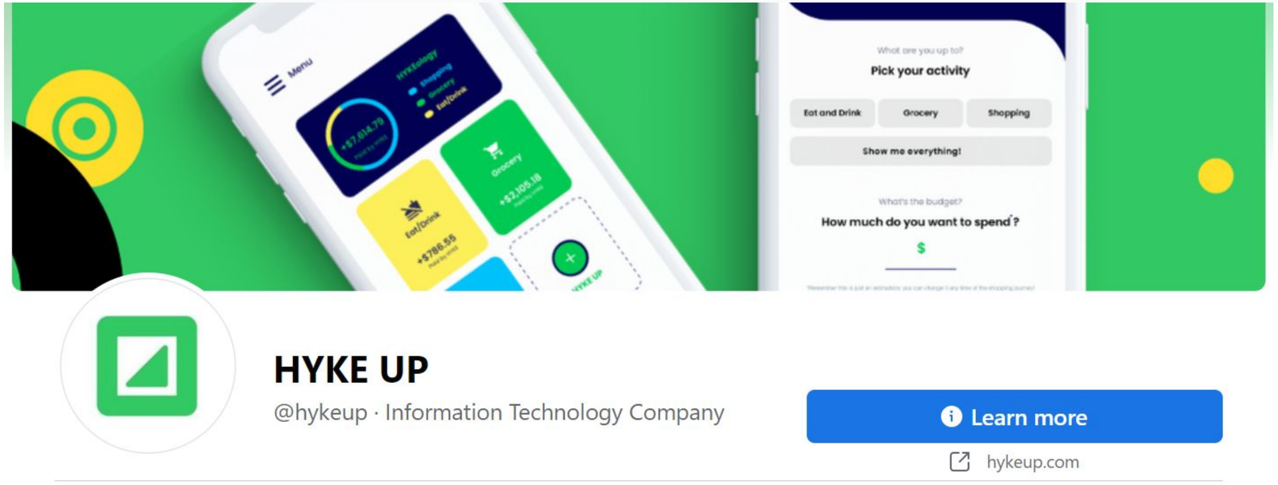
**HYKE Technologies Inc.**

Information Technology Company

Cash Upfront for your everyday purchases!

Coming soon to Canada! Join our waitlist and win up to $2,000 in Prizes!

hykeup.com/waitlist



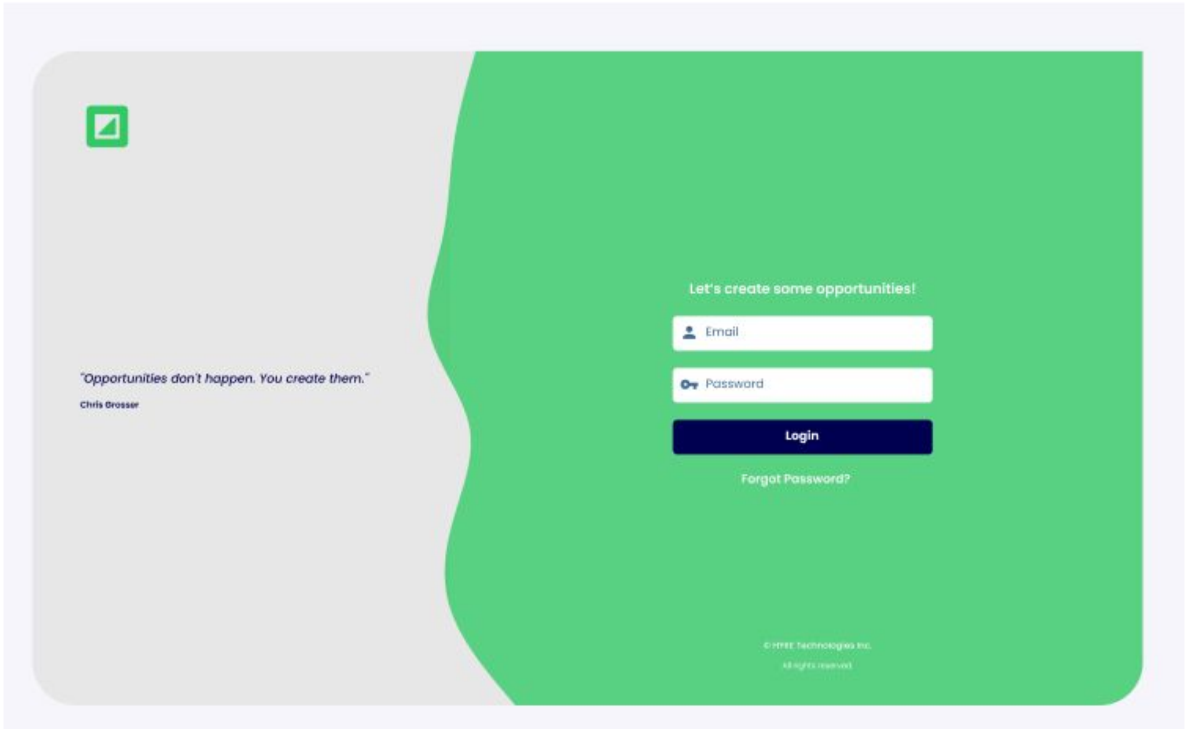
**HYKE UP**

@hykeup · Information Technology Company

Learn more

hykeup.com

# MY PROJECT



"Opportunities don't happen. You create them."

Chris Grosser

© HYKE Technologies Inc.

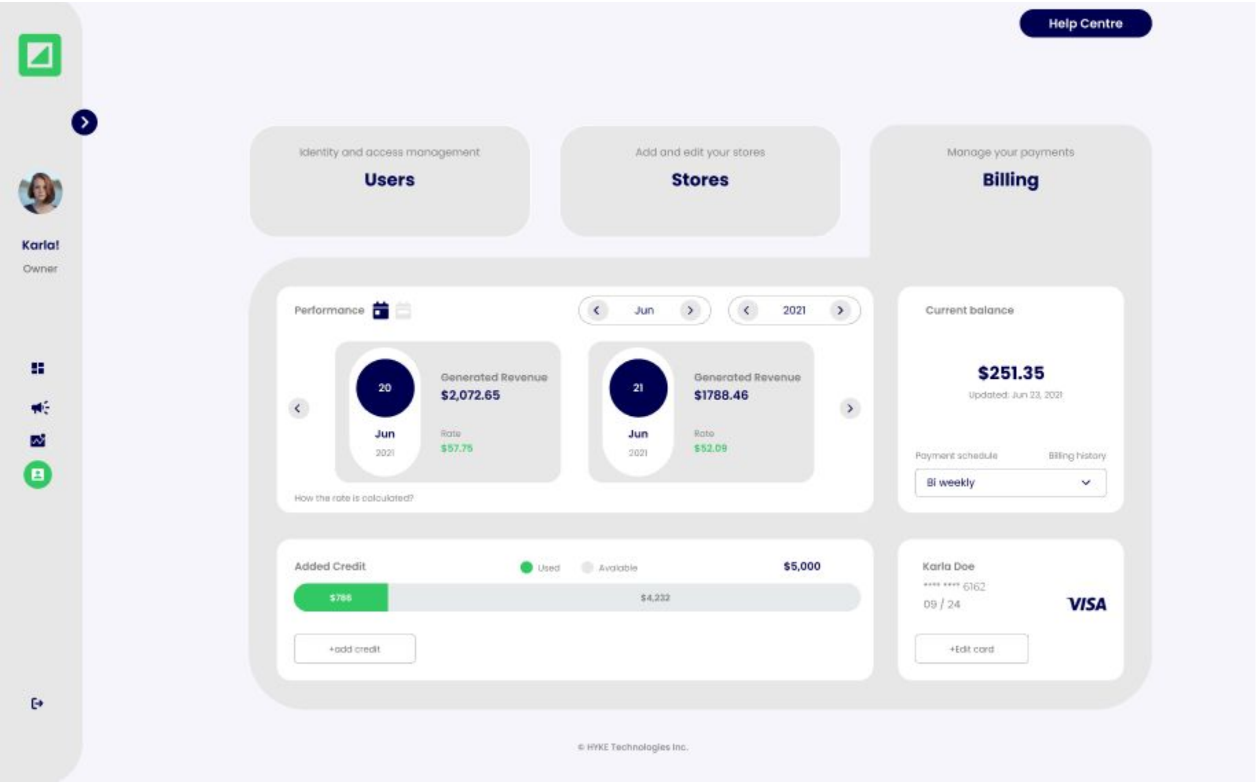
All rights reserved.

- Aided in the implementation of the hyke web application
- Implemented both front and backend
- Contributed in sprint planning, GitHub code reviews

## AGENDA

- How to assess the web app design for high level styling requirements
- Creating a themes and tailwind configuration file
- Creating reusable components
- Utilizing a utils file for recurring functions and variables

# ASSESSING THE DESIGN

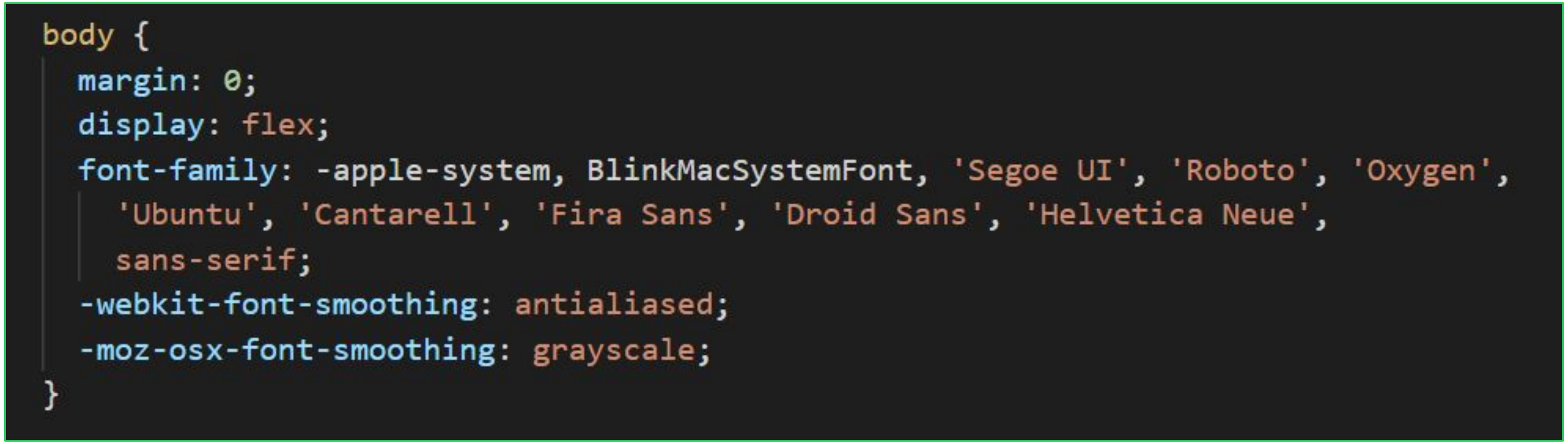


- First step, review total design
- Identify recurring themes, such as colors, fonts, components, etc
- From this design, there are clear color themes, as well as recurring components.

## CREATING A THEMES FILE

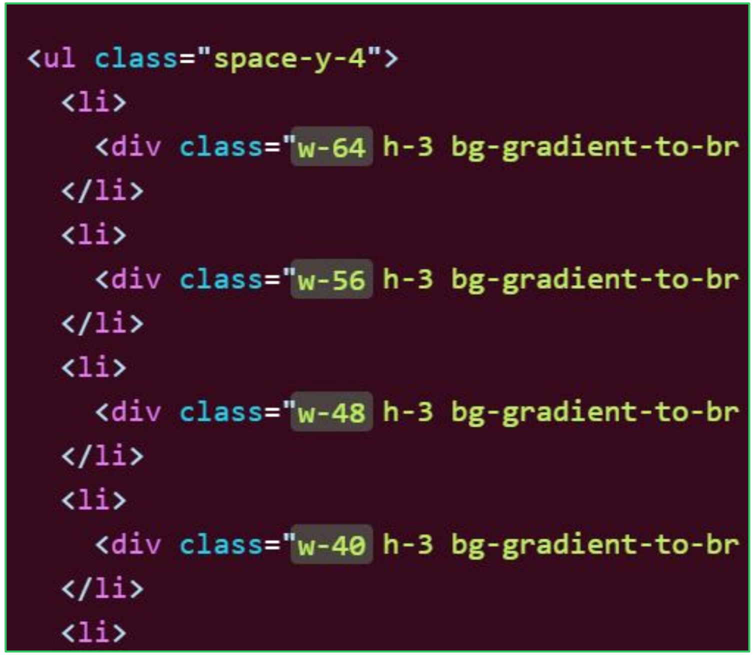
- A themes file contains all the highest level styling the web app will use
- This can include the default font style, size, color, etc.

**themes.css file**



```css
body {
  margin: 0;
  display: flex;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
```

# WHAT IS TAILWIND

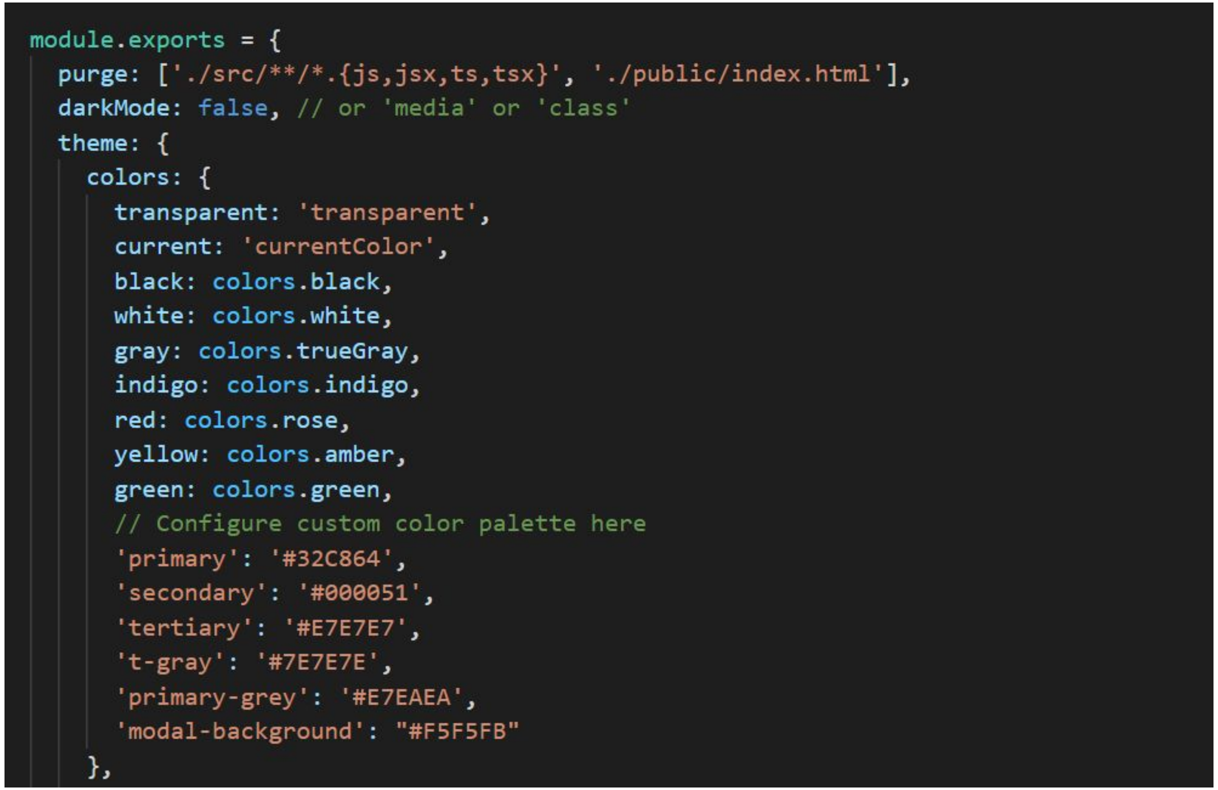


```
<ul class="space-y-4">
  <li>
    <div class="w-64 h-3 bg-gradient-to-br
  </li>
  <li>
    <div class="w-56 h-3 bg-gradient-to-br
  </li>
  <li>
    <div class="w-48 h-3 bg-gradient-to-br
  </li>
  <li>
    <div class="w-40 h-3 bg-gradient-to-br
  </li>
  <li>
```

- Tailwind CSS is a utility CSS framework that styles classes individually
- Reduces web app workspace in half as individual CSS files are no longer needed
- Eliminates possible styling dependencies, such as CSS classnames overlapping

# CREATING A TAILWIND CONFIGURATION FILE

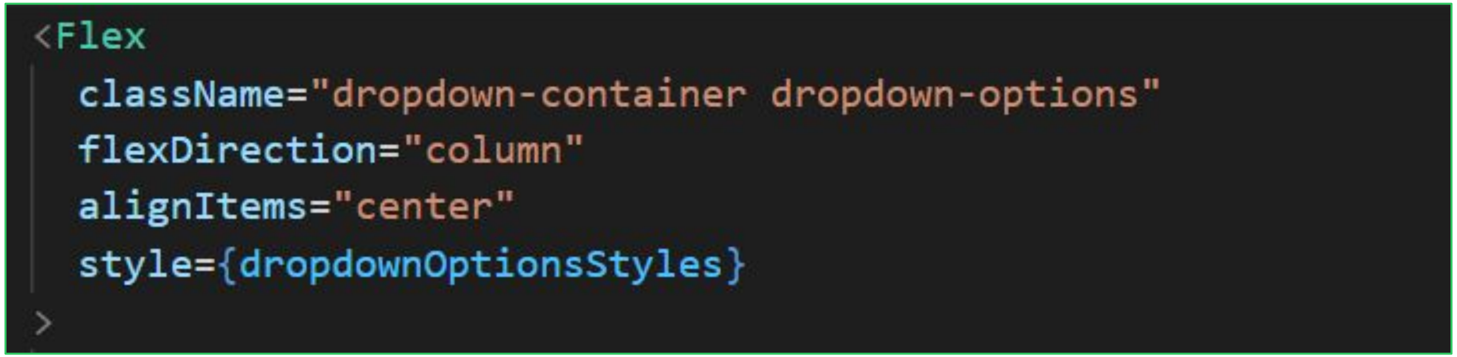
**tailwind.config.js file**



```
module.exports = {
  purge: ['./src/**/*.{js,jsx,ts,tsx}', './public/index.html'],
  darkMode: false, // or 'media' or 'class'
  theme: {
    colors: {
      transparent: 'transparent',
      current: 'currentColor',
      black: colors.black,
      white: colors.white,
      gray: colors.trueGray,
      indigo: colors.indigo,
      red: colors.rose,
      yellow: colors.amber,
      green: colors.green,
      // Configure custom color palette here
      'primary': '#32C864',
      'secondary': '#000051',
      'tertiary': '#E7E7E7',
      't-gray': '#7E7E7E',
      'primary-grey': '#E7EAEA',
      'modal-background': "#F5F5FB"
    },
```

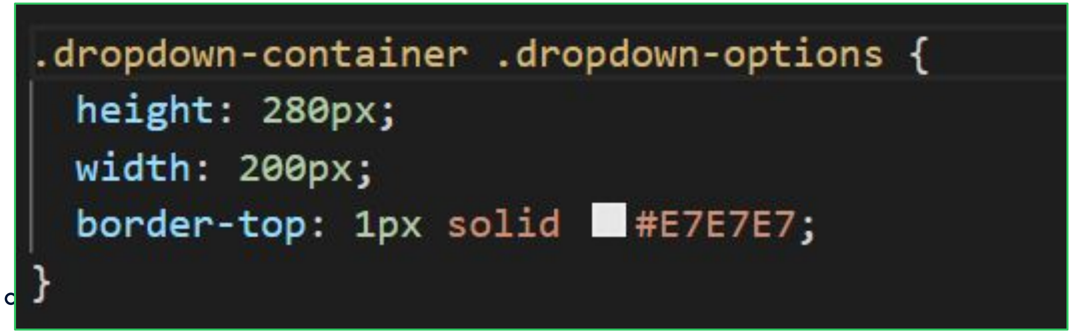
- Instead of default styles, can give names to repetitive styling options
- Can define colors, font sizes, container sizes, etc
- Can redesign certain style all at once in same file

## StoreDropdown.tsx

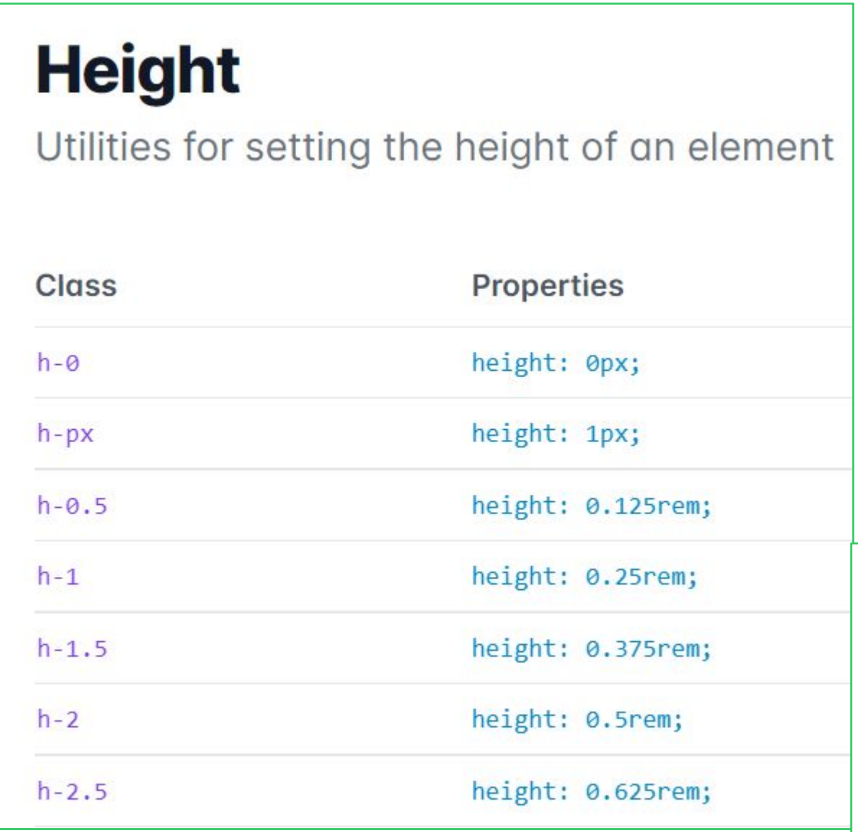


```
<Flex
  className="dropdown-container dropdown-options"
  flexDirection="column"
  alignItems="center"
  style={dropdownOptionsStyles}
>
```

## StoreDropdown.css



```
.dropdown-container .dropdown-options {
  height: 280px;
  width: 200px;
  border-top: 1px solid ⬜#E7E7E7;
}
```

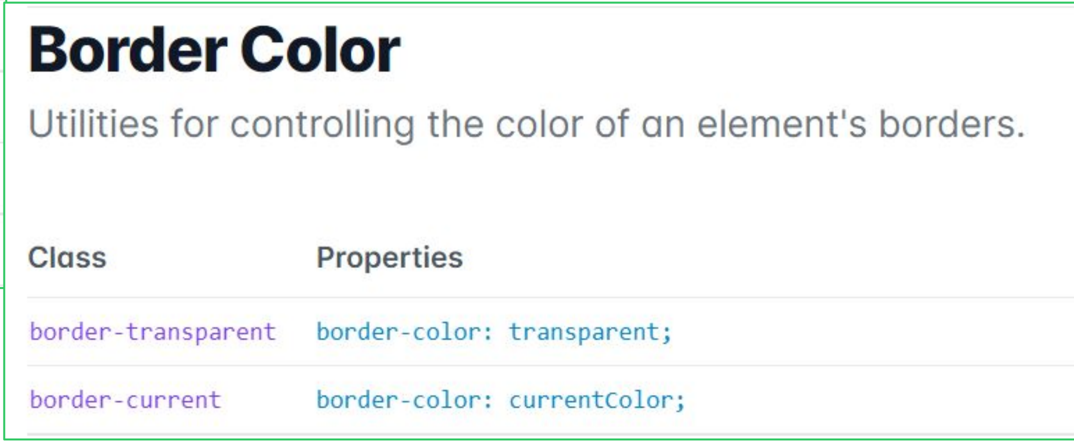


## Height

Utilities for setting the height of an element

| Class | Properties |
|---|---|
| `h-0` | `height: 0px;` |
| `h-px` | `height: 1px;` |
| `h-0.5` | `height: 0.125rem;` |
| `h-1` | `height: 0.25rem;` |
| `h-1.5` | `height: 0.375rem;` |
| `h-2` | `height: 0.5rem;` |
| `h-2.5` | `height: 0.625rem;` |

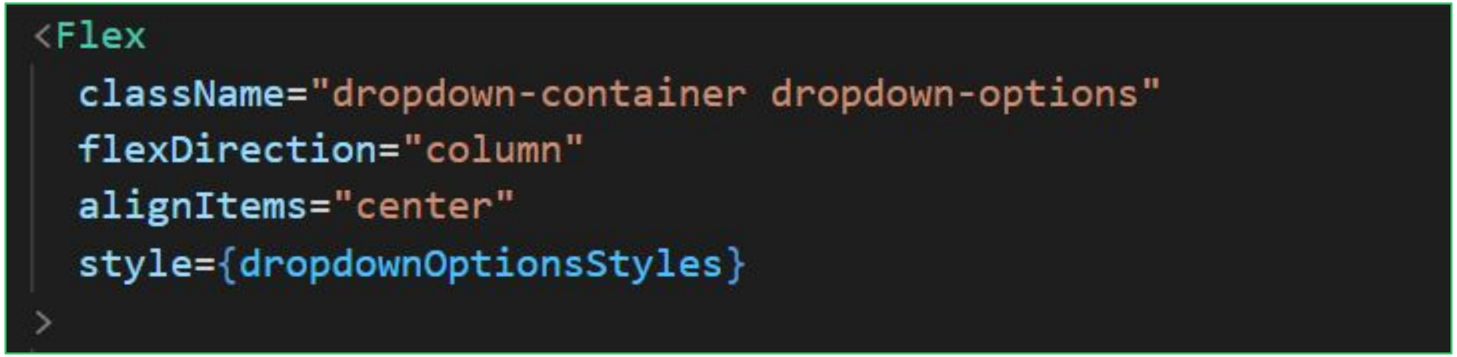
- Tailwind docs have the conversion details for every type of styling



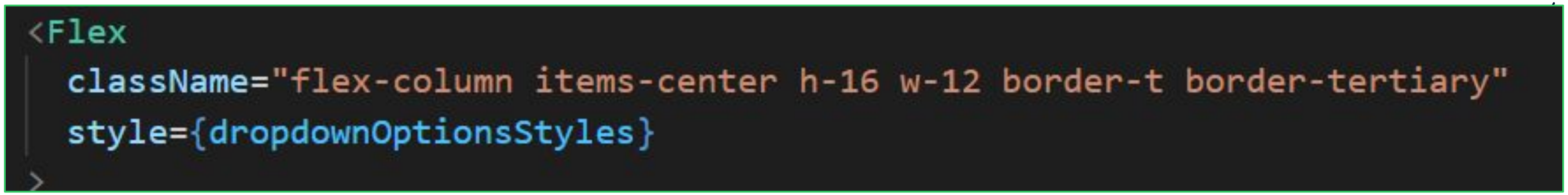
## Border Color

Utilities for controlling the color of an element's borders.

| Class | Properties |
|---|---|
| `border-transparent` | `border-color: transparent;` |
| `border-current` | `border-color: currentColor;` |



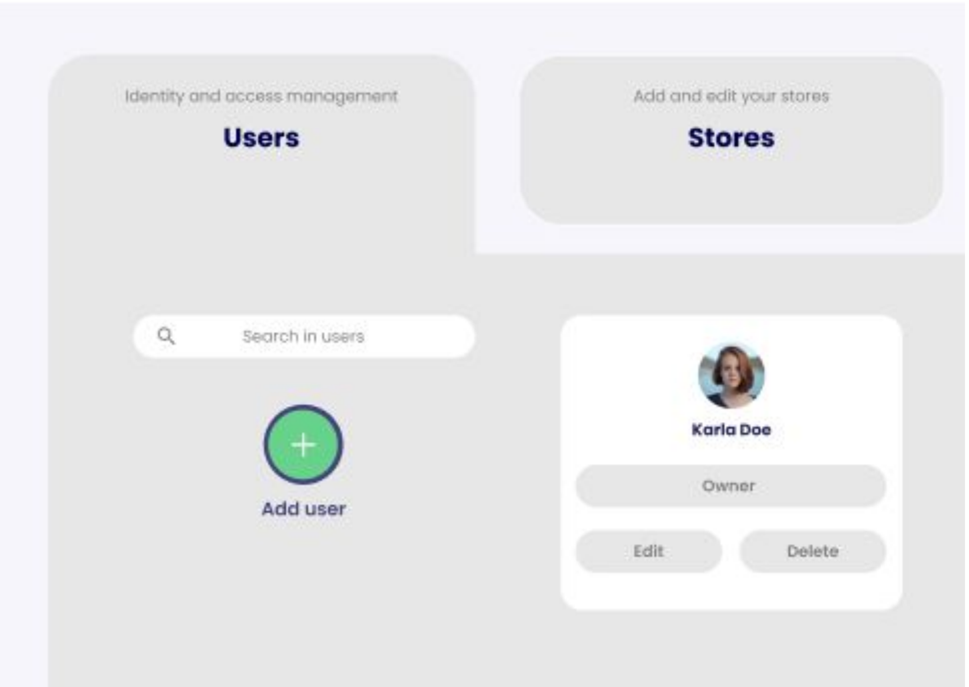
```
<Flex
  className="dropdown-container dropdown-options"
  flexDirection="column"
  alignItems="center"
  style={dropdownOptionsStyles}
>
```



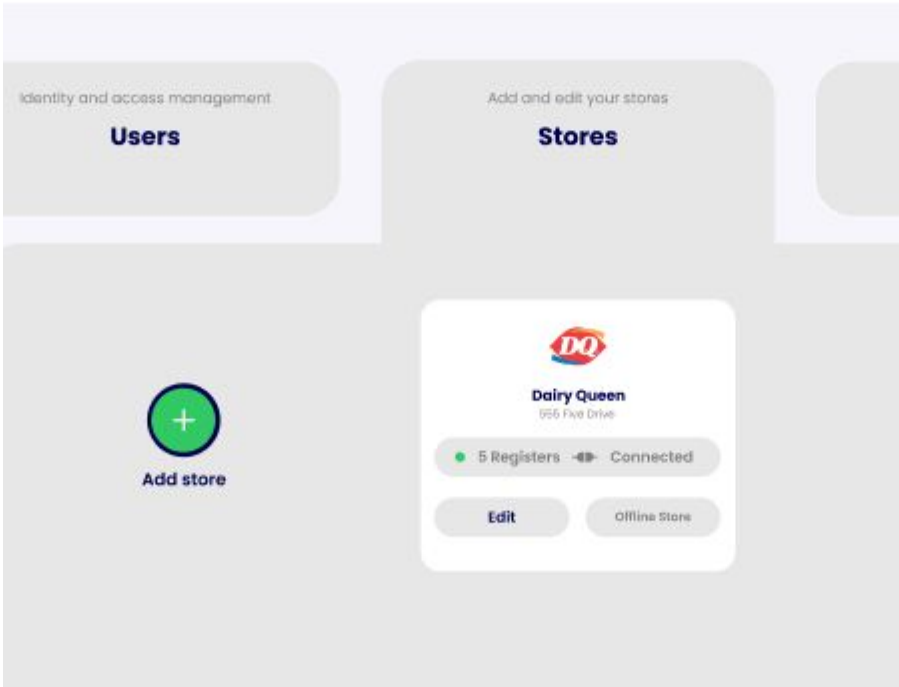
```
<Flex
  className="flex-column items-center h-16 w-12 border-t border-tertiary"
  style={dropdownOptionsStyles}
>
```

# CREATING RECURRING COMPONENTS

Users page - **Add user** button

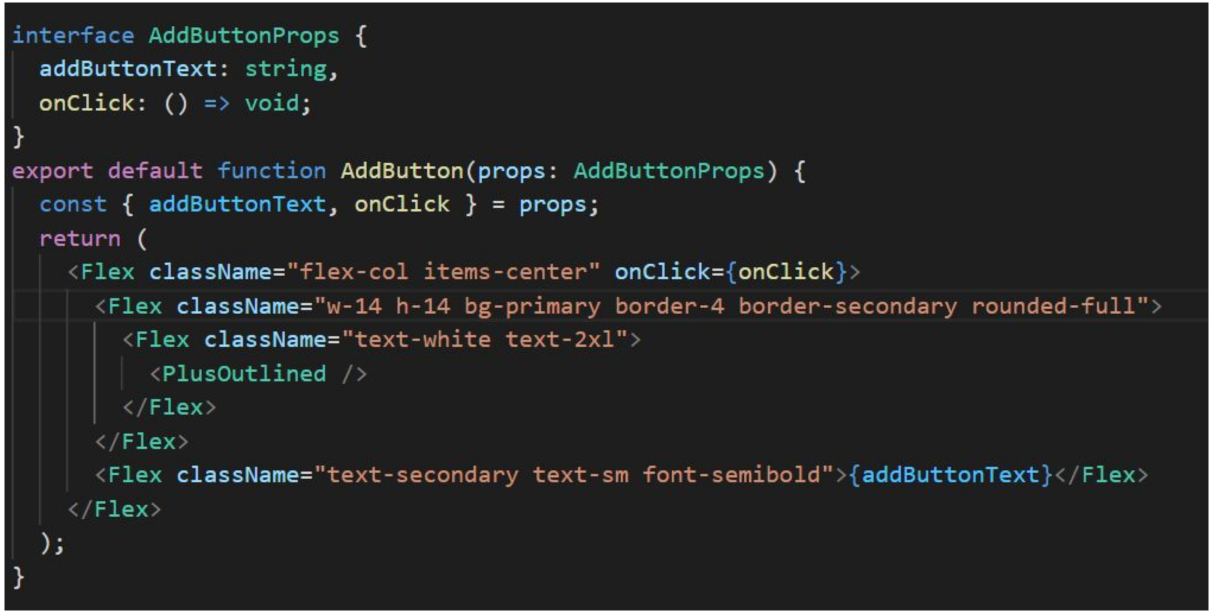


Stores page - **Add store** button



## CREATING RECURRING COMPONENTS

**addButton.jsx file**

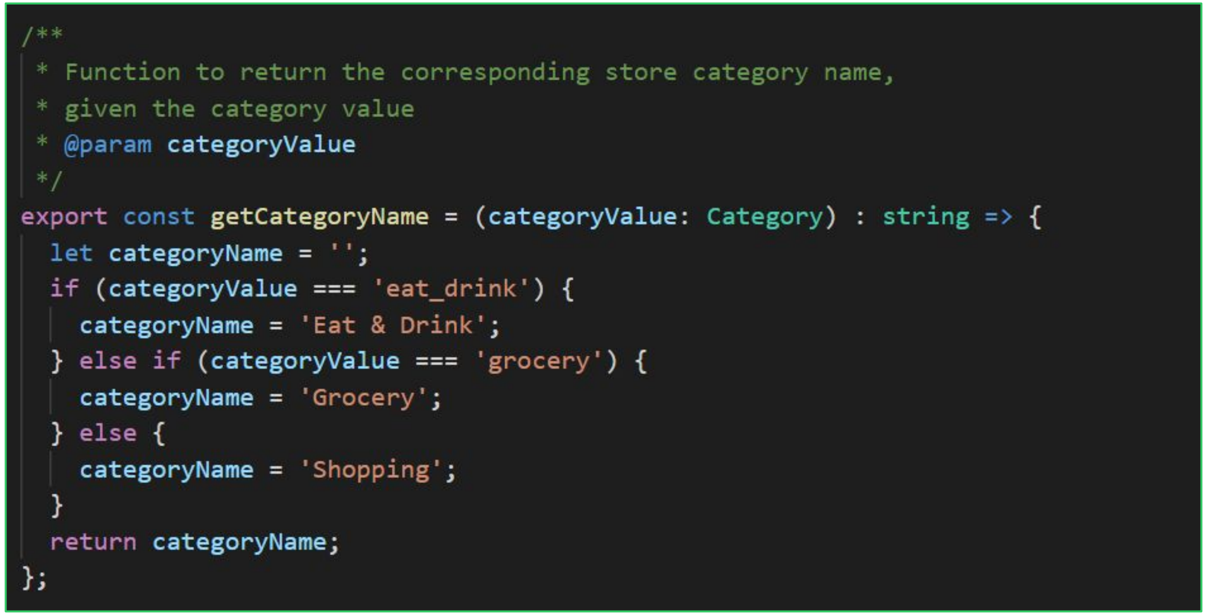


```
interface AddButtonProps {
  addButtonText: string,
  onClick: () => void;
}
export default function AddButton(props: AddButtonProps) {
  const { addButtonText, onClick } = props;
  return (
    <Flex className="flex-col items-center" onClick={onClick}>
      <Flex className="w-14 h-14 bg-primary border-4 border-secondary rounded-full">
        <Flex className="text-white text-2xl">
          <PlusOutlined />
        </Flex>
      </Flex>
      <Flex className="text-secondary text-sm font-semibold">{addButtonText}</Flex>
    </Flex>
  );
}
```

- All recurring components should be made first and then used by all programmers
- Components have props that make them unique, such as the **addButtonText**
- This way, the add button will have the same design, but is able to be customized

# USING UTILS FILES

**storeUtils.ts file**



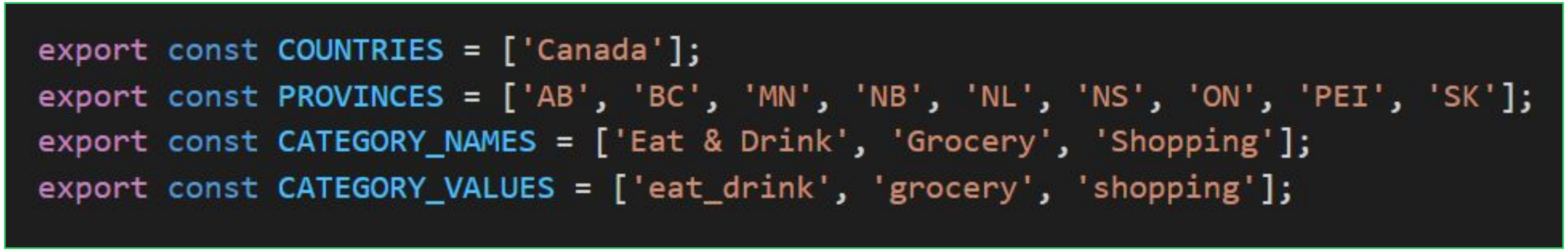
```
/**
 * Function to return the corresponding store category name,
 * given the category value
 * @param categoryValue
 */
export const getCategoryName = (categoryValue: Category) : string => {
  let categoryName = '';
  if (categoryValue === 'eat_drink') {
    categoryName = 'Eat & Drink';
  } else if (categoryValue === 'grocery') {
    categoryName = 'Grocery';
  } else {
    categoryName = 'Shopping';
  }
  return categoryName;
};
```

- Reusable functions should be placed in separate component specific utils file

- getCategoryName is used whenever a store interacts with the backend (when creating a store and viewing store information)

# CONSTANT VARIABLES IN UTILS FILE

**storeUtils.ts file**



```
export const COUNTRIES = ['Canada'];
export const PROVINCES = ['AB', 'BC', 'MN', 'NB', 'NL', 'NS', 'ON', 'PEI', 'SK'];
export const CATEGORY_NAMES = ['Eat & Drink', 'Grocery', 'Shopping'];
export const CATEGORY_VALUES = ['eat_drink', 'grocery', 'shopping'];
```

- Constant variables should also be placed in utils functions
- This way when multiple components use these variables, if the design changes, the variables can all be changed at once in the same place

## CONCLUSION AND REFLECTION

- To maintain consistency in styling, a themes and tailwind file should be created first, which can then be used by all programmers
- In order to avoid code blockages due to dependent components, create all components first, then assign full pages to programmers
- For most efficient re-designability, each components variables and functions should be stored in its own utils file

## THANKS FOR LISTENING!

## ANY QUESTIONS?

## REFERENCES

- "HYKE," Hyke Technologies Inc, 2021. [Online]. Available: https://hykeup.com/.
- “Tailwind CSS”, Tailwind CSS, 2021. [Online]. Available: https://tailwindcss.com/